TECHNISCHE
UNIVERSITÄT
DARMSTADT

COVERAGE OF LORA GATEWAYS, TRANSCEIVERS, ANTENNAS

TOBIAS THIEL, CHRIS-BENEDIKT VENN

Project

September 2, 2022

Secure Mobile Networking Lab
Department of Computer Science
Technische Universität Darmstadt

SECURE MOBILE NETWORKING

Tobias Thiel, Chris-Benedikt Venn, *Coverage of LoRa Gateways, Transceivers, Antennas*, Project, Technische Universität Darmstadt, 2022.

Date of submission: September 2, 2022

Advisor: Prof. Dr.-Ing. Matthias Hollick
Supervisor: Frank Hessel, M. Sc.

Secure Mobile Networking Lab
Department of Computer Science
Technische Universität Darmstadt

ABSTRACT

LoRaWAN is still a fairly new technology and almost no large scale field test has been done so far in recent research. We advance the study by providing a large scale field test containing more than 7,000 data points around the city of Darmstadt, Germany. Furthermore, we built our own real-time application with advanced filter functions containing a separate frontend and backend in order to evaluate the collected data. Constructing our own application server improved the efficiency of measuring and evaluation of the data. We tested two different node setups, two different antennas and two different spreading factors on three different gateways. In order to evaluate the results, we proposed different coverage maps including the classification in distance zones. We concluded that the range is quite different for each of the gateways and highly depends on the surrounding area. The overall furthest link we measured in Darmstadt was 1.76 km away from the gateway. Comparing the two nodes, the Feather Node had a slight advantage over the Nucleo node and the Nucleo Node could improve its range in certain spots by switching to a longer antenna.

ZUSAMMENFASSUNG

LoRaWAN ist immer noch eine sehr neue Technologie und es gibt dafür in der Forschung kaum großflächige Feldversuche. Wir möchten die Forschung dahingegen erweitern, dass wir einen großflächigen Feldversuch mit über 7.000 Datenpunkten präsentieren. Die Messungen wurden ausschließlich in Darmstadt, Deutschland gemacht. Für das korrekte Auswerten der Messungen haben wir ein eigenes Backend geschrieben zusammen mit einer Echtzeit-Webanwendung mit erweiterten Filter Funktionen. Diese Webanwendung hat uns dabei geholfen die Effizienz und Evaluation der Feldversuche zu verbessern. Weiterhin haben wir zwei verschiedene Node Setups, zwei verschiedene Antennen und zwei verschiedene Spreading Faktoren an drei verschiedenen Gateways getestet. Für unsere Evaluation haben wir verschiedene Abdeckungskarten aufgeführt und die Datenpunkte in gewisse Distanzzonen eingeteilt. Am Ende kamen wir zu dem Ergebnis, dass die Reichweite sehr unterschiedlich ist für jedes der drei Gateways und sehr von den umliegenden Gebäuden abhängt. Der weiteste Link, den wir gemessen haben war 1,76 km weit weg vom Gateway. Der Vergleich der zwei Nodes hat ergeben, dass die Feather Node etwas höhere Reichweite hatte als die Nucleo Node. Allerdings konnten wir an einigen Punkten auch die Reichweite der Nucleo Node verbessern in dem wir eine längere Antenne verwendet haben.

CONTENTS

1 Introduction 1
- 1.1 Related Work . . . 1
- 1.2 Project Definition . . . 2
- 1.3 Contribution . . . 3
- 1.4 Architecture Design . . . 3
- 1.5 Project Schedule . . . 4

2 Background 7
- 2.1 Architecture of LoRaWAN . . . 7
- 2.2 Spreading Factors . . . 8
- 2.3 Adaptive Data Rate . . . 8
- 2.4 OTAA vs. ABP . . . 9

