University of Zurich[UZH]

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

SDN-based LoRa Mesh

Daniel Reiss
Zürich, Switzerland
Student ID: 15-720-329

Supervisor: Dr. Eryk Schiller
Date of Submission: October 22, 2021

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

ifi

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

The goal of this thesis is to implement a Long Range (LoRa) mesh with Software Defined Networking (SDN) - based mechanisms. Devices used in LoRa based Wireless Sensor Networks are often limited in their range, therefore, a LoRa network with mesh topology is a principle to increase the range of these devices with multi-hop communication. The main principle of SDN is to improve network management through a controller. The implementation is performed for Raspberry Pis, which have become popular in Internet of Things, and the E32-868T20D which is a cheap LoRa shield. The implementation is provided in the Python programming language, which entails that the code is more simple and readable compared to a lower level programming language. LoRa networks with mesh topologies have already been demonstrated for other devices and programming languages; therefore, the goal of this thesis is to show that SDN-based mechanisms can benefit a LoRa mesh and to provide them as extensible software.

Das Ziel dieser Bachelorarbeit ist die Implementierung eines Long Range (LoRa) Mesh mit Software Defined Networking (SDN) - basierten Mechanismen. Geräte, die in LoRa-basierten Wireless Sensor Networks verwendet werden, sind oft in ihrer Reichweite begrenzt. Daher ist ein LoRa-Netzwerk mit Mesh-Topologie eine Möglichkeit, die Reichweite dieser Geräte mit Multi-Hop-Kommunikation zu erhöhen. Die Grundidee von SDN ist die Verbesserung der Netzwerkverwaltung durch einen Controller. Die implementierte Lösung ist für Raspberry Pis, die im Internet der Dinge populär geworden sind, und für das preiswerte LoRa-Modul E32-868T20D. Die Implementierung erfolgt in der Programmiersprache Python, was zur Folge hat, dass der Code im Vergleich zu einer low-level Programmiersprache einfacher und besser lesbar ist. LoRa-Netzwerke mit Mesh-Topologien wurden bereits in low-level Programmiersprachen implementiert. Das Ziel dieser Arbeit ist es zu zeigen, dass SDN-basierte Mechanismen für ein LoRa-Mesh von Vorteil sein können, und sie als erweiterbare Software bereitzustellen.

Acknowledgments

I want to thank everyone involved in this project, especially Dr. Eryk Schiller for supervising me. His experience and expertise enabled this thesis in the first place and his valuable feedback and support had a direct impact its outcome. Further, without the support of Daniel Meyer, Marcel Schiess, Sascha Giger, Mikail Douis, Herbert Bachofner and Filip Batur the deployment across the buildings of University of Zürich (UZH) would not have been possible. Moreover, I want to thank Prof. Dr. Burkhard Stiller for the opportunity and support to do this thesis at the Communication Systems Group (CSG) of the UZH. Special thanks go to Lloyd Rochester for his work in the ebyte-sx1276 Library. His contribution allowed me to focus much more on the high-level aspects rather than working on the communication between the hardware parts.

I also want to thank my family and friends for their support during this time.

Contents

Chapter 1

Introduction

1.1 Motivation

Research in the area of communication protocols for the Internet-of-Things (IoT) is getting more and more attention. Technologies such as Low-Power Wide Area Networking (LPWAN) offer communication which satisfies long range and low power requirements [1, 2]. The most adopted Medium Access Control arguably is the Long Range WAN (LoRaWAN) used by various operators such as The Things Network (TTN). It promises ubiquitous connectivity in outdoor IoT applications while keeping network management and structures simple [2].

LoRa has evolved into an interesting technology for smart sensing in IoT [3]. Based on the Chirp Spread Spectrum (CSS) modulation and a simple link layer called LoRaWAN, it defines a specific radio physical layer (PHY) [4]. The LoRa PHY operation depends on the following parameters: (i) Bandwidth (BW), (ii) Spreading Factor (SF), (iii) Coding Rate (CR). Higher SF leads to lower bit rate and lower Bit Error Rate (BER) and lower SF leads to higher bit rate and higher BER. The Coding Rate defines the ratio of the redundant information for the Forward Error Correction (FEC). The CSS modulation of LoRa results in high sensitivity enabling transmissions over long distances. It provides a range of hundreds of meters indoors and up to several kilometers outdoors [5]. For example outdoors, there is a less than 10% loss rate (i.e., the radio between dropped packets and packets sent in the network) over a distance of 2 km for SF 9-12, and a more than 60% loss rate for SF 12 over 3.4 km. The radio channel access of LoRaWAN is similar to ALOHA [4, 6]: a device wakes up and immediately sends a packet to a gateway (GW). The difference between LoRa and pure ALOHA is the packet length which is fixed in ALOHA and variable in LoRa.

In LoRa, an IoT device, e.g., Arduino [7], Raspberry Pi (RPI) [8], is equipped with a LoRa shield and uses an appropriate Software Application Programming Interface (API) to communicate with the LoRaWAN. The LoRa shield has a Semtech SX1276/SX1278 [9] radio transceiver compatible with the LoRa PHY manufactured by Semtech, which is used by the IoT device to send and receive packets. LoRa PHY is patented by Semtech and only a limited set of information about the chip design is known publicly. However, the SX1276/SX1278 chips manufactured by Semtech provide waveforms compatible with all other Semtech-based LoRa shields.

Typically, the LoRa PHY is applied in LoRaWAN where a star-of-stars topology is used. This means that the IoT devices send LoRa packets and a set of GWs receive them. The GWs in turn propagate the received information to a central network server as they have a network backbone. However, recently, a new mesh-based operation framework was proposed for LoRa [10], in which multiple nodes communicate directly with one another, rather than just sending packets to a GW. The LoRa mesh uses precise time synchronization to allow for coordinated transition between sleep mode and wake up within the transmission interval. This causes enhanced life-time of battery powered IoT devices. Furthermore, a tree-based routing is used in the mesh to allow for packets being exchanged in the network in a multi-hop manner, where every transmitting node can also become receiver and a forwarding node.

In the past, solutions based on Software Defined Networking (SDN) for Wireless Sensor Networks (WSNs) of mesh topology have been proposed. SD-Wise [11] is an SDN mechanism especially designed for WSNs. The management of a WSN is simplified through the separation of the control and data planes in SD-Wise. The main goal of this thesis is to specify, implement and deploy SDN-based mechanisms (i.e., inspired by similar examples from WSNs) for LoRa mesh, which could allow for centralized control of duty cycles, routing and network management.

1.2 Description of Work

This work is built upon the LoRa mesh framework [10]. First, a couple of use-cases for the SDN-based LoRa mesh were described and the initial architecture of the SDN-based LoRa framework were worked out.

Second, the environment meaning the RPI boards, RPI Operating System (OS), and the SX1276-based dongles attached to the RPI through the Universal Serial Bus (USB) were organized and prepared. Third, the hardware architecture of the IoT device had to be

worked out. Once the hardware architecture was complete, the OS on the devices were configured and the system was equipped with an API for the SX1276 chip. This allowed for sending and receiving LoRa packets between two immediate neighbours. In addition, it was demonstrated, that the two nearby devices communicate with the help of LoRa transceivers (i.e., LoRa PHY). The next step was to demonstrate a successful multi-hop communication. At this point the multi-hop transmission is of static nature meaning hard-coded forwarding. To this end, addressing was introduced in the network (i.e., every node received a unique identifier), the format of data plane messages exchanged was specified, a static routing mechanism was developed (i.e., routing tables, packet forwarding on the device, etc), static routes were configured on all devices involved in a multi-hop transmission and a successful multi-hop transmission was demonstrated within this LoRa network.

After verifying static routing, dynamic routing had to be enabled in the network. Therefore, a specific routing protocol was selected. All routing-related communication primitives such as routing protocol had to be specified, implemented and deployed. In addition, routing tables had to be adequately populated allowing for a plug-and-play multi-hop commutation in the LoRa mesh network. The selected routing protocol had to enable either tree based routing or a shortest path routing in the LoRa mesh network.

The basic routing should have allowed for the connection between the LoRa mesh nodes and an SDN controller (e.g., OpenDayLight, ONOS, Ryu, POX). This would have enabled more fine-grained routing in the network through exchanging enhanced information about link quality. The Path Computation Element (PCE) of the SDN Controller would have computed the optimal routes in the LoRa mesh according to the selected algorithm (e.g., Dijkstra's Shortest Path Algorithm) using important link metrics. After computing the optimal routes, the SDN controller would push the computed routes in the system and propagate them to all the nodes in the LoRa mesh network, which translate optimal routes into the installed flow rules. It is worth noting that the transmission parameters of the LoRa PHY should be optimal in order to guarantee good performance of multi-hop delivery in the network.

Finally, the test-bed of a couple of LoRa mesh nodes and the SDN controller was deployed. The performance of the developed mechanism was evaluated using selected use-cases with a set of Key Performance Indicators (KPIs).

1.3 Thesis Outline

Based on the mentioned aspects, the structure of this document is built as follows. Chapter 2 is dedicated to the related work concerning LoRa, LoRa with a mesh topology and the SDN relevant aspects of the work. The end product is categorized and justified based on existing solutions in this chapter as well. In chapter 3 the architecture is explained. This includes explanation and justification of the chosen hardware and software components. Chapter 4 explains and discusses the implementation of the SDN-based LoRa mesh. The final prototype is evaluated in chapter 5 where Key Performance Indicators are measured and discussed. Finally, chapter 6 concludes the thesis with a summary and conclusions. It also includes limitations and possible future approaches.

Chapter 2

Related Work

This chapter deals with the related work in the context of this thesis and shows the current state of knowledge regarding mesh topology in LoRa. The first two sections explain the concepts IoT and WSNs while section 2.3 explains LoRa as a technology and separates LoRa and LoRaWAN. Section 2.4 is about LoRa with mesh topology and gives a brief overview of different approaches to use a mesh topology in LoRa. Finally, the section 2.5 is about various SDN controllers and SD-WISE, which is an SDN controller especially designed and proposed for Wireless Sensor Networks.

2.1 Internet of Things

The Internet of Things (IoT) [12] indicates things that typically were not connected to the internet or to other devices. Nowadays, these devices are organized as networks or are directly connected to the internet. A device that measures the temperature in a truck is an example. The device periodically registers the temperature and sends a message to a centralized endpoint, where the data can be analyzed. To this end, the temperature sensor needs to be connected to either the internet or to another device that can access the internet.

2.2 Wireless Sensor Networks

A Wireless Sensor Network (WSN) [13] is a network with the purpose to monitor some conditions of the environment. The collected data is wirelessly and automatically sent to

an endpoint without any human interaction. WSNs are heavily related to IoT as WSNs are a consequence of IoT growing a lot recently. The example above could be extended with more temperature sensors that are wirelessly connected to the board computer. In such a case, the temperature sensors would wirelessly transmit the data to the board computer, which forwards it to the endpoint for analysis, therefore connecting WSNs to IoT. Expanding the idea of monitoring the environment and automatically sending the gathered information to a central place can have an innovative effect on a lot of areas. It is important to note that devices in Wireless Sensor Networks (WSNs) are typically system-on-chip embedded, located in hard-to-reach locations and power constrained.

2.3 LoRa

LoRa is a communication technology used in IoT. It is a closed proprietary product owned by Semtech and offers a secure, affordable and energy efficient radio technology used in Long Range Wide Area Networks (LoRaWAN) [14]. The non-profit organization LoRa Alliance with more than 500 Members, e.g., Semtech, Cisco, IBM, Swisscom, all over the world has specified LoRaWAN [15]. The radio frequency modulation makes it useful for low-power wide area networks (LPWANs) [4]. The name LoRa is referencing the long range data links this technology makes possible: with line of sight, the devices can reach a range up to 15 kilometers in rural areas and up to five kilometers in urban areas [4]. The frequencies used by LoRa are within the license free ISM band (Europe 433/868 MHz, USA 915 MHz, Asia, 430 MHz). Because LoRa applies frequency spreading, the interference is relatively low. The communication between LoRa devices takes place on different channels with data rates ranging from 0.290 to 50 kbit/s.

2.3.1 Modulation

The ISM band is located in the 902-928 MHz and 863-870 MHz bands in the USA and in Europe. Therefore the LoRa signals in the ISM bands are composed of so called chirps modulated using the Chirp Spread Spectrum (CSS) technique [16]. Furthermore, the usual LoRa Bandwidth (BW) is 125 kHz and 500 kHz in Europe and North America respectively [14]. By multiplying the data signal with a spreading code or a chip sequence the LoRa processing gain is introduced in the RF channel. The frequency components of the total signal spectrum are increased by increasing the chip rate. This spreads the energy of the total signal across a wider range of frequencies, which allows the receiver to discern a signal with a lower signal-to-noise ratio (SNR) [4]. Spreading Factor (SF)

describes the amount of spreading code applied to the original data signal. The SF ranges from SF7 to SF12, a total of 6 SFs, and the larger the SF, the further the signal can travel and still reach a receiver without errors [3]. It is important to note that a higher SF results in longer time on air (TOA), reduced payload size and therefore, lower bit rate [3].

2.3.2 LoRaWAN

While the LoRa physical layer enables the long-range communication link, LoRaWAN defines the standardized wireless network's system architecture and communication protocol [3]. The protocol and network architecture influence determining the battery lifetime of a node, the quality of service, the security, the network capacity and a variety of applications served by the network [3]. LoRaWANs allow single wireless hop communication between end-devices and a central gateway and therefore form a star-type network topology. ALOHA random access organizes data transmissions, which has high transmission efficiency under ideal conditions including line-of-sight, low SF, sufficient signal strength and short TOA [10]. However, the sparseness of sending packets without 'listen-before-talk' and once in range-critical scenarios may lead to high packet loss. By adapting SF, CR, BW, and transmission power these conditions can be compensated, but there will always be an inevitable tradeoff between TOA, the energy consumption and the possible data output and these parameters [10].

2.4 LoRa Mesh

In [2], Adelantado et al. simulate and discuss limitations regarding capacity in LoRaWAN networks due to densification and duty-cycle restrictions. Facing these restrictions, [17] reported that packet delivery ratio (PDR) was reduced to 25% due to packet collisions in a simulation of a large-scale application with very high node density. Further, [18] adapted an asynchronous time division multiple access (TDMA) with a separate wake-up radio channel for LoRaWAN and Reynders et al. [19] propose adapting LoRaWAN networks with lightweight scheduling.

Finally, Lee and Ke [20] tried overcoming the low PDRs with proposing a mesh approach. In their study, they implemented a LoRa wireless mesh network based on Semtech's SX1278 transceiver operating at 430 MHz. Their test-bed is a proof of concept deployed on their university campus, in which the gateway controls the medium access by periodically querying data from every joined node. In order to avoid collisions all nodes are

continuously in receive mode. However, this means this study does not focus on low power consumption or time synchronization. The authors argue that a LoRa mesh network is a tradeoff between reducing the number of nodes that can be served in a given time interval and in favour of longer range with multiple hops [20].

In a different approach, the work in [21] adapted the routing protocol Destination - Sequenced Distance Vector (DSDV) to LoRaWAN. Here, the nodes are classified as either a non-energy constrained routing node or a leaf node. The proposed solution was tested in a prototype system in linear and bottleneck scenarios and the authors conclude it being feasible even for the duty cycle limitations.

In the article [22] the authors propose improving the performance of multi-hop LoRa networks using networking cluster based on the spreading factor. A tree-based SF clustering algorithm (TSCA) assigned the nodes to several subnets, where multiple SFs are used. After testing the solution with practical scenarios and simulations, the authors concluded that an efficient multi-hop LoRa network needs to consider SF allocation.

With the focus on monitoring underground infrastructure, Ebi et al. [10] presented a new architecture to LoRaWANs: Sensor nodes form a LoRa mesh network by sending data to a repeater node acting as a sink for sensor nodes. While the repeater node communicates directly with the gateway, it can still be a battery-powered device and does not need an internet connection. The repeater node forwards the packets to the gateway and the sensor nodes act as routers in the mesh network. In this study, the authors conducted a field test with over 60 nodes and the results showed an improved PDR for underground devices [10]. However, the analysis did not focus on energy consumption.

Cotrim and Kleinschmidt [23] classify nodes that perform simple tasks such as repeating packets as relay nodes. Since this approach is much simpler and more lightweight than routing, it might be more feasible for LoRaWAN due to the resource constrained devices. Cotrim and Kleinschmidt [23] define relay nodes as a sink for a mesh network as presented in [10] in addition to the repeating mechanic.

In order to increase coverage of a mesh, [24] present a new device called e-Node, which is a multi-channel relay node. In further work [25] the authors tested the e-Node in an industrial scenario. They concluded their solution could enhance the coverage of the network with high reliability of packets sent by end-devices located at a far distance. Further, Tanjung et al. [26] propose opportunistic forwarding, opportunistic transmission and network coding as a multi-hop technique. The relay node only repeats packets that have not been received by the gateway. This study was a simulation and it resulted in a similar PDR compared to simple multi-hop. It required fewer transmissions, nonetheless.

Regarding the energy efficiency, Ochoa et al. [27] analyzed the star topology of a LoRaWAN and the mesh topology. Furthermore, the authors present an analysis to find the optimal relations between distance and the LoRa transmission parameters such as spreading factor, transmission power, coding rate and bandwidth. They concluded that the relation depends on the distance between the gateway and the sender. Similarly, [28] discuss a model that analyzes energy consumption in multi-hop and single-hop LoRaWAN networks. They used simulations with MatLab, based on which they concluded that nodes closer to the gateway consume more energy than nodes further distanced from the gateway. On the other hand, nodes near the gateway consume less energy in the single-hop scenario. Further theoretical work [29] discusses various possible multi-hop setups for LoRa networks with three hops to the gateway. They suggest that there are mesh topologies which could enhance PDR and energy consumption when comparing them to conventional single-hop LoRaWAN.

2.5 SDN

Software Defined Networking (SDN) has emerged in the past decade and it's architecture is manageable, dynamic, adaptable and cost-effective which makes it ideal for modern applications that are of dynamic nature and rely on high bandwidths [30]. The SDN architecture decouples forwarding functions and network control allowing for the network to be directly programmable and in general, SDNs usually include a controller, that allows for centralized network management including monitoring of status, health, events and alerts of devices and administration of hardware [30].

The first well established SDN solution is OpenFlow [31] which enabled researchers to run experimental protocols in a network they use every day. OpenFlow is designed for wired networks and it is based on an Ethernet switch. Further, Open Flow has an internal flow-table with a standardized interface to remove and add flow entries. McKeown et al. [31] put forward Open Flow as a means for researchers to run experiments regarding protocols in their usual networks and they suggest that as a compromise, vendors do not have to expose the internals of their switches, because the researcher can use the SDN components.