3 Setup 11
- 3.1 Nucleo Node Setup . . . 11
- 3.2 Feather Node Setup . . . 12
- 3.3 Server Setup . . . 12

4 Implementation 15
- 4.1 Overview and Architecture . . . 15
- 4.2 Hardware . . . 18
- 4.3 Backend . . . 20
- 4.4 Frontend . . . 23
- 4.5 Architectural Decisions . . . 26
- 4.6 Possibilities and Drawbacks . . . 26

5 Evaluation 29
- 5.1 Gateway Pankratiusstraße . . . 29
- 5.2 Gateway Mornewegstraße . . . 32
- 5.3 Gateway Adelungstraße . . . 35

6 Conclusions 43
- 6.1 Discussion . . . 43

Bibliography 45

LIST OF FIGURES

Figure 1 Project Architecture . . . 4
Figure 2 Project Schedule . . . 6
Figure 3 Network topology of LoRaWAN [15] . . . 7
Figure 4 Comparison of different LoRa Spreading factors [16] . . . 8
Figure 5 Wiring Arduino node . . . 11
Figure 6 Wiring diagram of the Feather Node [14] . . . 12
Figure 7 Pipeline of different server components . . . 12
Figure 8 Backend procedure . . . 21
Figure 9 React redux pattern hinted with used functions 25
Figure 10 Pankratiusstr. coverage area for both nodes using SF7 . . . 30
Figure 11 Pankratiusstr. coverage area for both nodes using SF12 . . . 30
Figure 12 Coverage of Arheilgerstr. (marked in black) with Feather Node SF12 . . . 31
Figure 13 Coverage of Arheilgerstr. (marked in black) with Nucleo Node SF12 using Antenna B . . . 32
Figure 14 Different distance zones of Pankratiusstr. gateway in 200 meter steps . . . 33
Figure 15 Mornewegstr. coverage area for both nodes using SF7 . . . 34
Figure 16 Mornewegstr. coverage area for both nodes using SF12 . . . 35
Figure 17 Mornewegstr. long distance route using Feather Node with SF12 and Antenna B . . . 36
Figure 18 Mornewegstr. long distance route using Nucleo Node with SF12 and Antenna B . . . 37
Figure 19 Different distance zones of Mornewegstr. gateway in 300 meter steps . . . 38
Figure 20 Adelungstr. coverage area for both nodes using SF7 . . . 38
Figure 21 Adelungstr. coverage area for both nodes using SF12 . . . 39
Figure 22 Adelungstr. gateway mounted on balcony . . . 40
Figure 23 Different distance zones of Adelungstr. gateway in 200 meter steps . . . 41

ACRONYMS

ABP	Authentication By Personalization
ADR	Adaptive Data Rate
GIS	Geographic Information System
GPIO	General Purpose Input Output
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
HTTP	Hypertext Transfer Protocol
IO	Input Output
JSON	JavaScript Object Notation
LoS	Line of Sight
MAC	Medium Access Control
MEO	Medium Earth Orbit
MQTT	Message Queuing Telemetry Transport
OTAA	Over The Air Authentication
RF	Radio Frequency
RSSI	Received Signal Strength Indication
SF	Spreading Factor
SNR	Signal-to-Noise Ratio
SPI	Serial Peripheral Interface
TX	Transmit Power

1 INTRODUCTION

In the recent past, low-power wide-area network (LPWAN) are gaining more popularity for their desirable features of high range and low energy consumption at the cost of low throughput. In particular, one protocol named LoRaWAN is growing in popularity. Between the years of 2015 and 2018, there were 4,240 hits on Google Scholar [10], while in the recent past from 2018 to 2022 the number of matching papers skyrocketed to 16,400 [11]. It was originally invented by Cycleo, a company located in France and was later acquired by Semtech [29]. With the above-mentioned characteristics, LoRaWAN can achieve a very high range of approximately 6 km in urban areas [7]. Due to a high variance of the LoRaWAN infrastructure, measurements can vary a lot for different environments. Since most research on LoRa focuses on few chosen locations to take measurements, we want to extend the study by providing a large scale field test as well as provide appropriate tooling for evaluating such surveys. In addition to that, an extensive field test is needed in Darmstadt in preparation for a large scale gateway deployment in order to measure the overall coverage in different locations of the city and to evaluate the shortcomings of different hardware setups.