There are also established SDN solutions for wireless infrastructured network domains [32]. The first adaptation of the SDN paradigm to wireless networks has been attempted recently. Sensor OpenFlow [33] tried to implement an SDN protocol for wireless sensor networks. It uses the basic OpenFlow structure, i.e., nodes maintain a flow table in a stan-

dardized format. WSNs are usually energy constrained and Sensor OpenFlow supports in-network processing to accommodate this need.

Software Defined Wireless Network (SDWN) [34] addressed problems that arose with Sensor OpenFlow. It used less standardized flow table entries which lead to higher efficiency regarding resources and energy. In order to enhance energy efficiency even more, it also enables maintaining duty cycles.

Finally Software Defined Wireless Sensor Network (SD-WISE) [11] builds on top of SDWN and extends it with states. SD-WISE enables nodes to decide on forwarding decisions without asking the controller first. This increases the efficiency and makes the whole network faster [11].

SDN Controllers

OpenFlow is currently the most established SDN protocol. In the early days of SDN there have been adaptations such as Floodlight, NOX [35] and Beacon [36], which were centralized controllers. These solutions have later been replaced by decentralized solutions in order to fulfill the needs of today's large scale of SDN applications.

One of these distributed solutions is OpenDaylight [37]. It was the reaction to the centralized approach and distributed functionality regarding network management to several instances. It was still not completely distributed, but a first step into that direction. ONOS [38] is another approach in a distributed SDN architecture. While being based on Floodlight, it supports a layered architecture that can be extended to integrate devices and protocols other than OpenFlow.

2.6 SD-WISE

As mentioned before, SD-WISE [11] brings SDN to Wireless Sensor Networks. It is based on previously discussed SDN controllers, such as ONOS and SDWN. Focusing on the three main aspects related to SDN, Network Function Virtualization (NFV) and in-device security technologies, it stands out from other solutions. SDN is stateful, i.e., a node stores information about its state, which is modifiable by actions [11]. Furthermore, SD-WISE differentiates from previous solutions such as OpenFlow, as the Rules are more flexible, meaning that the node has more power to decide on information in the packets. Lastly, SD-WISE focuses a lot on energy-efficiency as the authors argue that the most

important resources in WSNs is energy. Therefore, SD-WISE controls duty cycles and transmission power, which have not been considered in OpenFlow, as it is not a major problem in infrastructured networks [11].

SD-WISE not only introduces states but also NFV to WSNs. Using NFV enables decoupling network services form the hardware running it. This means that SD-WISE uses ONOS to take advantage of running an Operating System (OS) on the sensor nodes, that can execute small programs received from the controller. In the third aspect, SD-WISE uses the Trusted Platform Module to secure the device, i.e., validate the rules, services and OS installed.

SD-WISE differentiates between SD-WISE nodes and SD-WISE controllers, SD-WISE nodes being wireless sensor nodes that support network function virtualization and implement the software defined networking approach. Thus, WISE Flow Tables determine the forwarding behaviour of SD-WISE nodes. When a node receives a packet, it checks in the WISE Flow Table in order to know what to do with the received packet. If there is an entry, the packet is treated accordingly. Otherwise, the SD-WISE node needs to request the information from the SD-WISE controller. The new rule is then inserted into the WISE Flow Table. It is important to note that both mentioned devices need to be aware of the path towards each other.

Furthermore, SD-WISE nodes are running on the device embedded OS, which contains all information related to the node and which is able to download and deploy application layer functions as specified by the SD-WISE controller [11]. The Trusted Platform Module ensures that the nodes are running on a trusted firmware which can control all peripherals of the node. A Recognized Authority is identified by the SD-WISE controller and is thus able to release directives onto the nodes. Comparably to current trends in SDN, the SD-WISE controller consists of several network applications and a network operating system, where the network applications dictate how packets flow, i.e., routing protocols.

Chapter 3

Design

The idea at the start of the thesis was to have a mesh, where each connection between any two nodes would be optimized, i.e., having the spreading factor, bandwidth and coding rate which has the lowest packet error rate or the highest packet delivery ratio. Setting these parameters allows to have optimal link quality between a node and a gateway. However, if these parameters are used in a mesh, it is a lot more complicated. Because the receiving end needs to have the exact same parameters as the sending node, it is very complex to manage a mesh, where each link has different parameter settings.

Therefore, a lot of LoRa shields reduce the parameters to a selection of predefined channels. This means that only a channel can be selected on the module and this channel corresponds to a selection of LoRa transmission parameters. For example the modules used in this project have the parameter Air Data Rate (ADR), which is a combination of the three parameters.

Due to such configurations, it was decided to have a mesh on one of these channels. This allows every node to receive and send with the same configuration and simplifies setting the configuration dynamically, as only one value has to be set instead of three values individually. Further, the concept uses the lowest ADR, i.e., the highest spreading factor, as a rendez-vous channel, because SF12 has the highest range. This enables packets on the rendez-vous channel to have the highest range possible, which is less important in the actual mesh, but has more impact in the rendez-channel. The rendez-vous channel is a widely used approach in WSNs and thus, it is applied in this project to allow nodes to join an existing mesh [39]. The other five channels are used for the LoRa mesh. The main idea is to allow for dynamic plug and play, i.e., let nodes join, and to further alter the ADR with a controller. The controller is a regular mesh node with additional power. It is able to analyze the state of the mesh and to decide on increasing or decreasing the

channel, i.e., ADR. In order to be able to do that, it needs to be accepted as controller by the other nodes. The main task of the controller is to find an optimal channel for the mesh, and to make sure all nodes find themselves on the optimal channel to be in the mesh.

3.1 States

A node has different threads running and these threads depending in which state a node is in. A node is basically always listening for packets and it does not matter on which channel it is. However, it depends on the state, which packets a node is able to send.

3.1.1 Rendez-vous

The rendez-vous channel is designed for nodes that want to join the mesh. When a node is in the *rendez-vous* state, it is on the rendez-vous channel. This means it is not sending any messages on its own and only listening for incoming advertisements. These advertisements are either sent by the controller or any other node that is already in the mesh. Therefore, nodes end up going onto the rendez-vous channel when they drop out of the mesh for some reason, i.e., they do not detect a neighbour in a certain amount of time, which means that no other node was around on the same channel. These nodes wait for other nodes to hop on the rendez-vous channel where they advertise the channel of the active mesh (if there is one). If there is no active mesh at this point in time, the controller will decide on which channel the mesh will be built and advertise it to all nodes on the rendez-vous channel. Once a node on the rendez-vous channel receives an Air Data Rate of an active mesh, it responds with an ack and changes the channel to the advertised Air Data Rate in order to join the active mesh.

3.1.2 Joining

A node is in the state *joining*, after acknowledging the advertisement on the rendez-vous channel. The node is now on the same channel as the mesh and is sending hello messages to enable neighbours to detect it. A joining node is only accepted as a neighbour if a certain amount of hello messages is received over a defined amount of time. The reason behind this concept being that a node should only be accepted as a neighbour if enough packets are actually received from it. When the threshold is reached, the new node is accepted and changes its state from *joining* to *mesh*.

3.1.3 Mesh

The state *mesh* is reached, when a node has accepted another node as a neighbour. Therefore, it has an entry in its Link State Data Base (LSDB) and this triggers the routing protocol. In Section 3.2, this is explained in more detail. The routing protocol enables the node to have a list of neighbours and an LSDB, i.e., an up-to-date model of the LoRa mesh. The node updates the entries in the LSDB through exchanging link state advertisements with its neighbours. With the updated LSDB the routing algorithm computes the shortest paths for that node, and therefore enables multi-hop forwarding as the node knows the next hop for any destination of a given packet. Additionally, when a node is in the *mesh* state, it will go onto the rendez-vous channel to advertise the mesh for nodes that want to join. This happens periodically, and every node of the mesh does it. Even though the node is not in the mesh in that moment, the state is not changed meaning it is still in the *mesh* state. When a node is waiting to join on the rendez-vous channel, there is no way to know which node that is part of the mesh is in range of the waiting node. Therefore, every node of the mesh has to periodically drop out to advertise. Furthermore, the configuration shall be that only one node at a time is advertising, such that as many nodes as possible remain in the mesh. Otherwise, the multi-hop communication is threatened. Several actions are happening upon receiving hello messages, which is discussed in section 3.2.

3.2 Routing Protocol

The routing protocol implemented in this project is an adaptation of Open Shortest Path First (OSPF). OSPF is a link-state routing protocol and it is designed to be run internally for an autonomous system (AS) [40], which fits with this case. Each router, or in this case each node, stores the identical database that describes the autonomous system. With the foundation of this database, a routing algorithm outputs the routing table. Furthermore, the routing table serves as a look up table for the next hop of a given destination.

3.2.1 Hello Protocol

In a first step, OSPF's Hello Protocol [40] was adapted, as it is responsible for managing neighbours. It is important to note that it ensures a bidirectional communication in OSPF. Because it often occurred in this project, that links were not working bidirectionally, for example node A receives packets from node B, but node B does not receive packets from

node A in a similar ratio or not at all, this had to be adapted as the thesis progressed. Therefore, this part was altered such that each connection between any two nodes is treated as two individual connections in each direction. In this project, the adapted Hello Protocol is for broadcast networks. Thus, each node advertises itself by broadcasting hello packets periodically. In OSPF, the hello packets include the view of the autonomous system and the list of routers whose hello packets were received recently. This was adapted to the following structure: [255, Source, View of AS], i.e., not including the recently seen hello packets. The reason to exclude it being trying to minimize the packets in general. The first byte identifies the packet to be a hello packet, source is the address of the sending node and View of AS is the beginning byte of the hash of the database. It is important to note, that later on in the thesis, the structure of hello packets was extended with a counter to [255, Source, Counter, View of AS] as shown in Table 3.1 to enable calculation of a ratio of packets sent by node A and packets received by node B from node A.

As mentioned above, a neighbour is only accepted after a certain amount of received hello packets in a given time frame to ensure a connection that is good enough from the start. An accepted neighbour is added into a versioned list of neighbours. After discovering and accepting a neighbour, the adjacency is sealed by synchronizing the link state databases (LSDBs).

3.2.2 Synchronization of Databases

In a link state routing protocol, it is of paramount importance that the routers' LSDBs are synchronized. In OSPF this is simplified by requiring only adjacent routers to remain synchronized [40]. Therefore, each entry in the LSDB describes a set of link state advertisements (LSAs) that belong to the nodes database. Again, the original process from OSPF was adapted to achieve a better fit with a LoRa mesh. Instead of having a sequence after forming an adjacency, a more dynamic approach was used. The mentioned structure of hello packets enables comparing the hashes of the LSDB. Meaning every incoming hello packet includes the view of the autonomous system of the sending node. On the other hand, the receiving node can compare this view to his own view and act in requesting the LSAs from the adjacent node, if the views are different. The link state request (LSR) works in unicast, while the LSA is broadcast, i.e., every node seeing this packet can pick up the LSAs. This was adapted to lower the number of LSAs sent in total. Upon receiving a LSR, a node responds in sharing his LSDB entries as LSAs. LSAs follow the structure [254, key, version, neighbour 1, neighbour 2, ...] and LSRs have the structure [253, source, target]. The first byte identifies the packets to be an LSA or LSR, respectively. As the

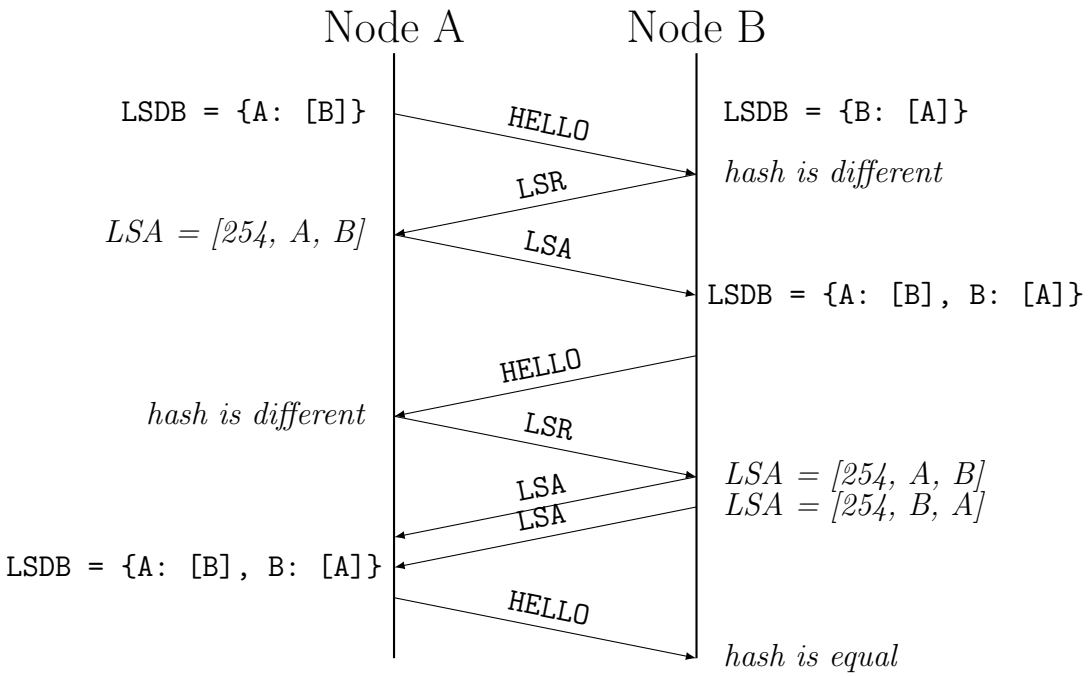

Figure 3.1: Communication diagram for two nodes synchronizing their LSDBs with Hello, LSR and LSA packets.

LSR is unicast, it includes the source node and the target node. The key in LSA represents an address of a node, followed by the list of neighbours which includes a version number at the beginning. This LSA structure not only allows the requesting node, but also all other nodes seeing this packet adding it to their own LSDB. Obviously, this only happens if the entry is not already there or the version is higher than the one of the entry they already own. This whole process guarantees synchronization of the LSDBs across the mesh and ensures that is finishes in a finite period of time.

Figure 3.1 summarizes the following explanation of exchanging packets for two immediate neighbours, i.e., when they have accepted each other as neighbours. At the end of the exchange, both nodes can run the routing algorithm to find the shortest path to the newly added node, and in case of other nodes already being in the LSDB at the start, try to optimize these paths as well.

It is important to note that the interval between hello packets dictates the pace of the synchronization. It can easily be changed to a different value. but for the evaluation in Chapter 5 this time interval was chosen to allow for better monitoring. Analogously, the number of nodes participating is set to five and needs to be changed, if tests are performed with a different number of nodes. This is due to the controller making decisions based on this number as explained in paragraph 3.2.3.

Table 3.1: Overview of all packet structures.

Main Packet	Bytearray
HELLO	[255, Source, Counter, LSDB_hash]
LSR	[253, Source, Target]
LSA	[254, Key, Version, Neighbour1, Neighbour2, ...]
Multi-Hop	[Next-Hop, Source, Destination, Payload]
Rendez-vous Advertisement	[255, 255, Air_Data_Rate]

Table 3.1 shows all types of packets. Every packet starts with an identifier, that allows determining the type of packet. Further, Hello and LSR packets include the Source, which is the address of the packets origin. This enables a receiving node to know the sender of the packet. As mentioned, hello and LSA packets are broadcast meaning they do not include a target, while LSR and multi-hop packets include one as unicast messages. From the structure of the LSAs it can be deducted that sharing an LSDB affords as many packets as there are LSDB entries. Because the payload of LoRa packets can be very limited, it was decided to use this approach rather than sending the whole LSDB in one packet. With a LoRa transmission with SF12, it would form a bottleneck of 52 bytes per packet, limiting the mesh to about a dozen nodes. Further, the LSAs include a Version, such that nodes can replace outdated entries with newer ones. On the other hand, the multi-hop packets include a next-hop and a destination, such that the next-hop is the forwarding node anywhere between Source and Destination. A packet has arrived at its destination when Next-Hop = Destination. How a node can find the Next-Hop for any given Destination is explained in Section 3.3 in more detail. All these messages are sent to the Unix Domain Socket and therefore over LoRa PHY as bytearrays.

3.2.3 Additional Features

Additionally, the routing protocol is extended with aspects that are key to grant a mesh with a decent quality. First, the routing protocol is designed to drop a neighbour, if no packet from it is received in a certain amount of time. It makes sense to drop these nodes, as the multi-hop communication is not granted at this point. On the other hand, a node that does not receive any packets in general, i.e., from any other node, drops out of the mesh itself. Then it goes back to the rendez-vous channel to try an join again, as mentioned in section 3.1. Furthermore, the controller needs to gather information about the link qualities in the whole mesh. Therefore, each node stores a percentage of received messages per neighbour and sends it to the controller periodically in a multi-hop manner. These numbers are based on the periodical hello packets as they can be accumulated through the included counter in the hello packets. Basically, a node stores the counter of

the first received hello packet and thus, it counts how many packets arrive and compares it to the counters of the more recent hello packets.

3.3 Routing Algorithm

The routing algorithm is used to calculate the optimal routes in the mesh. Every node is able to calculate these routes on their own in a distributed fashion. That means each node will calculate the optimal route to every other address in the mesh with the starting point being its own address. Therefore, every node has to calculate the routes on their own, as they cannot rely on other nodes, i.e., a controller, to inform them about routes. As mentioned before, every node stores all the links of the mesh in an LSDB, which serves as the basis for the routing algorithm.

In a first step, the LSDB is transformed into an adjacency matrix. It is important to note, that the mesh is a directed graph, meaning that a connection between two nodes is treated as two connections, i.e., from node A to node B and from node B to node A. Due to the physical layer being LoRa PHY, it can happen that node A receives packets from node B, but at the same time node B does not receive packets from node A. This can have multiple reasons, meaning that a directed graph has to be used.

The created adjacency matrix is the input for the routing algorithm. The implementation of the Shortest Path Problem Between Routing Terminals – Implementation in Python by Renuka Narayanan [41] was adapted. This implementation is based on Dijkstra's Algorithm that computes the shortest paths for a given start node. Due to all the edges in the graph being the same cost, the shortest path will have the fewest hops, also if there are multiple paths available. This translates to energy consumption, as having minimal hops means less power is needed.