1.1 RELATED WORK

Related work has been proposed by Bardram et al. in [4]. They conducted a field study in a Danish harbor with the goal of monitoring the occupation of boat spots using LoRaWAN and came to the conclusion that LoRa is well suited for that task. Their setup involved just one gateway placed at a height of 2 meters. The furthest distance they measured was 919 meters away with Received Signal Strength Indication (RSSI) values between -41 dB and -118 dB. Rahman and Suryanegara did another field test including Line of Sight (LoS) and Non-LoS communication of temperature measurements at the University of Indonesia campus [24]. They measured a LoS distance of up to 700 meters with RSSI values between -55 dB to -90 dB while their Non-LoS test involved distances of up to 400 meters with RSSI values between -62 dB to -99 dB. They concluded that LoRa is well suited for usage in the IoT world of both LoS and Non-LoS condition. Furthermore, Jovalekic et al. presented a LoRa transceiver with improved characteristics which allowed them to establish a LoS link of 112 km in distance using only low-cost, of-the-shelf, directional, rubber duck antennas [17]. More work has been done by Loriot et al. [18]. They

conducted a study at the 110 hectares large Scientific Campus of the University of Lille, France. Their study around 145 buildings and 100 km of urban networks showed a maximum covered distance of 1.2 km with an -18 dB Signal-to-Noise Ratio (SNR). Their gateway was placed at the first floor of a building on the campus. Another field test was done by Aref and Sikora [2]. With a gateway height of 22 m, they received a maximum coverage of 8.15 km with an RSSI of approximately -130 dB and an SNR of approximately -16.6 dB. In [21] Petajajarvi et al. measured the link quality on ground and on water. Receiving packets up to 15 km on ground and close to 30 km on water. For their ground test, the gateway was mounted on the roof rack of a car. For the on water study it was mounted to the radio mast of a boat.

1.2 PROJECT DEFINITION

This project is about creating a useful tool to evaluate LoRaWAN measurements, as well as providing large scale field tests in order to analyze the link coverage and the versatility of different LoRaWAN hardware setups and how they affect connectivity and range. In order to do so multiple field tests are conducted with varying hardware and antennas as well as different gateway locations. The field test is taking place in and around the city center of Darmstadt, Germany. The following research questions might be compelling and will be dealt with in the next couple of chapters.

- Does our constructed application improve the overall measurement and evaluation process?
- How is the overall coverage for the three deployed gateways?
- Can the coverage be extended with different spreading factors and different antennas?

Furthermore, we split our project into four phases:

1. During the initialization phase, we get familiar with the hardware and set up a basic network in order to test communication flow.
2. In the second phase, we conduct our first field test in Darmstadt using our default setup.
3. The third phase revolves around making multiple field test using different hardware setups as well as visualization of the results.
4. The last phase includes the wrap up of the project by finishing the documentation.

Furthermore, a beta version is proposed where an initial field test was conducted together with a visualization of the corresponding

results. The final version will include multiple field tests conducted with different hardware and an evaluation of the results. There should also be a high emphasis on reproducibility in order to reuse the setup for adding further measurements.

1.3 CONTRIBUTION

Our contribution involves a fully developed real-time asynchronous backend with push architecture design server written in Python, a frontend written in React with modern hook-based state flows including a global object store enabling a truly interactive web application. Furthermore, we made a large scale field test to evaluate the coverage in Darmstadt involving more than 7,000 data points which visualizes packets in real time as a circle on a map. The map features different circle sizes and colors depending on the RSSI and the SNR. Furthermore, it provides extensive filter functions including filters for time ranges, RSSI, SNR, altitudes, spreading factors, gateways and nodes. We also provide distance values for each packet in order to review its travelled distance. Additionally, we made it possible to export the filtered data into the GeoJSON format to make it usable with other applications. We tested two different Node setups, firstly a Nucleo Node based on the STM32L152RE microcontroller paired with an SX1276MB1MAS LoRa shield from Semtech and secondly an Adafruit Node containing the Adafruit Feather Mo LoRa. Both nodes were evaluated with Spreading Factor (SF) 7 and SF12, and the Nucleo node was tested with two different antennas to show an increase in range. The different node setups were tested on three gateways placed around the Darmstadt area.

1.4 ARCHITECTURE DESIGN

Figure 1 shows the basic idea of our architecture. The system should be as simple as possible, while being highly expandable if needed. It is planned to split backend logic from the LoRa architecture side, such that only a small interface at the application server level is needed to log measurements. The LoRa end-device or *node* is equipped with a Global Positioning System (GPS) module and should only transmit the current location in set intervals. The end-device sends its packets to the gateway which then forwards it to the network server. The network server does all the relevant processing and transfers the result to the application server which provides an interface for our logging and evaluation application. Our application receives *measurement points* containing timestamp, GPS coordinates, applied Transmit Power (TX) power, spreading factor, available bandwidth and type of antenna. The measurements are saved and can then be plotted onto a heatmap using the GPS coordinates and signal strength at the deployed gateway.

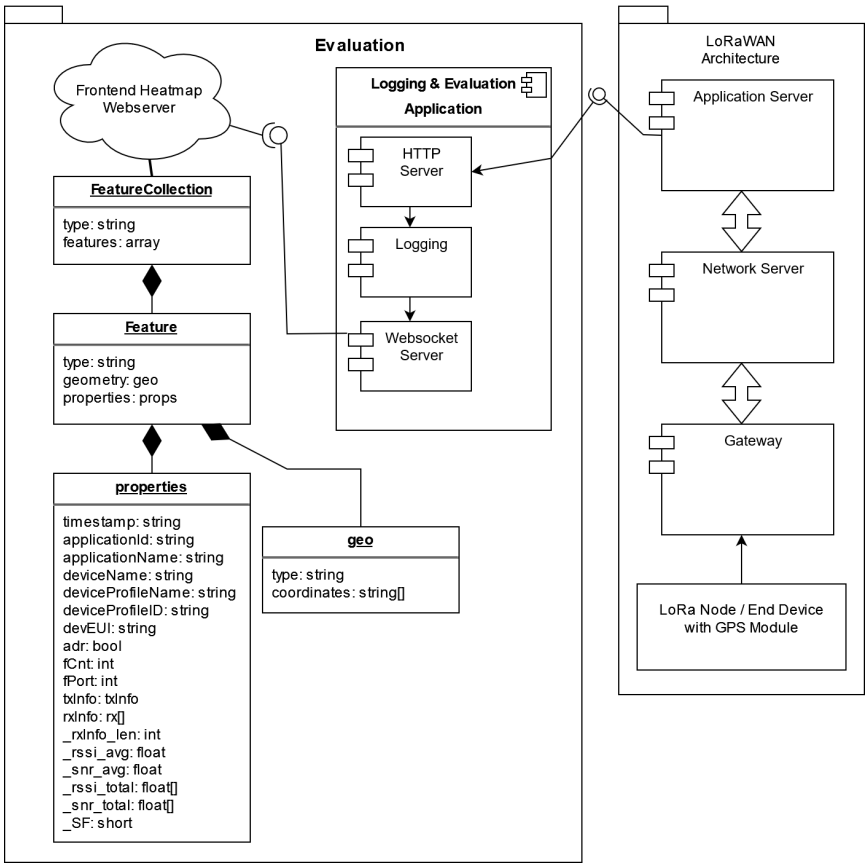

Figure 1: Project Architecture

1.5 PROJECT SCHEDULE

In Section 1.2, our planned schedule is separated into the four phases of initial definition, implementation, evaluation and documentation. The key milestones to accomplish the goal of measuring and comparing multiple LoRa controllers are the following.