Running the routing algorithm results in a routing table. The routing table is a simple hash table that stores the next hop for a given destination. Every node has a different routing table and is therefore responsible for maintaining it. Every time adjustments are made in the LSDB, i.e., a node joining or leaving, the routing table has to be updated, meaning that the routing algorithm is run.

3.4 Controller

In general, one of the participating nodes is defined to be the controller. The controller has some extra functionality that allows for central management of the mesh. Therefore, the controller decides the channel that is used for the mesh and from there on it decides if the speed can be increased or has to be lowered. In order to make these decisions, the controller uses its LSDB. In more detail, the controller periodically analyses the LSDB.

As shown in Algorithm 1, the controller first checks the number of LSDB entries. Because the controller knows how many nodes are trying to participate, it also knows how many nodes were not able to detect neighbours to join the mesh. It is the controllers responsibility to make sure that all nodes are participating in the mesh, which is the most important Key Performance Indicator. A nodes LSDB entry not showing up in other LSDB means that its packets are not received by any other node and therefore it is not accepted as a neighbour. In case of any nodes missing in the LSDB, the controller decides to lower the air data rate. Otherwise, the controller continues in checking every node for incoming and outgoing edges, as a node needs to have both in order to participate in the mesh. To check these conditions, the controller iterates though every LSDB-key and checks if this address occurs in the LSDB-values. If one key does not show an occurrence in the values, it means the packets of this node are not received and the data rate is lowered. This is also to ensure that the controllers packets arrive at some nodes.

Algorithm 1 Controller analysing the state of the mesh

```
increase ← True
K ← LSDB.keys()
V ← LSDB.values()
if length(K) < nodes then
    decrease_speed()
else
    for i in K do
        if i not in V then
            increase ← False
            decrease_speed()
        end if
    end for
    if increase == True and stop_increasing == False then
        increase_speed()
    else
        Stay on current data rate
    end if
end if
```

However, the controller decides to increase the air data rate, if all of these conditions are

met. The idea is to increase it as far as possible and once a mesh does not form with all nodes anymore, the controller goes back to the last rate that was working well. The flag stop_increasing is set to false when decreasing the speed in order to signal the next iteration of analysis not to increase again.

When a decision is made to increase or decrease, the controller informs all the nodes in its LSDB to change air data rate in multi-hop manner. Algorithm 2 displays how the controller sends the newly decided air data rate to all nodes in the mesh. It is important to note that the mesh can have long paths, it takes some time to send all packets. On the receiving end, the nodes do not change the setting immediately. They remain on the channel for another 60 seconds upon receiving the new ADR such that they can still forward packets for other nodes in multi-hop manner.

Algorithm 2 Controller sending new ADR to all nodes in LSDB.

```
for destination in routing_table.keys() do
    next_hop ← routing_table[destination]
    msg ← [next_hop, myAddress, destination, ADR]
    send(bytearray(msg)
    time.sleep(2)
end for
```

Additionally, the controller receives the hello packets received ratios as mentioned before. This additional information, compared to the other nodes, enables the controller to analyze the state of the mesh in more detail. This allows the controller to identify the nodes, that are the reason for keeping a mesh on the lower data rates. The controller keeps the received values in a matrix similar to the adjacency matrix where an entry $A[i, j]$ is the percentage of received hello packets at node i from node j. Due to not all nodes being able to send these values to the controller most of the time, this approach had to be redesigned at a later point. At the beginning of the thesis, the aim was to use only this approach to let the controller determine the optimal mesh. But after deploying the first couple of nodes, the approach was not working and adjusted to the functionality shown in Algorithm 1.

$$A = \begin{bmatrix} 0 & 80 & 50 & 0 & 100 \\ 90 & 0 & 0 & 0 & 0 \\ 40 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 \\ 100 & 0 & 0 & 100 & 0 \end{bmatrix}$$

3.5 Scenarios

Based on the mentioned concept three scenarios were derived that enable testing the specified functionality and behaviour of the LoRa mesh.

Scenario 1: Initializing the LoRa mesh

The setup for scenario 1 is as follows:

- No active mesh
- Nodes are scattered across different channels

In this scenario, there is no active mesh at the start. The nodes are on different channels and are sending hello messages. Because they are not detecting any neighbours, they go onto the rendez-vous channel to get information on which channel the mesh is. On the rendez-vous channel they do not send any messages on their own. Since there is no mesh, the controller decides the channel which all nodes will change to. After changing the channel, nodes change to the joining state, i.e., they start sending hello messages again in order to initialize a mesh. Once a node starts having entries its LSDB the state changes to mesh.

In this scenario it is assessed, how many nodes are in the mesh when it is formed. Further, this scenario monitors the different channels, the time it takes to synchronize the LSDBs, and assesses all the packets sent to categorize them as Key Performance Indicators (KPIs). The outcome of this scenario is that all nodes find themselves on the channel that the controller decided and form a mesh on this channel.

Scenario 2: New node joins the LoRa mesh

The setup for scenario 2 is as follows:

- Active mesh on a given channel
- Joining node is on rendez-vous channel

In this scenario, there is a mesh already formed and a new node wants to join it. Thus, the new node goes onto the rendez-vous channel to wait for an advertisement of the mesh. Since nodes that are part of the mesh periodically drop out of the mesh to advertise it on the rendez-vous channel, the waiting node will be picked up.

This scenario measures the time it takes to synchronize all LSDBs, all the packets sent and it also monitors the rendez-vous and the mesh channel. The most important KPI is if the new node is able to join, therefore an increasing number of nodes in the mesh. The expected outcome of this scenario is that the new node is picked up through a node that is already in the mesh. If the new node is not detected by any node, it is expected that the air data rate is lowered.

Scenario 3: Reliable mesh

In the third and last scenario, all the nodes are already in a mesh. Further, the controller periodically analyzes the state of the mesh, i.e., it checks if every node has an entry in the controllers LSDB and if every node is accepted as an immediate neighbour by another node. Based on this analysis, the controller makes decisions to either stay on the current air data rate, to increase it or to decrease it. In addition, every node gathers a message received / messages sent ratio for every neighbour and sends it to the controller periodically and in a multi-hop manner.

On this scenario, the channels are monitored and the amount of packets sent is assessed. The outcome is expected to have a mesh on the highest air data rate while still having all nodes in it.

3.6 Use Cases

The approach developed in this thesis can be applied to several real world scenarios. As LoRa is often used in environmental monitoring, this setup can be used in this area. It is possible that such a setup would increase the range of some of the nodes, such that the nodes can be more scattered than in a regular setup. The multi-hop forwarding ensures that packets still reach their destination even though the source and destination cannot communicate directly to one another. The same idea can be applied to underground scenarios where the range of LoRa modules is a lot more limited than above ground. In this aspect, a LoRa mesh also enhances the range and allows for propagating packets in

a multi-hop manner especially for nodes that are deeply underground which would not reach the ground level on their own.

Since the nodes of this thesis consist of Raspberry Pis, additional communication channels are accessible. A Raspberry Pi has built in Bluetooth which can be used to connect devices such as laptops or smartphones to the Pi and therefore to the LoRa mesh network. In case of a disaster situation where regular communication channels are not reachable anymore or inaccessible, the LoRa mesh can be used for communicating across a region, where such a setup is already installed.

Chapter 4

Implementation

This chapter explains the implementation of the project. It gives a detailed explanation of hardware and software used in the thesis, describes the architecture of the nodes and explains the developed Python modules in more detail. Further, this chapter gives an overview of the deployed test-bed that was used to evaluate the implementation.

4.1 Hardware and Software

Having the goal to implement the concept mentioned in Chapter 3, it was decided to use the following hardware components. A node consists of:

- Micro Controller Unit: Raspberry Pi Model 3B or 4 [8]
- Serial Port Module: E32-868T20D incl. SX1276 LoRa Chip [42]

4.1.1 Raspberry Pi

The Raspberry Pi 3 Model B [8, 43] is part of the third generation of Raspberry Pi released in 2016. It has become very popular and is among the more powerful microcontrollers. Some highlights among the features of the Raspberry Pi 3B are

- Quad Core 64bit CPU with 1.2GHz
- 1GB of RAM

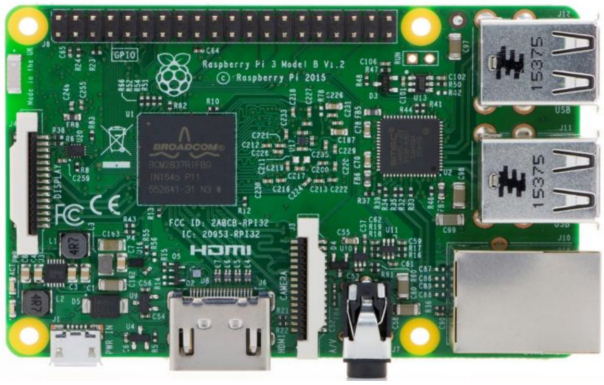

Figure 4.1: Raspberry Pi 3B [43].

- 100 Mbit Ethernet port
- on board wireless LAN and Bluetooth Low Energy
- one Universal Asynchronous Receiver Transmitter (UART) interface and
- four USB 2 ports

It made sense to use Raspberry Pis in this project, because the mesh nodes are not power constrained and need to be accessed for programming and testing when deployed. This is easily achievable with Raspberry Pis. The Raspberry Pis run on Raspberry Pi OS Lite (Version 2021-03-04), which include neither desktop or recommended software [44]. All the Pis are in a headless setup with Secure Shell (SSH) enabled in order to access them from another device for programming and monitoring. Figure 4.3 shows a node consisting of a Raspberry Pi 3B and the E32-868T20D with a LoRa antenna.

The development of this project required software to not only write and debug code, but also grant a connection to the Raspberry Pis via SSH. Since the Integrated Development Environment Visual Studio Code delivers both of these requirements, it was used in this project. Visual Studio Code allowed for accessing the Raspberry Pis via SSH to write, debug and run code directly on the Raspberry Pis from anywhere using the VPN of the University of Zürich (UZH-VPN). This required the Raspberry Pis to be setup to report their IP addresses periodically, which was achieved using crontabs. The crontabs also included a scheduled reboot once per day, granting optimal conditions on the devices.

4.1.2 E32-868T20D

The LoRa module used in this project is the E32-868T20D [42]. It is a wireless serial port module based on Semtech's SX1276 RF chip. It is working on 868MHz, has multiple

Figure 4.2: E32-868T20D [42].

transmission modes and is compatible with 3.3v and 5v. The transmitting power of this module is 20 dBm supporting a communication distance up to 3 km with an air data rate of up to 19.2 kbit/s. Table 4.1 shows an overview of the available channels of the module. ADR = 300 bit/s is used as the rendez-vous channel and all other ADRs are available to the controller to pick for the channel of the mesh. Table 4.1 gives an overview of the available values for ADR. It is important to note that the higher speed have lower SF and therefore lower range. Further, the rendez-vous channel at 300 bit/s has the highest SF and therefore the highest range. This has a direct impact on the joining mechanism, as rendez-vous packets have the longest range and are a lot more likely to be received by a waiting node.

Table 4.1: Air Data Rate corresponding to LoRa parameters according to [45].

Air Data Rate (bit/s)	SF	CR	BW (KHz)
300	12	4/5	500
1200	11	4/5	500
2400	11	4/5	250
4800	8	4/6	500
9600	8	4/6	250
19200	7	4/6	125

In order to access the LoRa module from the Raspberry Pis, the third party Library called ebyte-SX1276 was applied, as it allows for communication between the Raspberry Pi and the E32-868T20D, granting a low level connection between the hardware components. The ebyte-SX1276 Library is a project from Lloyd Rochester and it is designed to have a communication between a Raspberry Pi and the Ebyte E32 module. The Ebyte E32 module is the same LoRa module as the Delock 12591 which was used in this thesis, the only difference being the manufacturer of the module. Therefore, it was applicable for this thesis. The command line tool grants a low level connection between the hardware

Figure 4.3: Raspberry Pi 3B with the E32-868T20D attached.

components RPI and LoRa module, allowing the nodes to communicate with each other over LoRa PHY. Further, the library enables sending and receiving LoRa packets from the command line, which is ideal for checking connections. The project is written in C and only minor adjustments were made to it during the thesis. In addition to using it from the terminal, the library is able to be run as a Unix Systemd Service in the background. The blog posts [46] referencing the repository guide the user through the necessary steps to use the full potential of the library. After the standard installation, the tool provides Unix Domain Sockets as an interface. There is one socket for configuration and another one for transmitting. This interface allows users to access the LoRa module from any programming language, in this case Python, when running the tool as a background service. In other words, Python scripts access the E32-868T20D through the Unix Domain Sockets in this project.

4.2 Architecture

Figure 4.3 shows a node with the hardware parts wired up and Figure 4.4 gives a conceptual overview of the LoRa mesh. A node consists of a Raspberry Pi and a E32-868T20D. The Raspberry Pi sends a data packet for transmission to the E32-868T20D over the UART connection, which in turn sends the packet over LoRa PHY. This packet is then received at another node over LoRa PHY with its E32-868T20D, which sends it to the RPI over the UART.

Further, the RPIs contain the developed Python module of this thesis and the third party library E32 as shown in the Figure 4.5. The E32 manages the communication between RPI and E32-868T20D. It uses Unix Domain Sockets for transmission and configuration

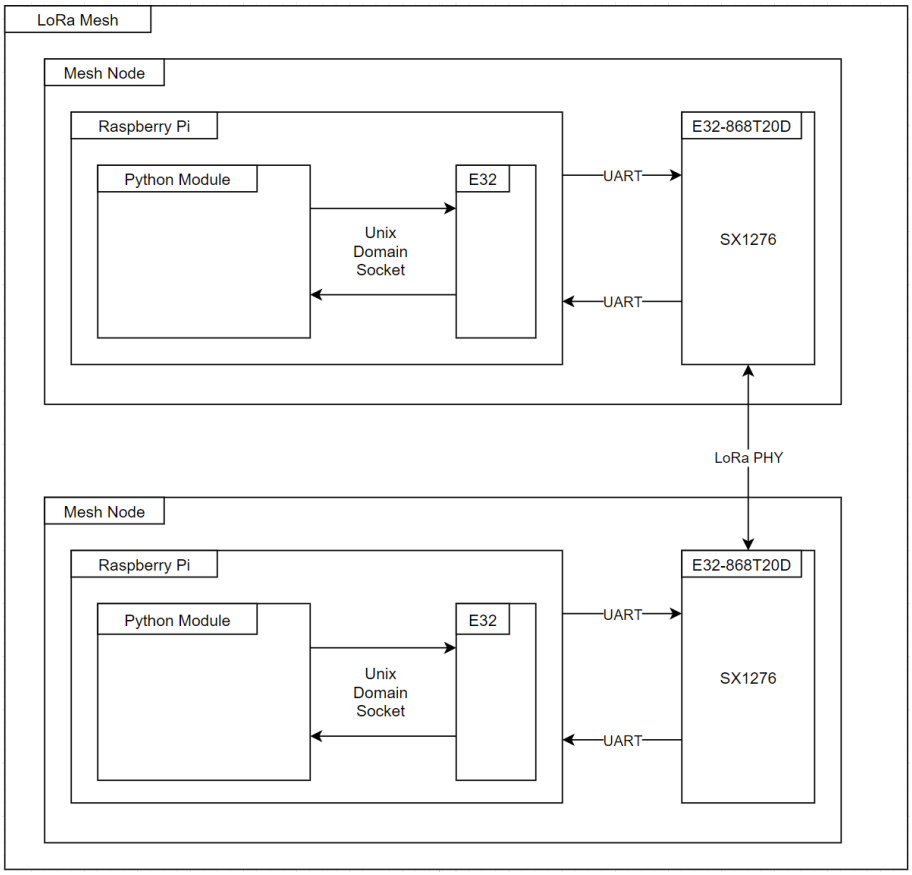

Figure 4.4: Components of the LoRa mesh.

via Python. Every packet sent or received by the RPI is going through the E32 library. The Python module developed in this thesis consists of three main components: the Routing Protocol, the Routing Algorithm and the Controller. The Routing Protocol does everything related to detecting neighbours and synchronizing link information. The Routing Algorithm uses this link information to calculate routes and the controller module manages the mesh. It is important to note that the controller module is on every node, but it is only activated on a node when it is assigned the role of the controller. This means that every node has the possibility to be elected and to execute controller mechanisms in such a case.

4.3 Python Modules

As mentioned in the previous section, the developed Python module of this thesis consists of the three parts Routing Protocol, Routing Algorithm and Controller. The overall performance depends on configuration details of these parts.

First of all, all three modules are dependant on the number of nodes. As the controller

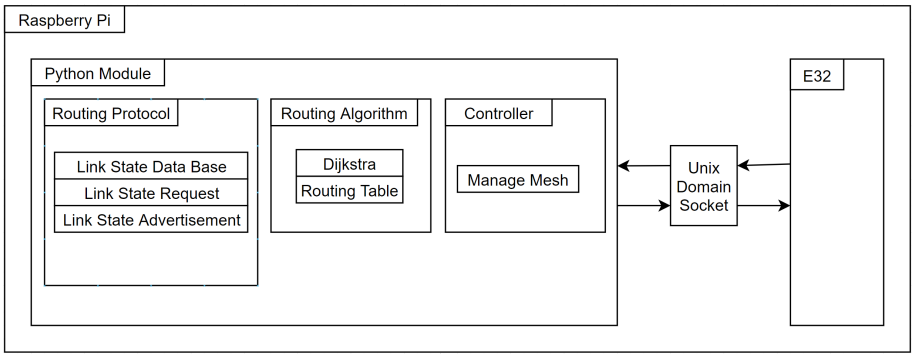

Figure 4.5: Components of a Raspberry Pi.

needs to know how many nodes are participating, the number of nodes has to be set. In fact, it is set on every node, because every node can be the controller.

The Routing Protocol is heavily dependant on the interval between hello packets. For the evaluation this was set to 60 seconds to allow for more tests. In a real deployment scenario this could be set much higher to have less overhead, or once the mesh is formed, the interval could be stretched. For the purpose of evaluation however, this is not applicable. Further, the mechanism to accept a node as immediate neighbour is dependant on the hello packets. As it is set to three out of five hello packets, it takes only a few minutes with the hello packet interval of 60 seconds. With a longer interval, this increases accordingly and might need to be reconsidered.

Moreover, running the Routing Algorithm discussed in Section 3.3 is linked to the LSDB. With the implemented configuration, the routes are newly calculated, every time the LSDB changes. To reduce overhead, it could be adjusted to only run when the LSDB is full or start running it as it is almost full. Especially with more nodes, this becomes a lot more expensive. In such case, it might be a good idea to take this decision away from the nodes and to let the controller decide when to run the routing algorithm.