- Connecting a LoRa node to a gateway and logging RSSI, SNR, spreading factor, frequency and bandwidth.
- Getting the GPS antenna up and running on the microcontroller/LoRa node side. Sending GPS information to the LoRa gateway.
- Successfully logging data from the microcontroller onto the SD card and logging it to the database.
- Implementing a backend for retrieving and analyzing incoming data.
- Testing multiple LoRa radio controllers, LoRa shields, antennas and spreading factors. Testing upper distance limits for reliable LoRa communication around the campus.

There is also a project schedule which can be found in Figure 2.

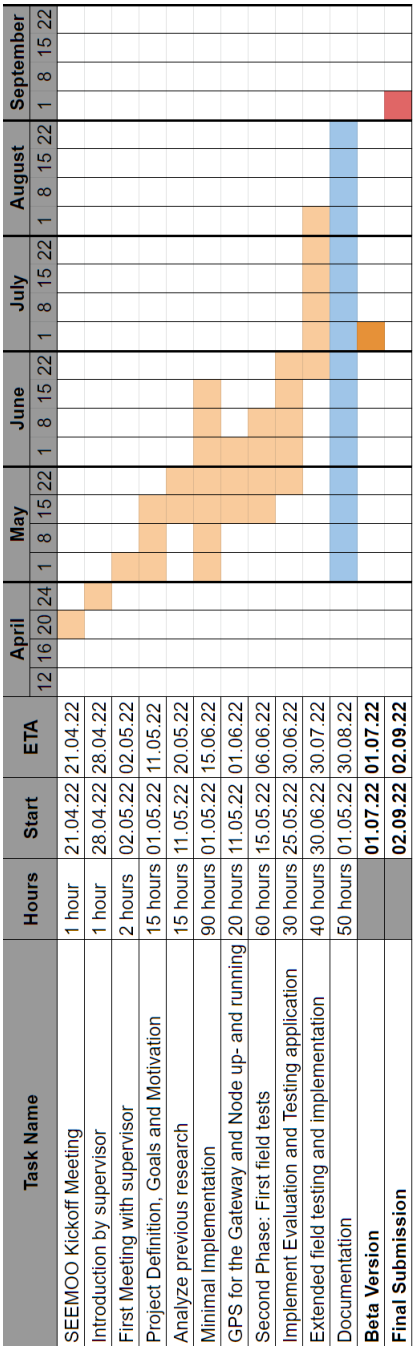

Figure 2: Project Schedule

2 BACKGROUND

2.1 ARCHITECTURE OF LORAWAN

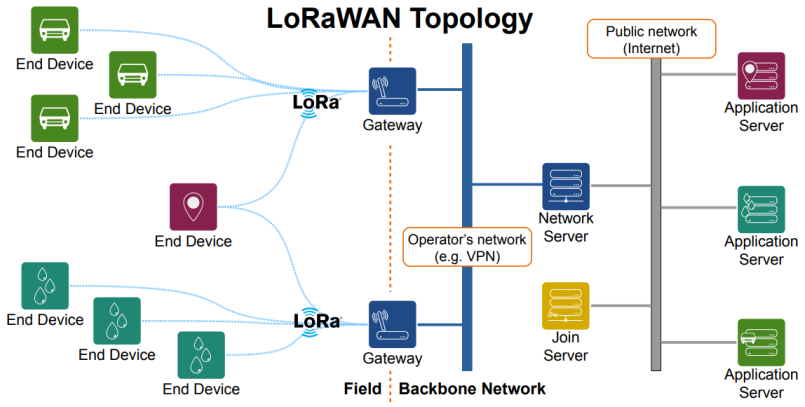

Figure 3: Network topology of LoRaWAN [15]

LoRaWAN is a Low Power Wide Area Network (LPWAN) MAC layer protocol introduced in 2015. It operates on the license free ISM bands and has a public specification by the LoRa Alliance. It is based on the LoRa physical layer which is proprietary. LoRaWAN has many desirable features such as long range communication, low energy consumption and low cost in terms of devices and maintenance. The payload size is limited though with just a few hundred bytes at maximum. The topology of a LoRaWAN Network can be found in Figure 3. It consists of end devices, gateways, a network server, a join server and multiple (or just one) application server. The end devices are the low power embedded devices (e.g. temperature sensors) which send their data to a gateway. Since the LoRaWAN network architecture has a star topology, the end devices can only communicate with the gateways and not directly with each other. It does not matter to which gateway an end device sends its information since it is not bound to a specific gateway. The gateway has no logic other than forward the packet to the network server. The network server controls the whole communication flow and is the brain of the system. Furthermore, an application server may also be used in order to visualize the received data. There can also be a dedicated join server which handles new end devices joining the network.

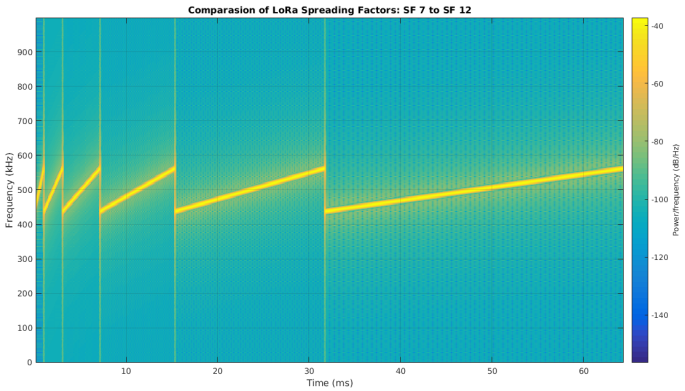

Figure 4: Comparison of different LoRa Spreading factors [16]

2.2 SPREADING FACTORS

LoRa uses a spread spectrum modulation called Chirp Spread Spectrum where chirps encode the data [9]. A chirp, abbreviated for "compressed high intensity radar pulse" is a signal moving up or down with a specific speed. The spreading factor determines the speed of the chirps where lower spreading factors result in faster chirps and thus a higher transmission rate and higher spreading factors in lower chirps with a lower transmission rate. An illustration can be found in Figure 4. The illustration covers the common spreading factors between seven and twelve (SF7 to SF12). Starting on the left with SF7 we can see that from SF7 to SF8 the length of the chirp is roughly doubled. Additionally, an increase in the SF results in a greater penetration and range of the transmission. This allows the network to dynamically adjust the spreading factors to increase or decrease the data rate for every device at the cost of range. Furthermore, the spreading factors are orthogonal to each other, which means that signals with different spreading factors, transmitted on the same frequency channel, at the same time do not interfere.

2.3 ADAPTIVE DATA RATE

In LoRaWAN, Adaptive Data Rate (ADR) is a mechanism that lets the network server control data rate, bandwidth, and spreading factor [8]. The goal is to optimize the data rates and transmission power of each device, that it uses less airtime to transmit data. Less airtime means less antenna usage which results in lower power consumption. End devices which are closer to the gateways should use low spreading factors and a higher data rate, while devices further away should use higher spreading factors because they need to penetrate through more objects. ADR should only be enabled when the end device is stationary. For our use case this setting is not relevant and has been deactivated in the upcoming field tests.