The periodical behaviour of sending statistics to the controller is also based on the hello packets. Compared to the other mechanisms, this is based on the incoming hello packets. The ratio of received packets is updated every time a hello packet is received and sent to the controller after every five hello packets received from the same neighbour. The reason for this configuration is to have at least five hello packets received to have a decent starting ratio before sending the value to the controller, and to continuously update the controllers view of the mesh, such that the controller bases its decision on an updated state.

Lastly, the controller analyses the LSDB periodically. Due to varying convergence times, this is set to 15 hello packets sent, i.e., 15 minutes with hello packet interval set to 60

seconds. This is due to the fact that after 15 hello packets, the LSDBs have usually converged if they were converging. Otherwise, the controller should decrease the speed anyway and therefore, it does not matter if the LSDBs are still synchronizing or not.

4.4 Test Network

In order to test the implementation of the developed concept, a test network was deployed across the buildings of the University of Zürich. The budget for this project allowed to install five nodes. All nodes had to be connected to the UZH Virtual LAN in order to be accessible for programming and debugging. Five locations that fulfilled the requirements were secured as seen in figure 4.6.

Node Locations

As the nodes are setup above ground, factors such as obstructions or line of sight influence the reliability of the transmissions between any two nodes. Therefore, this section shows in which buildings of UZH [48] the nodes are deployed and further, which direction they are facing.

Table 4.2: Distance matrix of the five deployed nodes in meters. Underlined values mark line of sight. Distances obtained with Google Maps[47].

Node	BIN	THA	AND	TFA	Y13
BIN	0	170	140	1480	1840
THA	170	0	180	1370	1700
AND	140	180	0	1350	1710
TFA	1480	1350	1370	0	415
Y13	1840	1700	1710	415	0

Figure 4.7 shows the exact locations of the nodes in BIN, THA and AND. The node in BIN is located on the second floor in the room BIN 2.E.07, which faces south, while the node in THA is on the forth floor directly behind the window on the west side of the building. On the other hand, the node in AND faces towards the inner (and covered) courtyard. In addition, this node is only on the second floor out of five. A node is deployed in TFA behind the window in room 10.20, which faces north and is on the top floor as visualized in Figure 4.9. The last node is on the roof of Y13 on the Irchel Campus. It is in an outdoor case that is attached to a pole. In the Nebra IP 67 [49], this node, shown in Figure 4.8 is safe in any weather condition. The Figures 4.10 and 4.8 show the exact

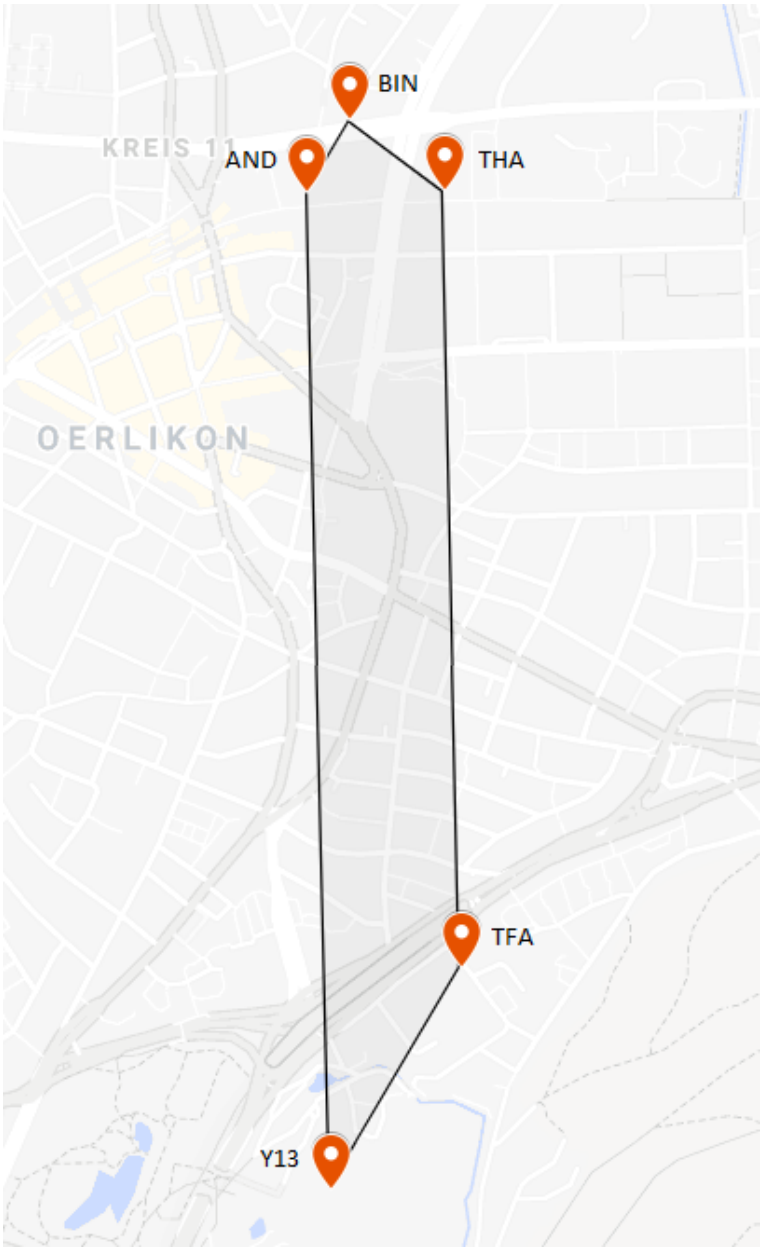

Figure 4.6: Overview of the five deployed nodes. Map created with Google Maps [47].

location and the outdoor deployment respectively. Further, the direct distances between all nodes is presented in table 4.2. It is important to note that there is only direct line of sight between BIN - Y13 and BIN - TFA.

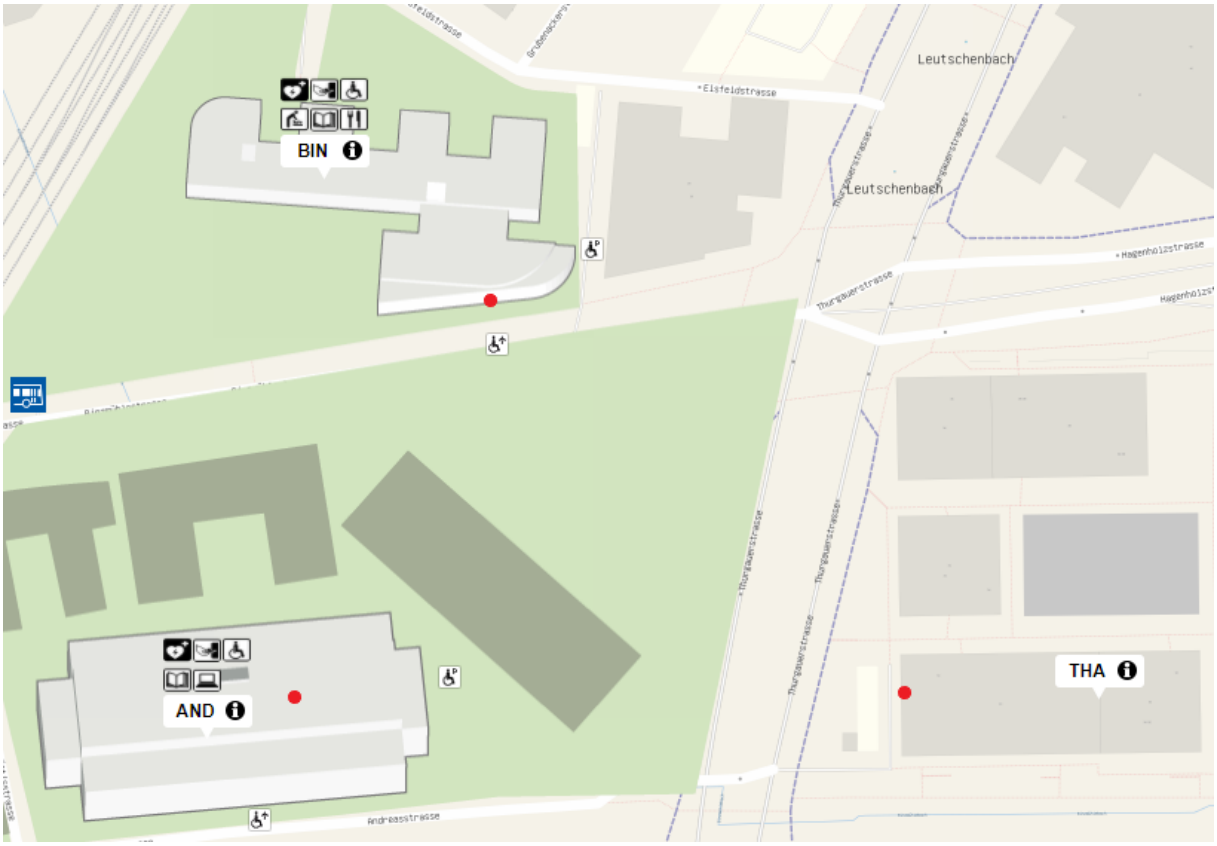

Figure 4.7: Exact locations of the nodes in BIN, AND and THA as red dots.

Figure 4.8: Outdoor node installed on the roof of Y13.

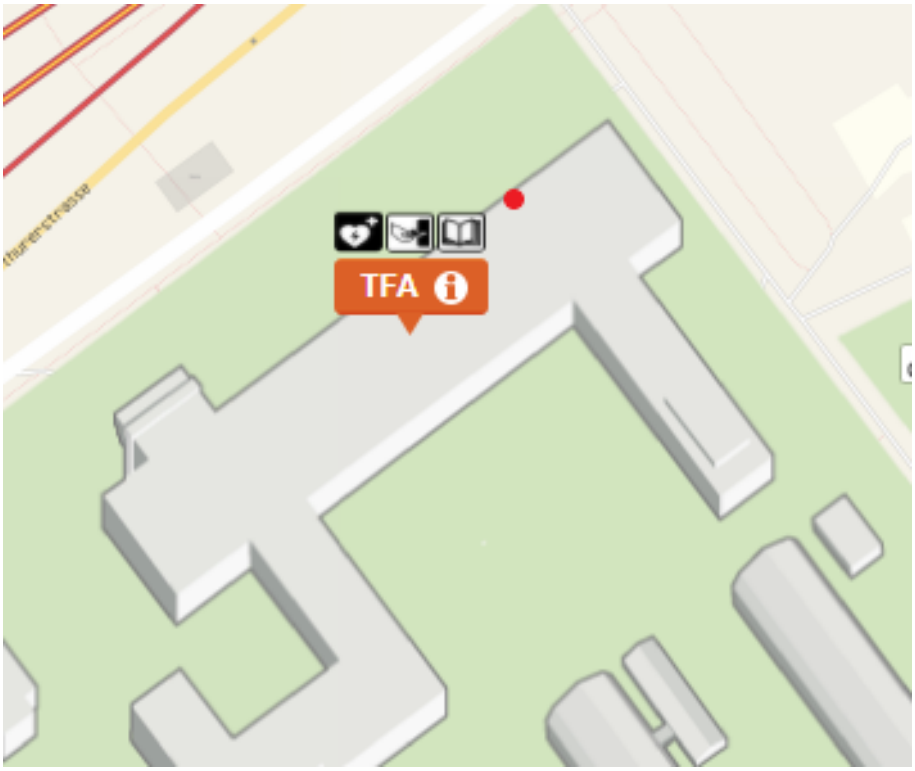

Figure 4.9: Exact location of the node in TFA as red dot.

Figure 4.10: Exact location of the node on the roof of Y13 as red dot.

Chapter 5

Evaluation

For the evaluation of the concept and its implementation a test network was deployed as described in the Section 4.4. In order to assess the test-bed, three scenarios were defined and these are evaluated in this Chapter. The following Section 5.1 shows the meshes that resulted from this test-bed across the different air data rates (ADR) with four or five nodes respectively to ease the understanding of the evaluation regarding the scenarios in the latter Sections.

5.1 Mesh structure

Due to the fact, that the Node in TFA is only working outside of office hours and is therefore disrupting testing with five nodes, this Section is split into two categories. First, mesh structures with four nodes are explained followed by the explanation of mesh structures with five nodes. It is shown how the meshes are structured across the different air data rates for both setups.

5.1.1 Mesh with 4 Nodes

This subsection discusses how the resulting mesh looks like with four deployed nodes across varying ADRs. Figure 5.1 visualizes the mesh on the mid channels, i.e., 2400 to 9600 bit/s. It is important to note that the only difference in topology between these ADRs and 19200 bit/s is that the Node in AND does not have an outgoing edge as shown in Figure 5.2. This means this node is not able to join as it's outgoing packets are not received by any other node.

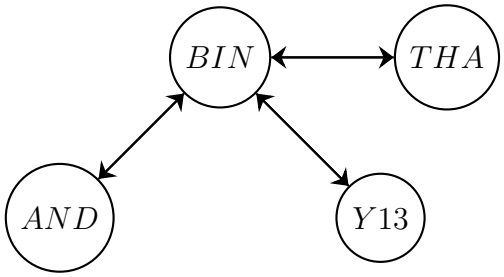

Figure 5.1: Typical graph for the LoRa mesh with 4 nodes at air data rate = 2400 bits/s up to 9600 bit/s.

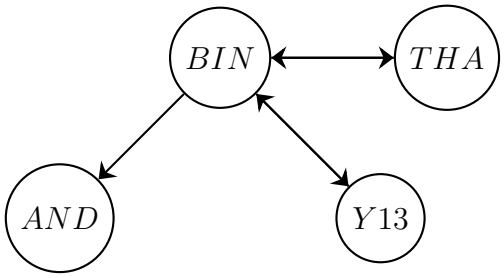

Figure 5.2: Typical graph for the LoRa mesh with 4 nodes at air data rate = 19200 bit/s.

Occasionally, there is also an edge between AND and THA in both directions as these nodes detect each other on the channels with ADR = 1200 or 2400 bit/s. It is not included, because the graph in Figure 5.1 suffices to grant multi-hop transmission for every node and the extra edges in general produced overhead when exchanging LSAs. Especially, edges that are not forming reliably produce overheads as the LSDBs are adjusted every time they appear or disappear. Moreover, on the higher channels the links BIN - AND and BIN - THA become very weak and unreliable, meaning AND and THA cause problems due to receiving packets at a reduced ratio.

5.1.2 Mesh with 5 Nodes

With the test-bed including all five nodes, BIN and Y13 seem to be the most central nodes and therefore candidates to be the controller. As shown in Figure 5.3, BIN has edges to all other nodes and Y13 to three other nodes with ADR = 1200 bit/s. Furthermore, all nodes have multiple links going in and out except THA, which has a very weak edge from AND. This means that the packets received by THA are usually from BIN. For the sake of completeness, this also occurs with the test-bed of four nodes. In addition to losing this link, increasing the air data rate to 2400 bit/s further leads to losing the links AND - Y13 and THA - Y13 as visualized in Figure 5.4.

Increasing the air data rate further leads to the graph in Figure 5.5. Even though this

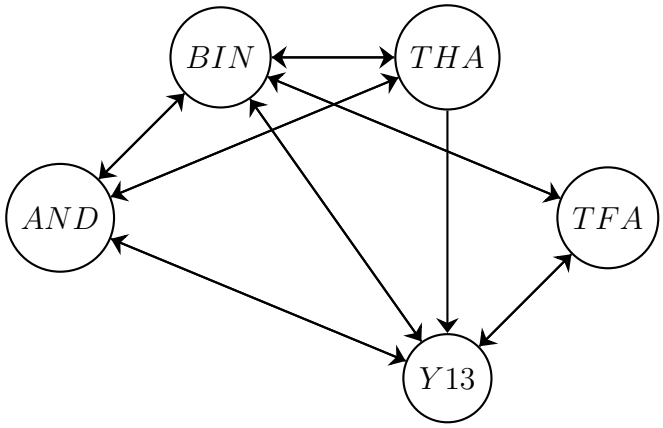

Figure 5.3: Typical graph for the LoRa mesh with 5 nodes at air data rate = 1200 bit/s.

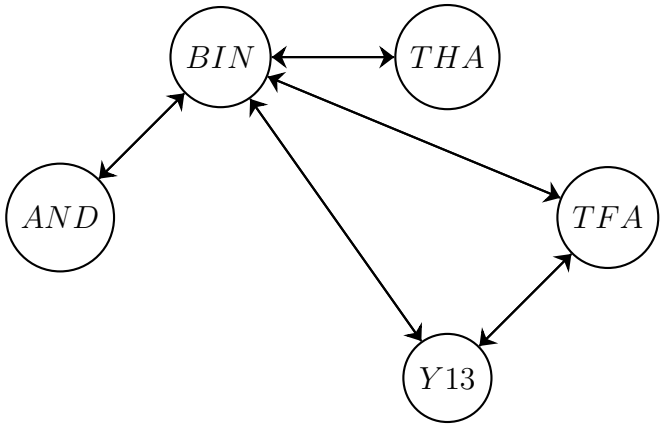

Figure 5.4: Typical graph for the LoRa mesh with 5 nodes at air data rate = 2400 bit/s.

mesh is optimal as it has the lowest possible number of edges, it produces problems. As mentioned before, the links between BIN - AND and BIN - THA are weaker on higher ADRs, making it difficult for AND and TFA to receive all LSAs. Therefore, increasing the air data rate leads to more overhead although there are less edges.

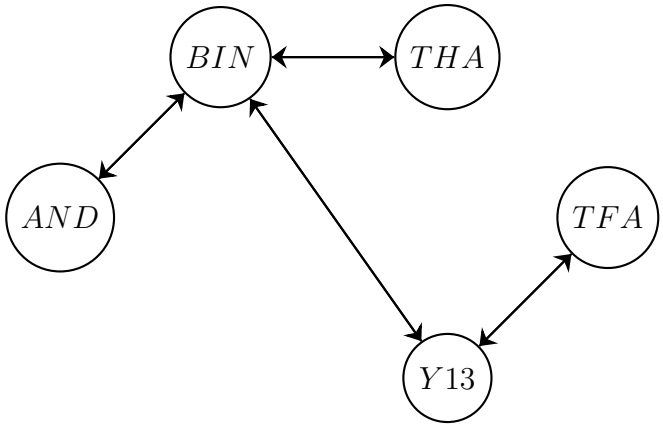

Figure 5.5: Typical graph for the LoRa mesh with 5 nodes at air data rate = 4800 bit/s.

In general the axis BIN - Y13 - TFA was very reliable and sustained up to the second highest air data rate. BIN - Y13 even works on 19200 bit/s sometimes, while Y13 - TFA usually caps at 9800 bit/s. This might be due to the fact that these nodes have direct line of sight as mentioned in the Section 4.4 and Y13 being outdoors. All other nodes being indoor, either directly behind the window or in case of AND, in the inner courtyard, might be the reason these individual links are unreliable.

5.2 Scenario 1: Initializing a mesh

This section analyses how the LoRa mesh is initialized. This scenario is assessed in three different aspects. Therefore, multiple KPIs are measured. First, the different channels are monitored to follow how the nodes are scattered across the channels. In particular, the rendez-vous channel and the channel where the mesh is initialized are focused on, to measure how many nodes are participating. Second, the time until all the LSDBs are synchronized is assessed and finally, all the packets being sent in that time are recorded in order to categorize them.

5.2.1 Channel monitoring

Figure 5.6 displays the number of nodes on the rendez-vous and the mesh channel, which is 1200 bit/s in this case, over time. Due to the nodes being scattered across the channels at the beginning of the scenario, only one node is on the mesh channel. The other four nodes are on the other channels (not rendez-vous). Such a situation can occur when a node for example missed an announcement that the channel is changed. Therefore, there must be a mechanism to unite the nodes on the same channel. The implemented mechanism puts nodes back to the rendez-vous channel, when they have not received any packets in the time they should have received five hello packets from one of their neighbours (i.e., five minutes). Due to this mechanism all the nodes show up on the rendez-vous channel after five minutes. On there, the controller picks a channel and announces it to everyone listening on the rendez-vous channel. In case of more nodes joining the rendez-vous channel, the controller stays a bit longer than the other nodes and continues advertising the mesh channel after the other nodes have already changed the channel. This is observable at minute 9 in Figure 5.6. Three nodes receive the announcement and leave to initialize the mesh on the other channel and two devices remain on the rendez-vous channel. As just mentioned, one is the controller and the other one does not receive the announcements, due to it being out of range of the controller. Finally, the controller

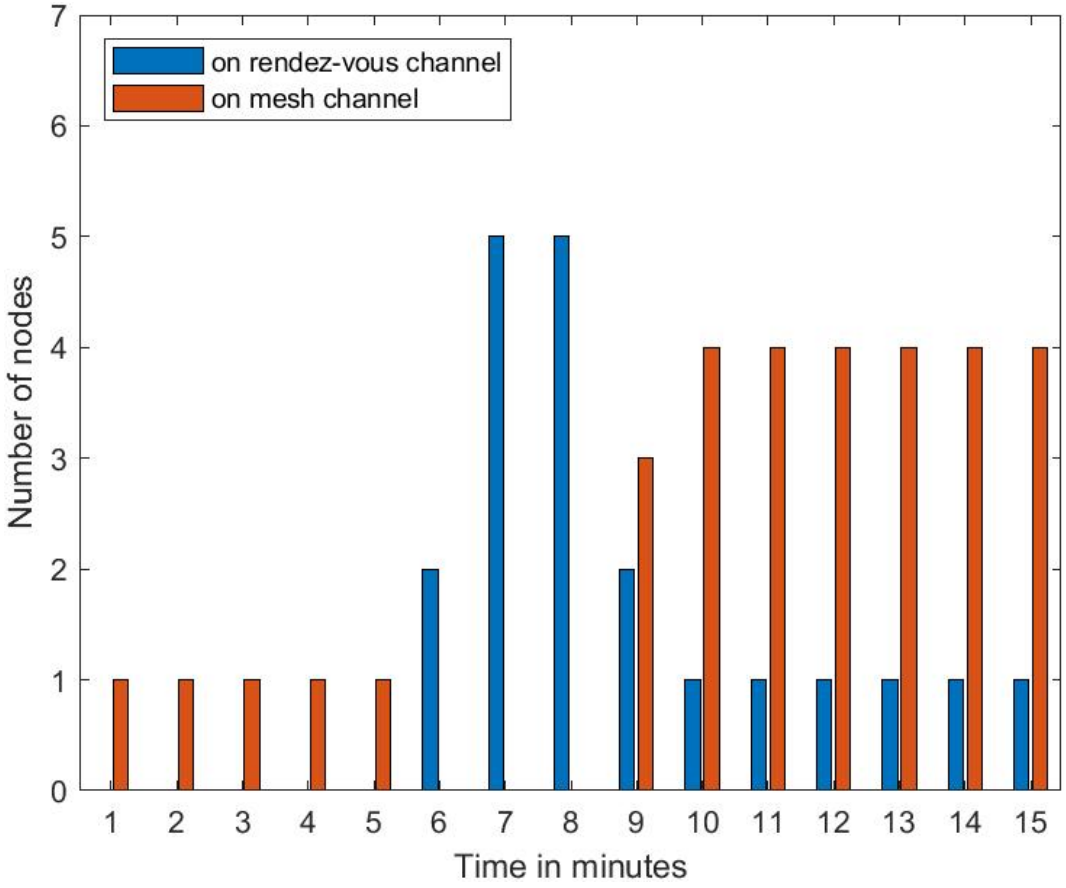

Figure 5.6: Node allocation on rendez-vous and mesh channel at 2400 bit/s.

also joins the mesh and one node is left behind. This means that only four of five nodes are participating in the mesh in this data set.

Figure 5.7 additionally shows the number of nodes in the controllers LSDB, which reaches the maximum at 14 minutes. It is important to note that this process is highly dependant of the amount of immediate neighbours of the controller. With more immediate neighbours, building the initial LSDB is considerably slower as more nodes are involved. However, once the first entries are synchronized, they are flooded faster. On the other hand, if the controller has for example only one neighbour, they would only form a mesh of two nodes and the controller has to synchronize the whole LSDB through this neighbour. The previous Section 5.1 showed the usual meshes that formed in this test-bed and Figure 5.3 shows no edge from Y13 to THA, which also occurs on the rendez-vous channel. As Y13 is the controller in this data set, this missing edge is the reason that one node, i.e., THA, is left behind on the rendez-vous channel. There are two ways to include THA in the mesh: (i) let THA join later through advertising the mesh on the rendez-vous channel (scenario 2) or (ii) use a different controller, which reaches every other node on the rendez-vous channel. Due to the fact that an optimal controller could be determined by

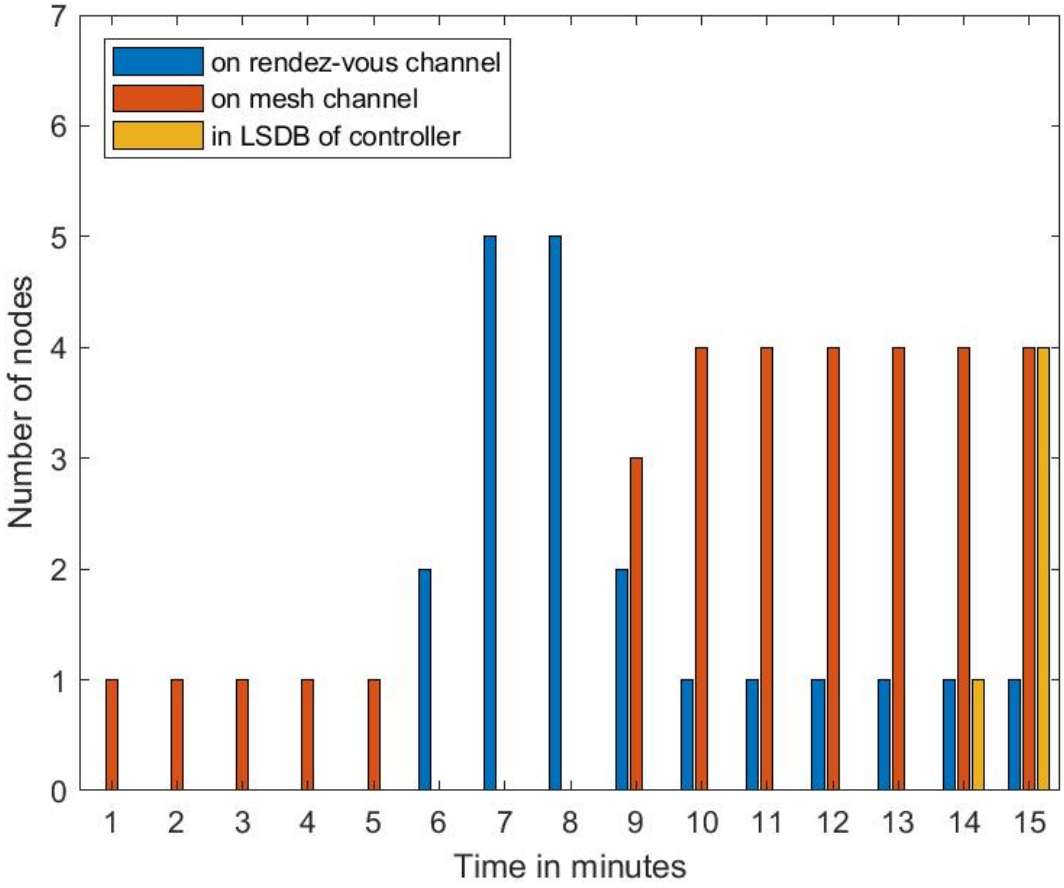

Figure 5.7: Nodes on rendez-vous channel, mesh channel and in the LSDB of the controller (Y13).

detecting neighbours on the rendez-vous channel, the role of the controller was assigned to a node with the most neighbours, i.e., BIN. This leads to a better outcome in scenario 1 as shown in Figure 5.9, as all five nodes join the mesh due to all nodes receiving the advertisement on the rendez-vous channel.

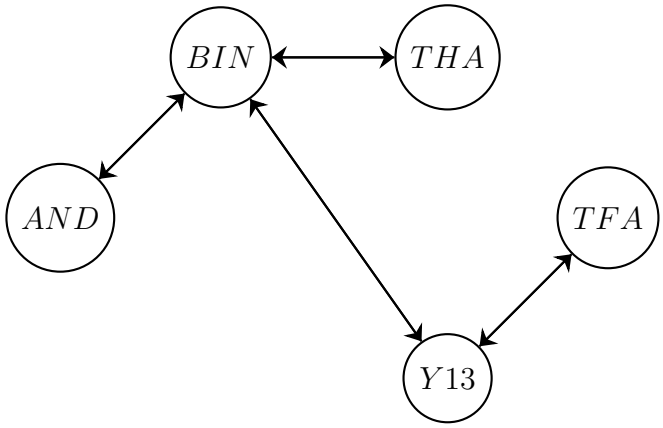

Figure 5.8: Typical graph for the LoRa mesh with 5 nodes at air data rate = 4800 bit/s.

Figure 5.8 is the visualization of the typical graph that formed in the test-bed and it shows

the node in BIN with the most neighbours. Analogously, it also shows that there is no edge from BIN to TFA. This mostly occurs on the mesh channels, but also rarely on the rendez-vous channel. Such a missing link between any node and the controller on the rendez-vous channel is the reason that the TFA node sometimes remains on the rendez-vous channel. However, in the majority of cases the scenario results in this graph. In addition, Figure 5.9 displays the channel monitoring when TFA receives the advertisement from BIN on the rendez-vous channel. In such a case, all five nodes find themselves on the same channel and initialize the mesh.

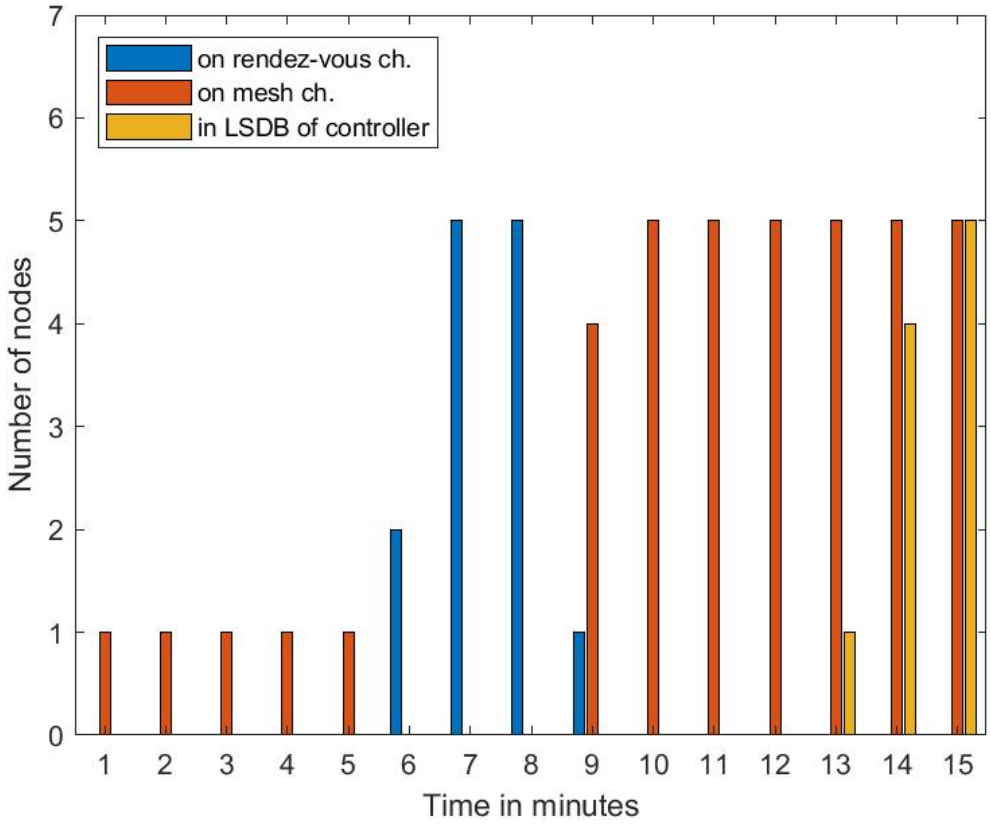

Figure 5.9: Nodes on rendez-vous channel (ch.), mesh channel and in the LSDB of the controller (BIN).

5.2.2 Amount of packets sent

An important aspect of initializing the mesh is the total amount of packets sent. Figure 5.10 shows a typical distribution of the packets, where hello packets stay constant as they are periodically defined. In this case, hello messages are sent once per minute, thus there is one hello packet per node and per minute. In the first three minutes in the mesh channel, the nodes count the hello packets they receive and only after they received enough to grant a good enough connection (i.e., received at a minimum three out of five), they accept the

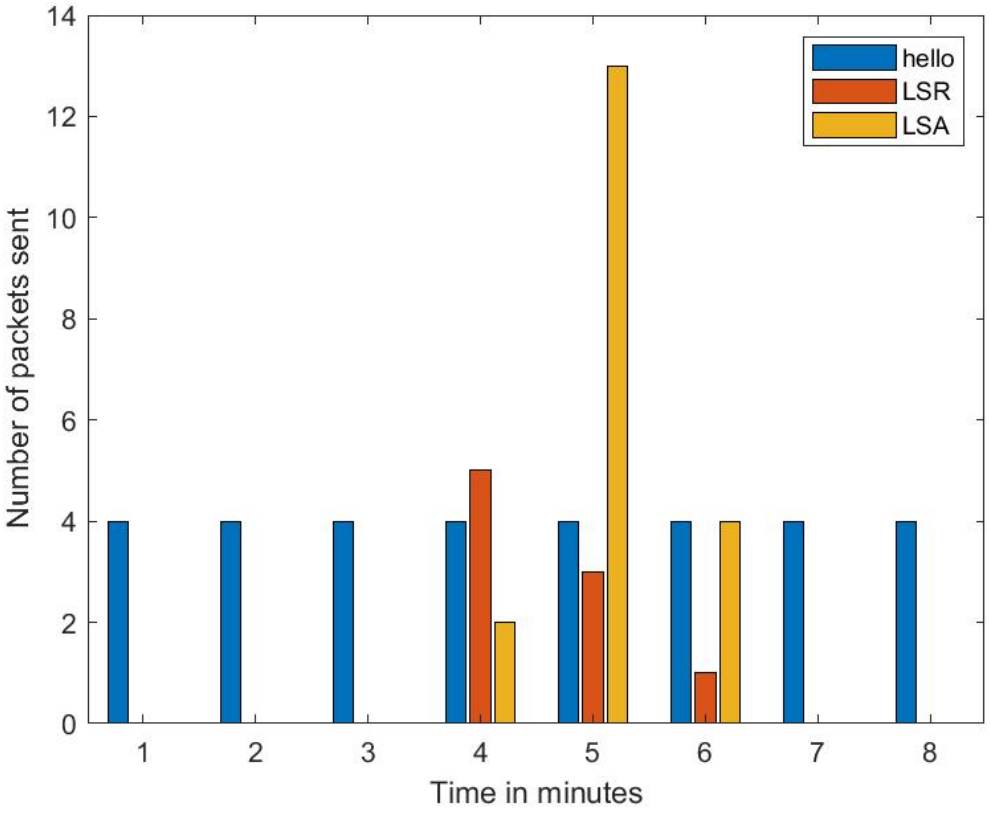

Figure 5.10: Amount of hello, LSR and LSA packets over time while initializing mesh with 4 nodes.

sender as a neighbour. Therefore, the routing protocol only starts working at minute 4. Figure 5.10 shows that there is an intensive period for a few minutes when the routing protocol is operating in requesting and sharing LSDB entries. It regularly occurs that there are more LSRs than LSAs at the start, because some nodes still have an empty LSDB and therefore cannot send any LSAs. On the other hand, the requesting node already has entries in the LSDB and therefore compares the hashes in the hello messages. Because of that, it requests LSAs even though the other nodes does not possess any. After that however, most nodes have recognised neighbours, i.e., entries in their LSDBs, and this is when the LSAs dominate the amount of packets sent in and around the fifth minute, until the majority of LSDBs are synchronized. Finally, the count of LSRs and LSAs decreases until they completely vanish as all hello packets contain the same LSDB hash. At this point the hello messages ensure any two neighbours to stay in contact with each other to grant a safe multi-hop transmission.

In case of a node not receiving any hello packets from an accepted neighbour, it will get dropped from its list of neighbours. However, it is not implemented, to drop it completely from the mesh. One approach to implement such a mechanism is to notify the controller that one node might have dropped such that the controller could take an appropriate

action. Otherwise the dropped node will only be deleted in the list of its neighbours but its entry in the LSDB will remain. The controller could decide to delete it from the LSDB. In such case, the dropped node would have to join the mesh again, which would make sense because it is at this moment in time not guaranteeing a safe multi-hop transmission.