2.4 OTAA VS. ABP

There are two different activation modes in LoRaWAN in order for end devices to join the network, namely Over The Air Authentication (OTAA) and Authentication By Personalization (ABP) [1]. In this chapter we discuss the differences between those two. Starting off with ABP, which is the simpler one, an end device has a fixed device address and session keys for a specific network, hard-coded into their memory. They remain the same throughout the lifetime of the end device. Using this mode, there is no join procedure. The node just transmits data immediately to the specific network using the just discussed parameters. Now in OTAA on the other hand, end devices now only have an application key hard-coded into their memory. With this application key, an end device performs a join procedure with the join server of the given network. After the join procedure a dynamic device address is assigned to the end device and the application key is used to derive the session keys. Thus, device address and session key change on each new session. Coming back to ABP, it has some substantial drawbacks we will discuss in the following. ABP end devices have a fixed device address which means an end device can only work in their predefined network and cannot join other networks. In OTAA this is possible because a new device address is assigned each new session which allows them to move to different networks. Furthermore, in ABP frame counters of the ABP devices should not be reset, otherwise messages from that device might get dropped by the network server. If the end device somehow loses power and its memory is erased, it will start his frame counter at 0 again which results in packet loss since the network server will most likely not accept packets with reoccurring frame counts. Additionally, when an ABP end device runs out of frame counters, he cannot send any more messages, thus hitting the end of its lifetime. Since the session keys are hard-coded into the end devices, they cannot be changed. If those session keys were ever leaked, this will be a serious security threat. In OTAA end devices renegotiate the session keys and frame counters whenever they establish a new session. Thus, the lifetime is not limited by the frame counter. Generally, OTAA can be seen as the superior method for most scenarios. In our use case, we do not really benefit from the explained advantages in OTAA since we do not really care about security and do not want to join other networks. Thus, we decided to use ABP for our measurements. It avoids any unnecessary steps and makes transmissions of the node as simple as possible.

3

SETUP

This chapter is about setting up our hardware components in order to communicate with each other. We first assembled all the hardware including our two nodes and one gateway and also shine some light on the software configurations during the installation process.

3.1 NUCLEO NODE SETUP

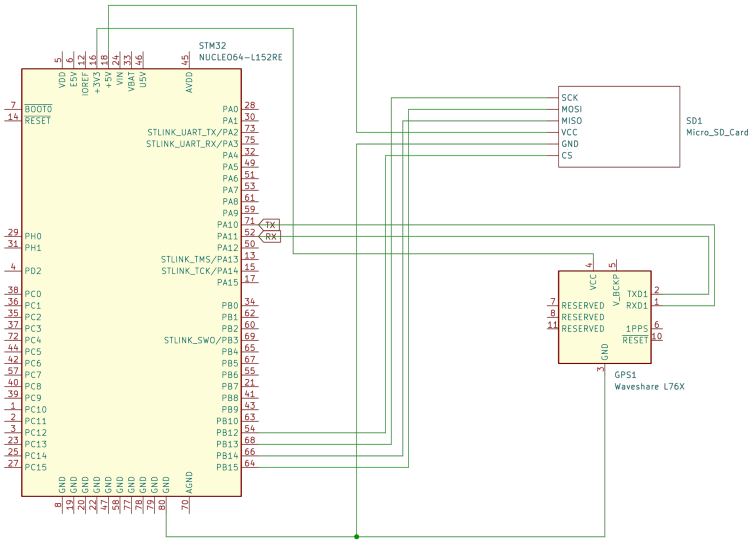

Figure 5: Wiring Arduino node

For our initial setup of the node the Nucleo STM32L152RE microcontroller [30] is used, paired with SX1276MB1MAS LoRa shield from Semtech [31]. In addition to that, a GPS module and an SD card reader is also connected to the Nucleo board. As shown in Figure 5, we connect an off the shelf SD card reader module via synchronous Serial Peripheral Interface (SPI) and a Waveshare L76X GPS breakout board via RX/TX serial interface. The Figure also omits the connected LoRa Shield. In order to program the Nucleo board, PlatformIO [22] and the Arduino platform are used. A reference implementation for a LoRa Node using ABP or OTAA is also present on GitHub [20]. The payload consists of the current GPS coordinates of the node directly pulled from the GPS module. The node also logs every outgoing packet including frame count, timestamp and GPS information on the external SD card. Moreover, we have disabled adaptive data rate (ADR) because our device is constantly moving and in that case it is recommended to turn ADR off. Ultimately, the usage of the Arduino

platform failed. Unfortunately, we had range issues with the library, and we were only able to receive packets at really low range in line of sight. Considering the time aspect, we decided to switch to a RIOT OS implementation by Frank Hessel called lora-gps-tracker, accessible at [14]. After switching to RIOT OS, our SD card logging was not usable anymore and had to be implemented again. Thus, we decided to keep going without it.