5.2.3 Time to converge LSDBs

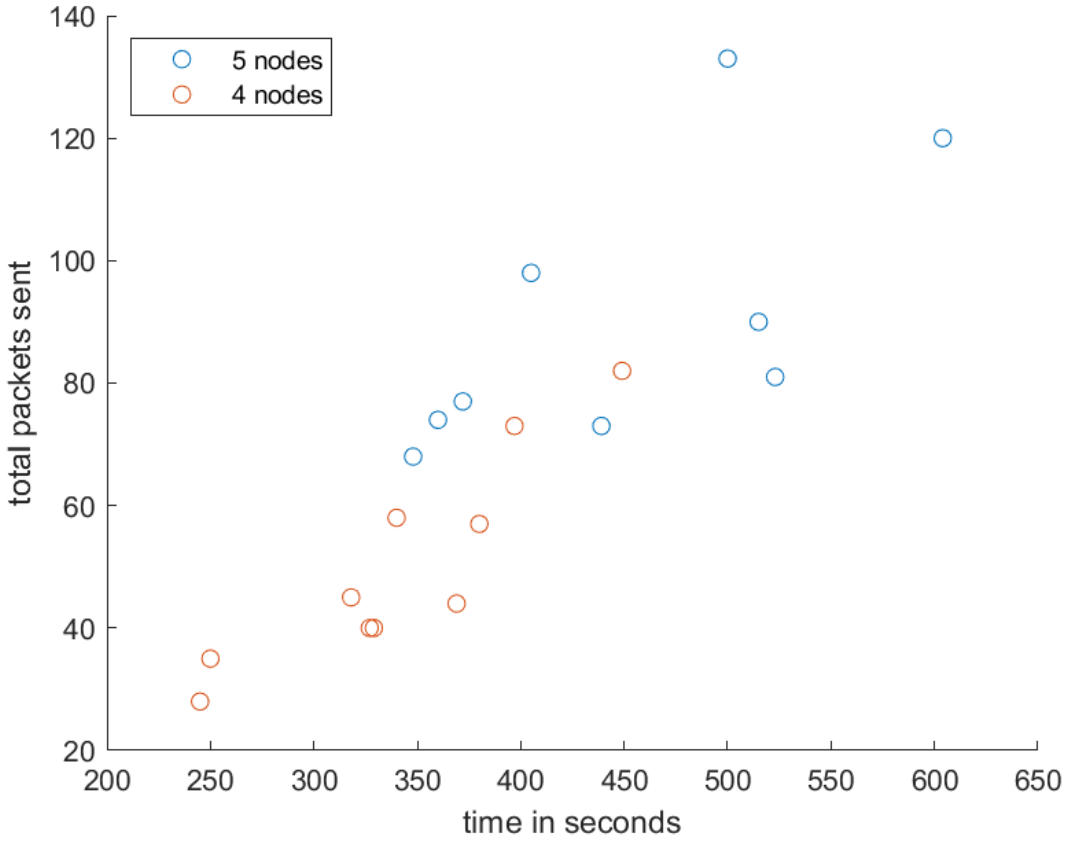

Figure 5.11: Total number of packets sent until all LSDBs are synchronized.

The time it takes until all LSDBs have converged depends on various factors, predominantly number of nodes. Therefore, Figure 5.11 shows the total number of packets sent until all nodes have the same LSDB with five nodes and four nodes in the mesh. The air data rate in this data set is on the lower channels, such that the LSDBs converged. On the higher air data rates, it either occurred similarly, i.e., similar time to converge, or the LSDBs did not converge at all. Since this data set is on the lower air data rates available for the E32-868T20D, it means that this configuration leads to a mesh with a higher number of edges in the graph as discussed in Section 5.1. Therefore, it takes longer to synchronize the LSDBs. The mesh with five nodes typically converged in less than 11 minutes while the mesh with four nodes converged in less than 8 minutes. As just

mentioned, the air data rate also influences the time to converge the LSDBs, such that the LSDBs either converged in 11 or 8 minutes respectively or not at all on the higher air data rates. With higher air data rate, there are less edges due to the lower spreading factor, i.e., lower range of the modules. This could have lead to a faster convergence if the links were still strong. However, the gathered data does not show a faster convergence. In practice, it actually takes longer to converge at higher air data rates due to the links not being strong enough, meaning not all packets are received. In addition, the LSDBs often do not converge at the higher air data rates, as some nodes either only have an edge going in and none going out, or the other way around. In order to be able to synchronize its LSDB, a node needs to have both. Without an outgoing edge, it cannot share its own entry and without an in-going edge it cannot receive all the other entries. This leads to a situation visualized in Figure 5.12, as this situation regularly occurs with node AND.

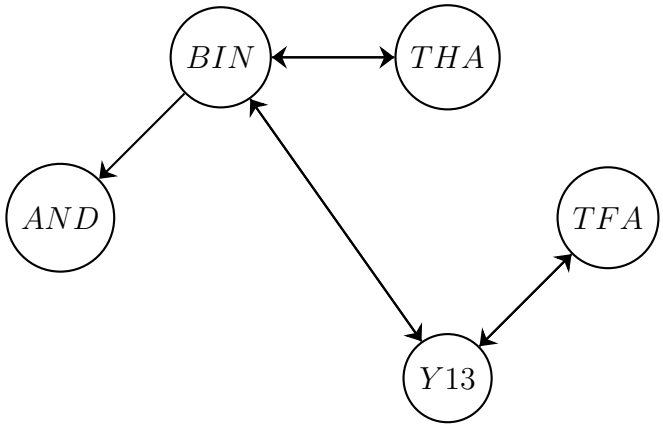

Figure 5.12: Graph with node (AND) not being able to join at air data rate = 9600 bits/s.

In this case, node AND cannot share its entry with node BIN and therefore, it cannot join the mesh. This might be due to the indoor location of this node, facing the inner courtyard of the building AND, lowering the range significantly with the higher data rate, i.e., lower spreading factor.

In addition, a mesh without bidirectional edges can take a lot longer to converge. Imagine an edge from AND to Y13 in Figure 5.12. The reason being, one node constantly requesting LSAs from a node that does not receive the request. In this case, the requesting node (AND) can only receive the LSAs of that node (Y13) through a third node (BIN), which might need to be synchronized first. This heavily influences the time of convergence.

5.2.4 No Convergence

As mentioned in the previous Subsections, the links in the test-bed fluctuate a lot. Sometimes, a mesh did not form even on the lower air data rates. Due to links not being strong enough, the LSDBs do not converge. Usually, this is not a problem, because the nodes unable to join go back to the rendez-vous channel. In some rarer cases however, a node receives enough packets to not leave, but at the same time not enough to be in the mesh constantly. This leads to a situation, where the LSDB constantly changes back and forth and the multi-hop transmissions are not guaranteed.

For scenario 1 it can be concluded that in most cases, all five nodes participate in the mesh and are accepted as a neighbour in it, therefore showing that the developed approach works. The performance depends on a controller that can reach as many nodes as possible on the rendez-vous channel. With more reliable nodes, in this case BIN - TFA, the scenario could have run even better.

5.3 Scenario 2: Joining node

In this scenario, there is an existing mesh and a new node joins it. Due to the varying reliability of individual links at the time of testing, the evaluation of this scenario was only performed on the lower air data rates, i.e., 1200 and 2400 bit/s. First, the evaluation of this scenario monitors the rendez-vous channel and the mesh channel. Second, the amount of the different types of routing protocol packets is measured and discussed. And finally, this Section assesses the performance in measuring the time of convergence regarding all LSDBs and the total amount of packets sent until convergence is achieved.

5.3.1 Channel monitoring

Figure 5.13 shows the distribution of nodes across the different channels. At the beginning, four nodes are in the mesh and one node is on the rendez-vous channel. This situation occurs for example, when a new node is setup or one dropped out. When a node dropped out, it does not detect neighbours and goes onto the rendez-vous channel. However, a new node can easily be configured to boot up and go onto any channel. There it will either detect the mesh or go to the rendez-vous channel in behaving similarly to a dropped node.

Once a node is part of the mesh, it will periodically go onto the rendez-vous channel to advertise the channel of the mesh. This is seen in minute four in Figure 5.13, where two

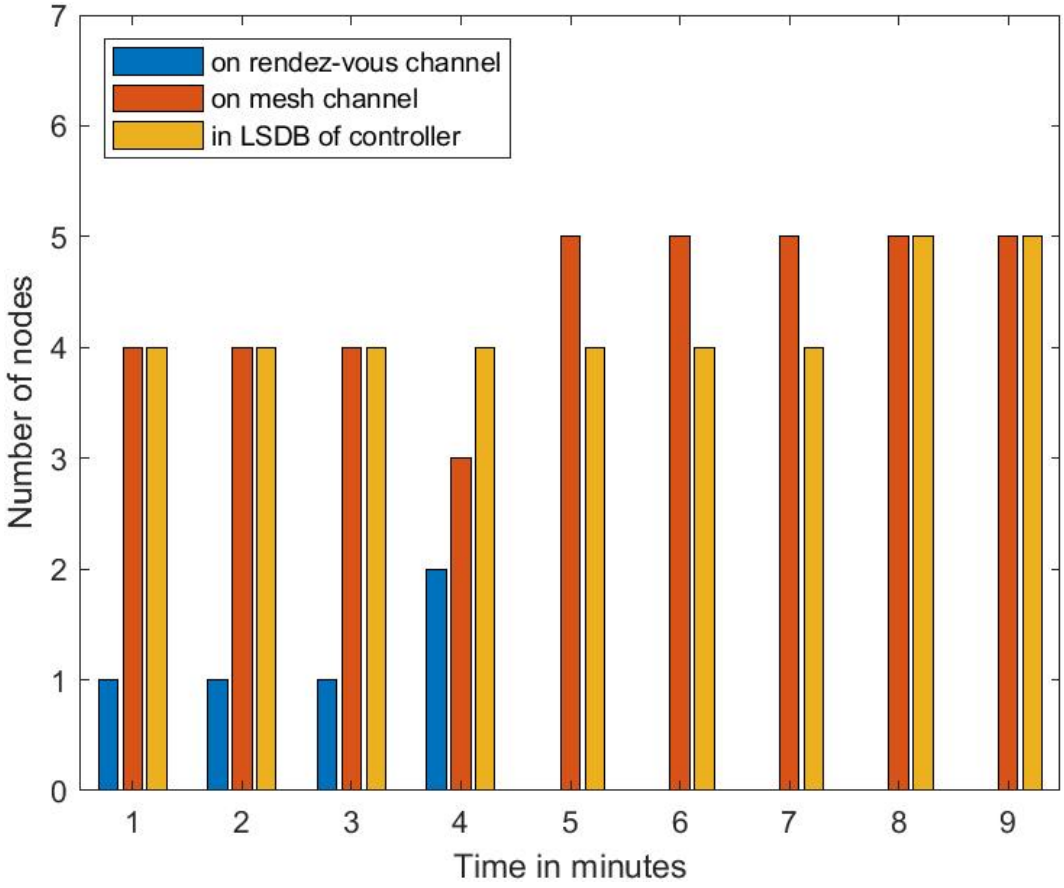

Figure 5.13: Nodes on rendez-vous channel, mesh channel and in the LSDB of the controller (BIN).

nodes are on the rendez-vous channel. Due to the fact that the rendez-vous channel is at 300 bit/s, which is corresponding to Spreading Factor (SF) = 12, the range of packets is maximized on the rendez-vous channel. This means that the chance of any joining node receiving the advertisements is as high as possible. In this test-bed however, joining nodes not receiving advertising packets occurs for some pairs, i.e., THA - TFA and AND, TFA.

Upon receiving an advertisement packet, the new node goes onto the mesh channel and starts the routing protocol, meaning the hello packets. After a few iterations of exchanging LSAs, the LSDBs usually converge. In this data set the controller received the LSDB entry of the new node very fast, only four minutes after the new node came onto the channel as shown Figure 5.13 at minute 8. However, this does not mean that all LSDBs are synchronized, but it shows that the new node has been accepted as an immediate neighbour by some node and that its LSDB entry has reached the controller.

This advertising behaviour occurs periodically and every node of the mesh does it to ensure that a waiting node is picked up. It is important to note, that advertising happens very rarely. This is to grant multi-hop transmissions as much as possible. As for the time

a node is advertising, it is not granting the forwarding mechanic. Generally, it should suffice to spread out this behaviour over time, largely depending on the application.

5.3.2 Amount of packets sent

Similar to the first scenario, the total amount of packets sent is an important aspect. As Figure 5.14 shows, only one node joining has a big impact on LSAs and LSRs sent. First of all, the hello packets increase by one per minute as there is one more node on the mesh channel. After the new node accepts immediate neighbours and the mesh nodes accept the new node as immediate neighbour, there is a huge spike in LSAs. The LSAs are much higher than the LSRs, because the LSDBs already have four entries when the new node joins. In the data set of Figure 5.14 the joining node is TFA and it only has a bidirectional edge to Y13. This means that all updated or new LSDB entries have to pass through Y13 and it is an average data set in convergence time and number of packets sent. In case of

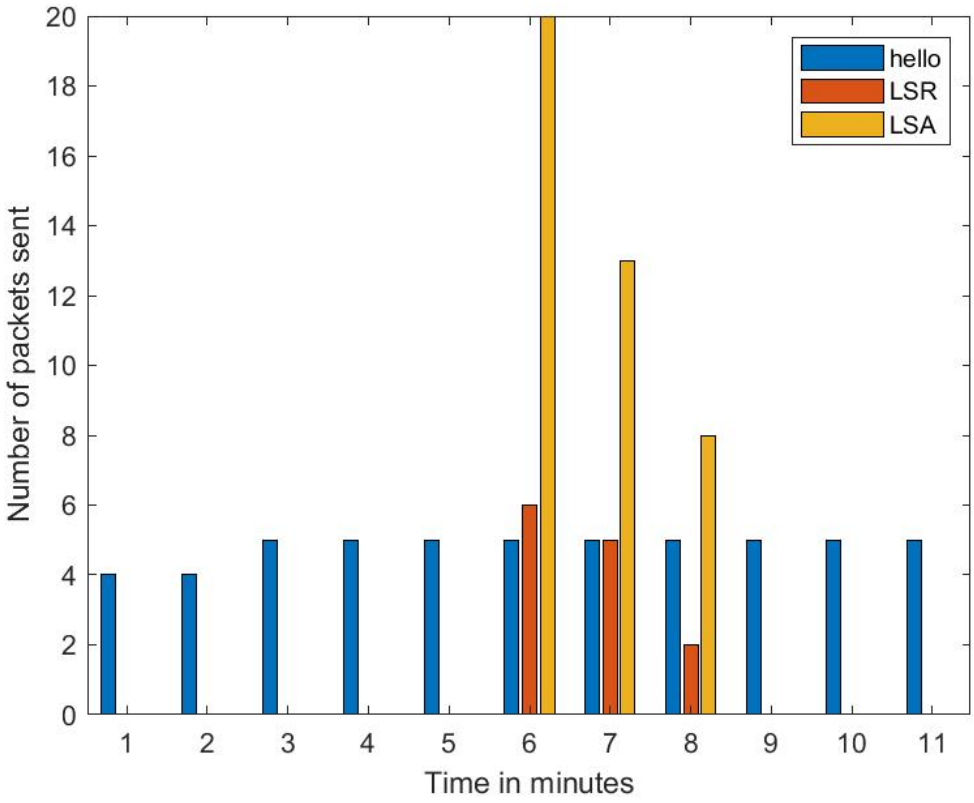

Figure 5.14: Amount of hello, LSR and LSA packets over time while one node joins the mesh of four nodes.

the joining node detecting multiple neighbours, there are a lot more packets. Due to the fact that the LSDB entry grows on size as the new node detect multiple neighbours, it is shared a lot more. The same phenomenon occurs to the the existing LSDB entries. In

such a case, there is not only one entry to be updated, but several. This increases the total number by a huge amount. In addition, the time to converge the LSDBs obviously has an influence on the total amount of packets sent, which is discussed in the next Subsection.

In addition, the link quality has a big influence on both convergence time and packets sent. If the mesh is containing any weak links, this process is a lot less optimal than shown in Figure 5.14. It actually does not matter, where the weak link is, i.e., if it is linked to the new node or if it was already in the existing mesh with four nodes. The weak link will increase the convergence time because the node will either miss LSRs or LSAs, leading to a loop of requesting links.

5.3.3 Time to converge LSDBs

The time it takes until all LSDBs are equal depends on multiple factors. This scenario was not tested with four nodes, therefore not revealing any information about this configuration. Figure 5.15 shows the convergence time and the amount of packets sent in that time. The gathered data suggests that the LSDBs converge in four to eight minutes and the amount of packets sent varies between 53 and 115, which is a lot considering the amount of packets sent is very similar amount of packets sent during initializing a mesh in scenario 1. However in scenario 2, there is already a mesh existing with four nodes, which needed between 30 and 80 packets alone. It seems that picking up a new node is very expensive compared to newly initializing the mesh.

The aim for this scenario is to show that a mesh with minimal edges can add a new node. However, it is not possible to show this, because the meshes are not constant enough. Due to AND and THA having weak incoming links, they often miss either the hello packet which initiates the LSR or the incoming LSAs which increases the time to converge the LSDBs. It is important to note that the LSDBs converged across various air data rates, but the convergence time depends a lot on weak links. With weak links already in the mesh or between the new node and the mesh, the LSDBs does not converge, even though a multitude of packets is exchanged. The reason for this being LSRs or LSAs not being received.

The graph in Figure 5.16 shows the mesh that would have resulted in a better performance. If this graph had strong links, scenario 2 could have worked better for a number of reasons. Such a situation would only add one new LSDB entry for THA and update the entry for Y13, which is then flooded in the mesh. In the field test however, this only occurs rarely. Usually the new node has several neighbours, meaning that all their entries are updated and the entry for THA is also updated multiple times due to THA having more than

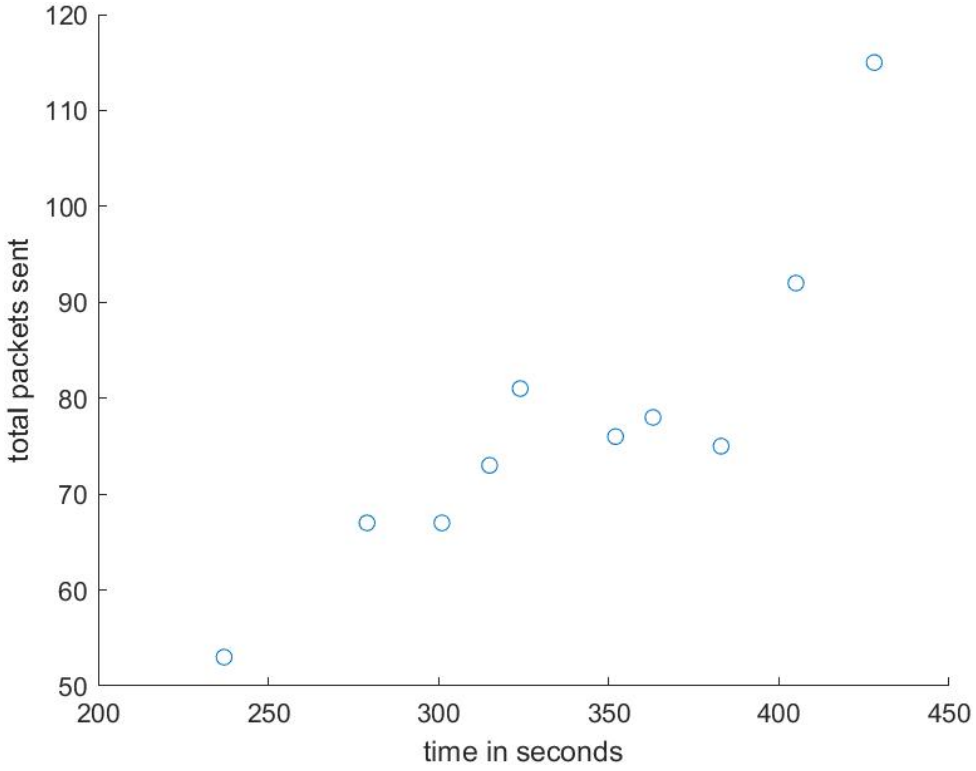

Figure 5.15: Total packets sent over time to converge LSDBs.

one neighbour. This had a big impact on the whole scenario as the number of packets increases a lot with increasing edges in the graph.

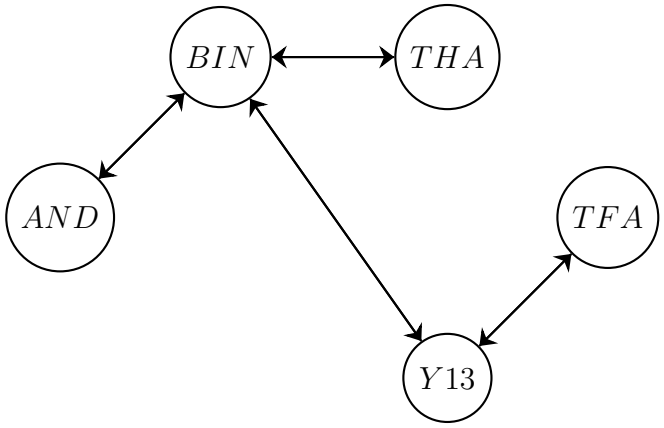

Figure 5.16: Graph with nodes AND and THA having weak links and TFA joining.

The evaluation of scenario 2 shows that the implemented solution is able to accept a new node in a plug and play fashion. Therefore, the goal to accumulate more nodes via the rendez-vous channel was accomplished. This mechanism is a basis for the next scenario, as it is applied in it. On the other hand, it can be argued that the mechanism is not reliable enough as the data sets showed varying results. Often, weak links in the mesh were the reason for not converging LSDBs. Improving the reliability of these weak links

might grant a better overall performance of scenario 2.

5.4 Scenario 3: Reliable Mesh

The third scenario combines the previous two scenarios and extends it with more controller functionality. The controller decides what air data rate is used and tries to find an optimal mesh. The optimal mesh is defined as a mesh containing all nodes with the highest possible air data rate. In order to evaluate this scenario, the channels are monitored as well as the number of entries in the controllers LSDB. To conclude the evaluation of scenario 3, the scenario was run with three nodes that proved to be nodes with the most reliable links.

In this scenario the controller has a more central role compared to the previous scenarios. As mentioned before, the role of the controller is assigned to one of the deployed nodes rather than letting the nodes figure it out on their own. As BIN has the most central role in the mesh, it was assigned the controller role to reduce the overhead.

Scenario 3 starts on a given channel with a mesh including all nodes. Figure 5.17 shows the channels on which nodes were active during a test with five nodes. This one started at the 2400 bit/s which is the default channel of the module. The mesh is already formed at zero minutes. As the LSDBs have all entries as shown in Figure 5.18, which is based on the same data set, and the LSDBs have converged the controller analyses the LSDB. It checks if all nodes have an entry and is detected by another node as discussed in Algorithm 1 in Section 3.4. In other words, the controller checks if every node has an incoming and an outgoing edge. As long as these requirements are met, the controller increases the air data rate. This occurs in both Figures at 7 minutes. Further, the controller tells all other nodes in a multi-hop manner to increase the air data rate. The controller needs to pay attention that the nodes further in the mesh receive the packet first, such that every node receives their packet, which is only accepted if it is coming from the controller. As discussed in Section 3.4, nodes that receive the new ADR stay on the old channel for 60 seconds, to still grant multi-hop communication, before they change the channel.

At this point all nodes delete their list of neighbours and LSDB in order to initialize the mesh again on the next channel. Because of that, Figure 5.18 shows an empty LSDB between 8 and 12 minutes and Figure 5.17 shows all nodes on the next channel, i.e., 4800 bit/s. Then in minute 13, the controllers LSDB starts having entries again. However, this time the LSDB is only growing to three entries rather than five. This is because two nodes have not received enough packets to stay on the channel and they went onto the

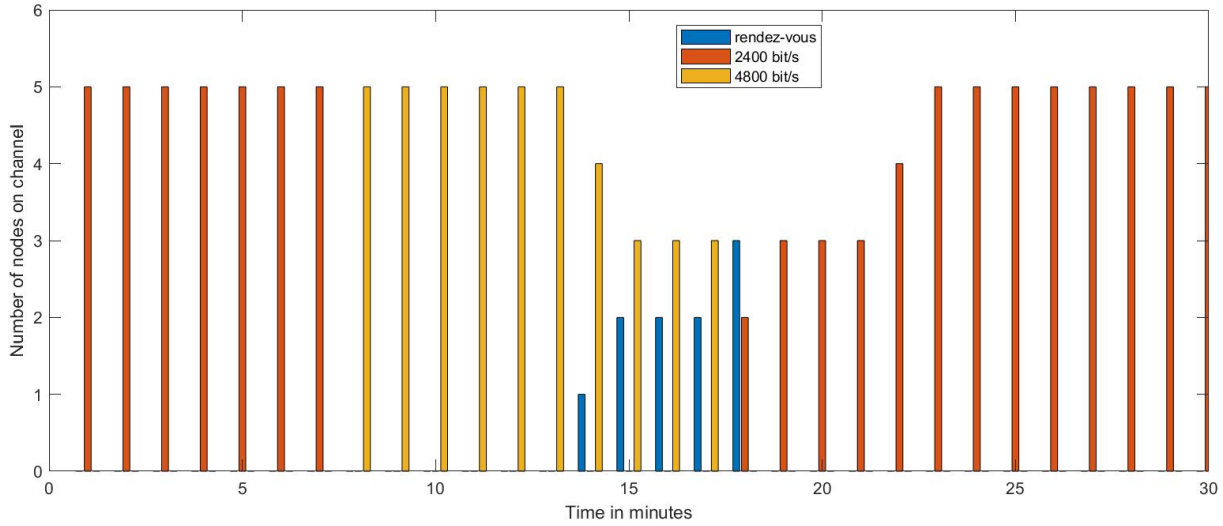

Figure 5.17: Channel monitoring during a test of scenario 3.

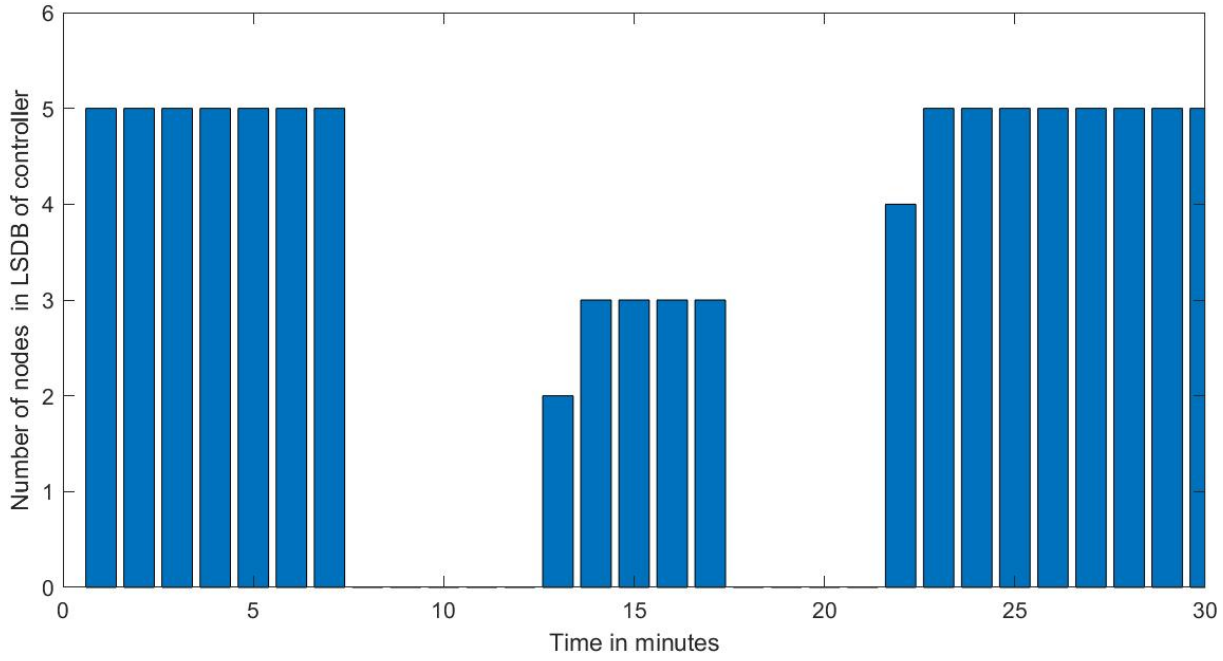

Figure 5.18: Number of nodes in controllers LSDB during a test of scenario 3.

rendez-vous channel which is visualized in Figure 5.17 at minutes 14 to 17. This means that the air data rate needs to decrease again.

Upon analysing its LSDB on ADR = 4800 bit/s, the controller decides to decrease the air data rate back to the value that it had previously, because the LSDB does not have five entries. First, the controller informs the other nodes in a multi-hop manner to decrease the air data rate. Then, the controller goes onto the rendez-vous channel to pick up the two nodes, as the controller knows that two nodes are missing. This is shown again in Figure 5.17 as there are three nodes on the rendez-vous channel at minute 18. Finally, all

5 nodes find themselves back on the same channel, where they formed a mesh previously with all nodes involved. At the end, the Figures show that all nodes are on the same channel and in the controllers LSDB respectively.

Running this scenario multiple times resulted in a data set describing which channels are more optimal in the test-bed. With four and five nodes, this scenario usually ends up on the lower channels as shown in Figure 5.19. Unfortunately, it does not converge every time, meaning the controller still wants to decrease the air data rate when it is already 1200 bit/s. This is again due to the weak links of AND and THA. Therefore, it was decided to run this scenario with only three nodes that grant the most reliable links, i.e., BIN, Y13 and TFA.

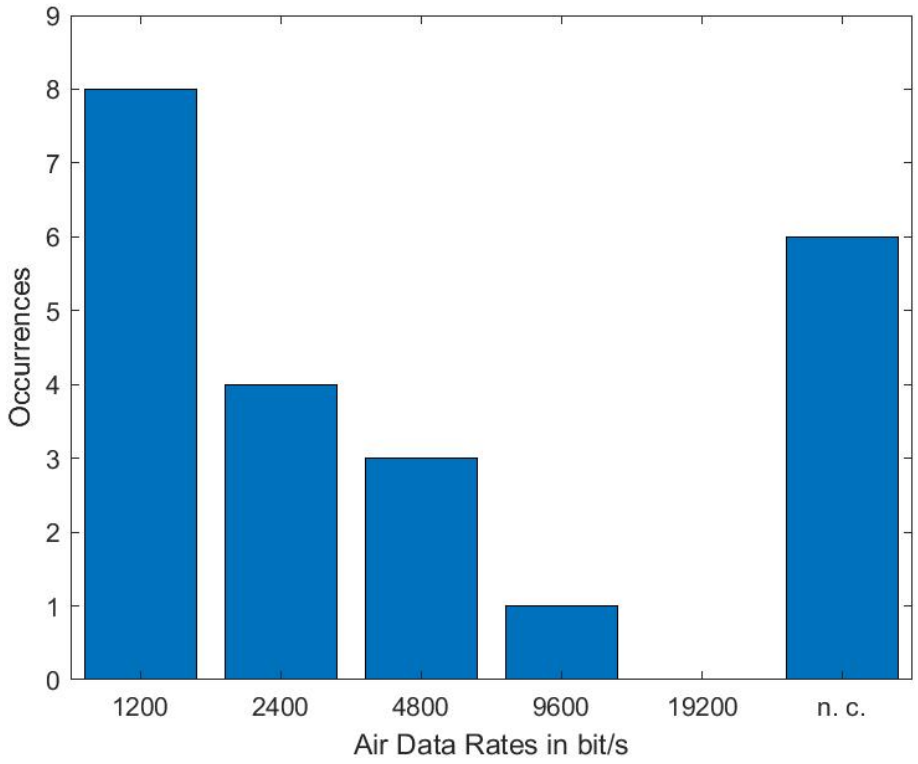

Figure 5.19: Resulting air data rates after running scenario 3 with four or five nodes, n.c. is the case when no convergence was achieved.

Interestingly, running the scenario with only these three nodes results much more often in a mesh on the higher air data rates as shown in Figure 5.20. Sometimes, there is even an edge from TFA to BIN on these higher channels. On 19200 bit/s the mesh usually lacks one edge from Y13 to BIN. This means BIN cannot join and it does not receive the LSDBs. Therefore, the mesh usually decreases the air data rate back to 9600 bit/s and stays there with these three nodes. It is important to note that scenario 3 always converges with these three nodes, showing that the implementation works reliably with nodes granting strong links.

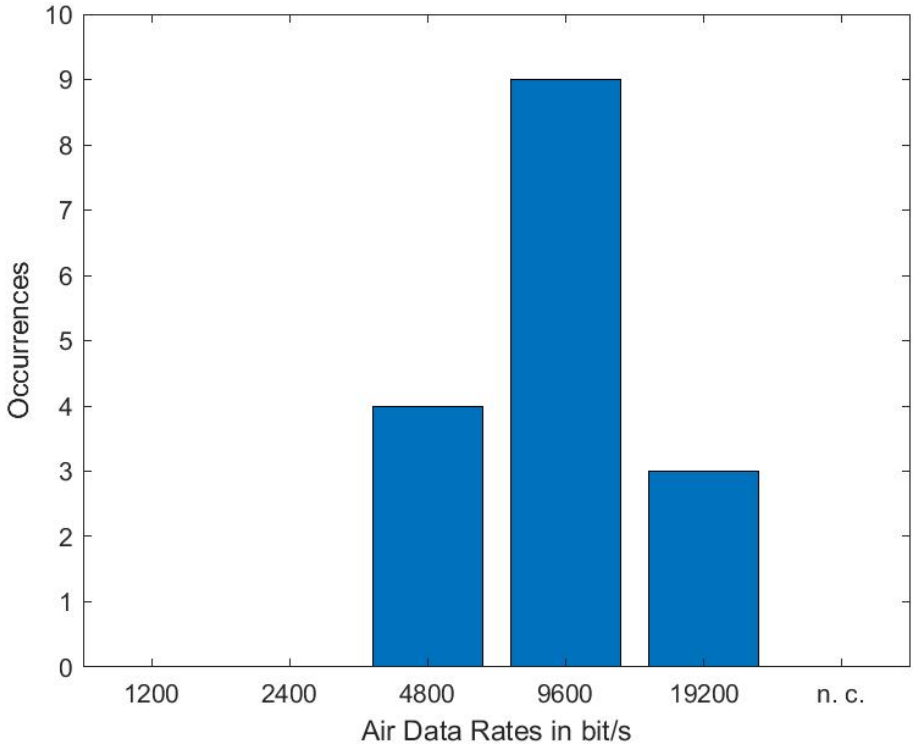

Figure 5.20: Resulting air data rates after running scenario 3 with three nodes, n. c. is the case when no convergence was achieved.

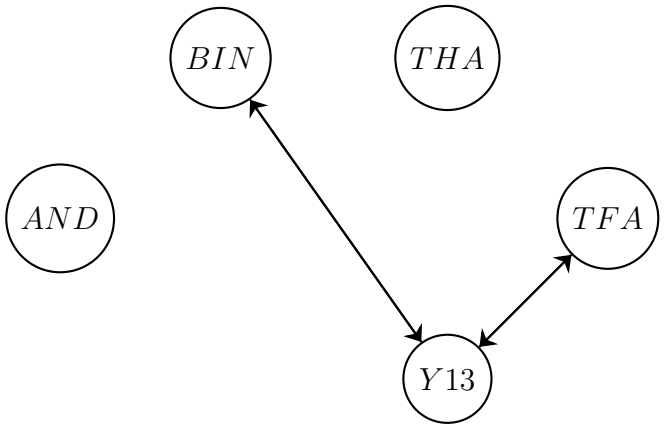

Figure 5.21: Resulting mesh on 9600 bit/s through running scenario 3 with three nodes.

It is not possible to reliably replicate this behaviour with the nodes AND and TFA. Actually, the mesh with all five nodes only rarely reached 4800 or 9600 bit/s. This has to be due to the locations of the nodes. The setup and configuration do not explain why the node in THA is so unreliable compared to BIN and TFA, as the setup is very similar right behind the window facing a direction to another node. On the other hand, the node in AND has a bad location, facing the inner courtyard on the second floor out of five.

The statistics matrix of the controller gives insight into which links are more reliable than

others. However, it is often not complete, because packets form nodes with weak links are sometimes unable to reach the controller. For example the following matrix from a data set on ADR = 4800 bit/s with five nodes where the rows correspond to the order of nodes: BIN, THA, AND, TFA, Y13. To create this matrix, the thresholds for leaving the channel and accepting a neighbour have been lifted. The matrix shows that the axis BIN - Y13 - TFA works well with very high percentages and it reveals the nodes with bad links, i.e., THA and AND. For example A[3,1] shows that AND only receives 6% of hello packets from BIN, where TFA and Y13 receive all of them. Because this matrix only accumulates when the thresholds are lifted, it is not possible to use it to determine nodes to exclude for the mesh in a reasonable time frame.

$$Adjacency\ Matrix\ with\ packet\ received\ ratios = \begin{bmatrix} 0 & 24 & 32 & 92 & 100 \\ 11 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 \\ 100 & 0 & 0 & 0 & 100 \\ 100 & 0 & 0 & 100 & 0 \end{bmatrix}$$

5.5 Evaluation of all scenarios

This Section combines the results from the three scenarios and the typical meshes that formed during evaluating the test-bed. In general, the results with all five nodes fluctuate a lot due the fact that the nodes AND and THA do not have reliable links. The scenarios 1 and 2 work on the lower channels, but even there, they often do not have converging LSDBs.

The results of scenario 1 show that a good selection of the controller has an impact. It shows less overhead in LSAs and LSRs when the controller is assigned to a node that is more central in a mesh. Therefore, a mechanism that elects the controller with the most neighbours on the rendez-vous channel would reduce the overhead. In scenario 2 however, the controller selection does not have an influence, as all nodes do the advertising.

Further, initializing the mesh and a new node joining the mesh both suffer from nodes that have weak links. The total number of packets increases rapidly when the mesh includes weak links, where the direction of them does not matter. It is possible that the weak links are the only reason that the time of convergence and the total packets sent fluctuate so much.

The evaluation of scenario 3 concludes that removing weak links has a positive effect on the performance. The data clearly suggests, that the reliable mesh suffers if there are

weak links in the mesh. Figure 5.19 shows exactly how the controller often wants to decrease the speed, even if the mesh is already on the lowest channel. Removing these nodes resulted in a much more interesting data set, where the LSDBs always converged. It is important to note, that most cases resulted in the second highest ADR, which is almost the optimal mesh in general. Further, it can be concluded that the mechanism is successfully implemented, as the LSDBs always converge and all nodes are in the mesh at the end. The successful implementation of scenario 3 also suggests that scenario 1 and 2 indeed work better with stronger links, as these two scenarios actually occur in scenario 3. The positive effect of removing the weak links suggests that improving the weak links might have a similar effect on the performance.

Especially the last scenario shows that the nodes AND and THA prevent meshes on higher air data rates as running the scenario without them makes it converge at higher air data rates. It is important to note that the higher channels are more effective in multiple ways, therefore, the aim is to reach a higher channel corresponding to lower SFs. Further, the lower SFs enable bigger payloads and higher data rates. While the data rates roughly correspond to the air data rates of the E32-868T20D, the SF12 and SF8 have a maximum payload size of 52 and 242 bytes respectively, enabling not only faster transmission rates but also bigger packets [3]. Similarly, higher SF means that the time on air of a packet is also higher, i.e., the time between sending at node A and receiving it at node B is longer, compared to lower SFs. For an 11-byte payload time on air ranges from SF10 = 371 ms to SF7 = 61 ms [3]. In this project, this has an impact on the routing protocol, as each packet delivery is extended on higher SFs. Moreover, the scenarios often result in meshes on lower ADRs, i.e., higher SFs, which means slower transmissions, longer time on air and increased power consumption. On the other hand, the evaluation of the third scenario especially shows that the controller can improve all these factors as long as the individual links are strong and reliable. Both building a mesh and picking up nodes in the rendez-vous channel also occur in scenario 3 and it suggests that these converge more reliably with the three nodes with strong links. Therefore, securing the best locations possible for a mesh is of paramount importance for this project.

Chapter 6

Summary, Conclusions and Future Work

In this thesis, a LoRa mesh with SDN-based mechanisms was implemented for Raspberry Pis. The selected hardware parts are Raspberry Pis 3B/4 and the E32-868T20D module, which includes the Semtech SX1276 LoRa chip. The Raspberry Pis were flashed with Raspberry Pi OS Lite and put into a headless setup with SSH activated to enable access once they were deployed. Each node consists of a RPI with the E32-868T20D connected through the serial port, allowing the nodes to communicate over LoRa PHY and forming the test-bed across buildings of the University of Zürich.

Architectural decisions had to be taken before starting the implementation as well as during the implementation as some new problems arose. The implementation of an SDN-based LoRa mesh was achieved and the complete implementation may be followed on GitHub. In the thesis the most important aspects of the implementation such as a new node joining and the controller altering the transmission rate were explained and discussed.

Further, the nodes were deployed across buildings of UZH forming a test-bed enabling evaluation of the project in a field test. Due to some nodes having locations that provided weak links in the mesh, the link state databases of the nodes in the developed mesh not always reliably converge. In the majority of cases the link state databases converge and include all nodes participating, therefore showing that the implemented concept achieves its goal. Removing these two nodes completely further revealed, that the controller can manage the network in controlling the air data rate of the E32-868T20D LoRa modules efficiently. Moreover, it could be shown that the controller picks the highest possible air data rate on which a mesh still includes all nodes. The controller in this project successfully increases the data rate, reduces the time on air and the energy consumption after the LoRa mesh is formed. The configuration of the controller could still be improved in automatically identifying the weak links in order to exclude these nodes from the

mesh, because they are slowing down the transmission rate. On the other hand, such an extension would slow down the decision making of the controller to increase or decrease the air data rate.

However, the test-bed of this thesis was not optimal as two nodes were the reason to regularly decrease the speed. Evaluating this project again with five nodes with better locations suggests leading to better results with five nodes compared to this thesis, because the third scenario was running very promising with the three well positioned nodes usually staying on the higher channels including all nodes. Further, antenna optimization was not considered and the supported number of nodes is limited due to the packet structures to 250 nodes. Having the maximum amount of nodes would also lead to a huge overhead in forming a mesh as the link state databases need to synchronize all 250 entries. At this point, introducing areas of OSPF might lead to better performance.

In the future, different enhancements might be possible. As this thesis uses Raspberry Pis and the E32-868T20D, one approach could be using a more powerful LoRa module. For example the E32-868T30D [50], which is the exact same module with more transmission power. This could already be enough to make the same test-bed perform well with all five nodes. In addition, this project could be reused with Raspberry Pis and other LoRa modules that have a similar interface to the E32 Library. Another approach could be a time synchronization across the nodes to allow for a more organized and efficient routing protocol, i.e., reducing the amount of unnecessary LSAs sent. In general, OSPF has its limitations in this project due to the links not always being bidirectional. Therefore, another idea could be to extend this project with additional states including information regarding link stability. The controller could rate each nodes stability to determine which nodes should be included or dropped. Contrarily, a different routing protocol could be adapted.

Bibliography

[1] E. Schiller, S. Weber, and B. Stiller, "Design and Evaluation of an SDR-based LoRa Cloud Radio Access Network," 2020. [Online]. Available: www.zora.uzh.chYear:2020URL:https://doi.org/10.5167/uzh-189754

[2] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne, "Understanding the Limits of LoRaWAN," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 34–40, 2017.

[3] Semtech Corporation, "LoRa and LoRaWAN: Technical overview," 2021. [Online]. Available: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan

[4] LoRa Alliance, "LoRaWAN ® Specification v1.0.4," LoRa Alliance, Tech. Rep., 2020. [Online]. Available: https://lora-alliance.org/resource_hub/lorawan-104-specification-package/

[5] T. H. To and A. Duda, "Simulation of LoRa in NS-3: Improving LoRa Performance with CSMA," *IEEE International Conference on Communications*, vol. 2018-May, 7 2018.

[6] D. Magrin, M. Centenaro, and L. Vangelista, "Performance evaluation of LoRa networks in a smart city scenario," *IEEE International Conference on Communications*, 7 2017.

[7] Arduino, "Products." [Online]. Available: https://www.arduino.cc/en/main/products

[8] Raspberry Pi, "Products," 2021. [Online]. Available: https://www.raspberrypi.org/products/

[9] Semtech, "SX1276." [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-core/sx1276

[10] C. Ebi, F. Schaltegger, P. A. Rüst, and F. Blumensaat, "Synchronous LoRa mesh network to monitor processes in underground infrastructure," *IEEE Access*, vol. 7, pp. 57 663–57 677, 2019. [Online]. Available: https://digitalcollection.zhaw.ch/handle/11475/17190

[11] A. C. Anadiotis, L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SD-WISE: A Software-Defined WIreless SEnsor network," *Computer Networks*, vol. 159, pp. 84–95, 8 2019.

[12] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 10 2010.

[13] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 3 2002.

[14] B. Stiller, E. Schiller, and C. Schmitt, "An Overview of Network Communication Technologies for IoT," Communication Systems Group CSG, Department of Informatics IfI, University of Zürich UZH, Tech. Rep., 2021.

[15] "Homepage - LoRa Alliance®." [Online]. Available: https://lora-alliance.org/

[16] P. Robyns, P. Quax, W. Lamotte, and W. Thenaers, "A Multi-Channel Software Decoder for the LoRa Modulation Scheme," 2018.

[17] N. Varsier and J. Schwoerer, "Capacity limits of LoRaWAN technology for smart metering applications," *IEEE International Conference on Communications*, 7 2017.

[18] R. Piyare, A. L. Murphy, M. Magno, L. Benini, F. Bruno Kessler, and I. piyare, "On-Demand TDMA for Energy Efficient Data Collection with LoRa and Wake-up Receiver," 2018.

[19] B. Reynders, Q. Wang, P. Tuset-Peiro, X. Vilajosana, and S. Pollin, "Improving reliability and scalability of LoRaWANs through lightweight scheduling," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1830–1842, 6 2018.

[20] H. C. Lee and K. H. Ke, "Monitoring of Large-Area IoT Sensors Using a LoRa Wireless Mesh Network System: Design and Evaluation," *IEEE Transactions on Instrumentation and Measurement*, vol. 67, no. 9, pp. 2177–2187, 9 2018.

[21] J. Dias and A. Grilo, "Multi-hop LoRaWAN uplink extension: specification and prototype implementation," *Journal of Ambient Intelligence and Humanized Computing 2019 11:3*, vol. 11, no. 3, pp. 945–959, 2 2019. [Online]. Available: https://link.springer.com/article/10.1007/s12652-019-01207-3

[22] G. Zhu, C. H. Liao, T. Sakdejayont, I. W. Lai, Y. Narusue, and H. Morikawa, "Improving the Capacity of a Mesh LoRa Network by Spreading-Factor-Based Network Clustering," *IEEE Access*, vol. 7, pp. 21 584–21 596, 2019.

[23] J. R. Cotrim and J. H. Kleinschmidt, "LoRaWAN Mesh Networks: A Review and Classification of Multihop Communication," *Sensors (Basel, Switzerland)*, vol. 20, no. 15, pp. 1–21, 8 2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7435450/

[24] E. Sisinni, D. F. Carvalho, P. Ferrari, A. Flammini, D. R. C. Silva, and I. M. Da Silva, "Enhanced flexible LoRaWAN node for industrial IoT," *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, vol. 2018-June, pp. 1–4, 7 2018.

[25] E. Sisinni, P. Ferrari, D. Fernandes Carvalho, S. Rinaldi, P. Marco, A. Flammini, and A. Depari, "LoRaWAN Range Extender for Industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5607–5616, 8 2020.

[26] D. Tanjung, S. Byeon, D. H. Kim, and J. Deok Kim, "OODC: An Opportunistic and On-Demand Forwarding Mechanism for LPWA Networks," *International Conference on Information Networking*, vol. 2020-January, pp. 301–306, 1 2020.

[27] M. N. Ochoa, A. Guizar, M. Maman, and A. Duda, "Evaluating LoRa energy efficiency for adaptive networks: From star to mesh topologies," *International Conference on Wireless and Mobile Computing, Networking and Communications*, vol. 2017-October, 11 2017.

[28] Misbahuddin, M. S. Iqbal, and G. W. Wiriasto, "Multi-hop uplink for low power wide area networks using LoRa technology," *2019 6th International Conference on Information Technology, Computer and Electrical Engineering, ICITACEE 2019*, 9 2019.

[29] M. O. Farooq, "Introducing scalability in LoRa-based networks through multi-hop communication setups," *2019 IEEE Global Communications Conference, GLOBECOM 2019 - Proceedings*, 12 2019.

[30] Open Networking Foundation, "Software-Defined Networking (SDN) Definition - Open Networking Foundation," 2021. [Online]. Available: https://opennetworking.org/sdn-definition/

[31] McKeownNick, AndersonTom, BalakrishnanHari, ParulkarGuru, PetersonLarry, RexfordJennifer, ShenkerScott, and TurnerJonathan, "OpenFlow," *ACM SIGCOMM*

Computer Communication Review, vol. 38, no. 2, pp. 69–74, 3 2008. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/1355734.1355746

[32] V. G. Nguyen, A. Brunstrom, K. J. Grinnemo, and J. Taheri, "SDN/NFV-Based Mobile Packet Core Network Architectures: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1567–1602, 7 2017.

[33] T. Luo, H. P. Tan, and T. Q. Quek, "Sensor openflow: Enabling software-defined wireless sensor networks," *IEEE Communications Letters*, vol. 16, no. 11, pp. 1896–1899, 2012.

[34] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo, "Software defined wireless networks: Unbridling SDNs," *Proceedings - European Workshop on Software Defined Networks, EWSDN 2012*, pp. 1–6, 2012.

[35] GudeNatasha, KoponenTeemu, PettitJustin, PfaffBen, CasadoMartín, McKeownNick, and ShenkerScott, "NOX," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 7 2008. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/1384609.1384625

[36] D. Erickson, "The Beacon OpenFlow controller," *HotSDN 2013 - Proceedings of the 2013 ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pp. 13–18, 2013.

[37] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, WoWMoM 2014*, 10 2014.

[38] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed SDN OS," *HotSDN 2014 - Proceedings of the ACM SIGCOMM 2014 Workshop on Hot Topics in Software Defined Networking*, pp. 1–6, 2014. [Online]. Available: https://dl.acm.org/doi/10.1145/2620728.2620744

[39] A. Kozłowski and J. Sosnowski, "Energy Efficiency Trade-Off Between Duty-Cycling and Wake-Up Radio Techniques in IoT Networks," *Wireless Personal Communications*, vol. 107, no. 4, pp. 1951–1971, 8 2019. [Online]. Available: https://link.springer.com/article/10.1007/s11277-019-06368-0

[40] J. Moy, "OSPF Version 2," RFC 2328, 4 1998. [Online]. Available: https://rfc-editor.org/rfc/rfc2328.txt

[41] Renuka Narayanan, "Shortest Path Problem Between Routing Terminals – Implementation in Python," 2 2020. [Online]. Available: https://www.geeksforgeeks.org/shortest-path-problem-between-routing-terminals-implementation-in-python/

[42] Reichelt elektronik, "DELOCK 12591 Funkmodul 868 MHz TTL Pfostenstecker SMA Buchse, 20dBm," 2021. [Online]. Available: https://www.reichelt.com/ch/de/funkmodul-868-mhz-ttl-pfostenstecker-sma-buchse-20dbm-delock-12591-p290488.html

[43] Raspberry Pi Foundation, "Raspberry Pi 3 Model B," 2021. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

[44] "Raspberry Pi OS – Raspberry Pi," 2021. [Online]. Available: https://www.raspberrypi.org/software/

[45] JTECH, "EBYTE E32 LoRa Transciever - Advanced Arduino Setup," 12 2019. [Online]. Available: https://www.youtube.com/watch?v=cmN9zEmoXjw

[46] Lloyd Rochester, "Ebyte e32 LoRa Module," 9 2020. [Online]. Available: https://lloydrochester.com/post/hardware/ebyte-e32-lora-getting-started/

[47] "Google Maps." [Online]. Available: https://www.google.ch/maps

[48] "UZH - Pläne und Orientierung." [Online]. Available: https://www.plaene.uzh.ch/

[49] "LoRa-Shop.ch | Nebra IP67 Waterproof/Weatherproof Enclosure." [Online]. Available: https://www.lora-shop.ch/nebra-ip67-waterproof-weatherproof-enclosure

[50] Delock, "Delock Produkte 12593 IoT LoRa Funkmodul 868 MHz 30 dBm TTL (3,3 V) Pfostenstecker > SMA Buchse." [Online]. Available: https://www.delock.com/produkte/1911_IoT-LoRa/12593/merkmale.html

Abbreviations

ADR	Air Data Rate (of the E32-868T20D)
CSS	Chirp Spread Spectrum
IoT	Internet of Things
GW	Gateway
KPI	Key Performance Indicator
LSA	Link State Advertisement
LSDB	Link State Data Base
LSR	Link State Request
OS	Operating System
OSPF	Open Shortest Path First
PCE	Path Computation Element
PDR	Packet Delivery Ratio
RF	Radio Frequency
RPI	Raspberry Pi 3B / 4
SDN	Software Defined Networking
SSH	Secure Shell
TDMA	Time Division Multiple Access
UART	Universal Asynchronous Receiver Transmitter
UZH-VPN	Virtual Private Network of the University of Zürich
WSN	Wireless Sensor Network

List of Figures

List of Tables

List of Algorithms

Appendix A

Installation Guidelines

Follow the instructions below to setup an SDN-based LoRa mesh:

A.1 Hardware

- Flash Raspberry OS lite on to the SD-card using Raspberry Pi Imager
- Create plain ssh file on the boot drive (file has no ending)
- Enable serial port through:

```
$ sudo raspi-config
```

- Add user to groups dialout, tty and gpio,

```
$ sudo adduser pi tty, gpio, dialout
```

- Wire the E32-868T20D LoRa module to the Raspberry Pi as shown in table A.1

Table A.1: Wiring between RPI and E32-868T20D adapted from [46].

RPI Pin	E32-868T20D Pin
+5V	VCC
GND	GND
GPIO23	M0
GPIO24	M1
GPIO18	AUX
GPIO14 UART TX	RX
GPIO15 UART RX	TX

A.2 Software

This project requires third party packages. To install them, use the following bash commands

- install git

```
$ sudo apt install git
```

- install autotools and autoconf

```
$ sudo apt-get install autotools-dev
$ sudo apt-get install autoconf
```

- install numpy for python3

```
$ sudo apt-get install python3-numpy
```

To get the Project including the E32 Library Version 1.9 get the Github Repository and install the Library as follows:

-
```
$ git clone https://github.com/dreiss94/ebyte_sx1276.git
```

- Alternatively, get the code as a zip and unpack it on the SD card
- Navigate to the folder ebyte_sx1276 and run the following commands:

```
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

- Check version with:

```
$ e32 -h
```