3.2 FEATHER NODE SETUP

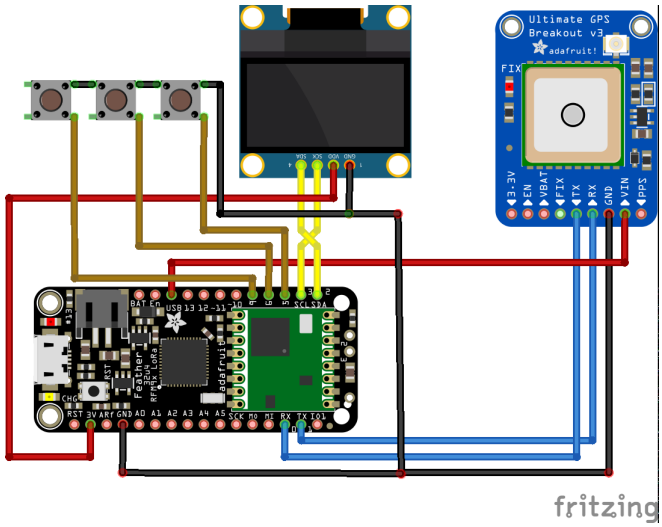

Figure 6: Wiring diagram of the Feather Node [14]

The second node we tested, was an Adafruit Feather Mo LoRa paired with the Adafruit Ultimate GPS Breakout v3, some push buttons to change spreading factors on the fly and an OLED display to show current debug information and an 868 MHz antenna, which is approximately 20.5 cm long. For the implementation of the node, we again used the example implementation at [14] which is based on RIOT OS [26]. On the GitLab page, there is a wiring diagram provided, which is also visualized in Figure 6.

3.3 SERVER SETUP

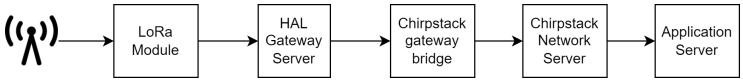

Figure 7: Pipeline of different server components

Beginning with the hardware, we first set up our gateway. The gateway consists of a Raspberry Pi 3, paired with a WM1302 Pi hat [34], the corresponding LoRa module [33] and an 868 MHz antenna. As

explained before, the LoRaWAN architecture requires different servers. Starting with the gateway server, which is hosted on the Raspberry Pi acting as the gateway in this scenario. A reference implementation is given by Semtech with their Hardware Abstraction Layer (HAL) [13]. The HAL receives the packets directly from the LoRa module and forwards them to the network server. For the network server, we chose our own instance of Chirpstack [6] which is hosted externally on the infrastructure of TU Darmstadt. The HAL does not send the packets directly to the network server though, instead it forwards them to the so called Chirpstack gateway bridge, which acts as a forwarder in between. The Chirpstack gateway bridge then forwards the packets from the HAL directly to the Chirpstack network server via Message Queuing Telemetry Transport (MQTT). In order to visualize the data we set up our own application server. For that, the Chirpstack network server sends every packet to the application server via a Hypertext Transfer Protocol (HTTP) POST request. The application server has its own logic to receive and process the packets. A visualization can be found in Figure 7 which shows the pipeline of every packet received at the gateway antenna. For our own instance of Chirpstack, we used the provided Docker image from their website [5]. The Chirpstack server is installed on the servers of TU Darmstadt and as explained before the HAL forwards the packets to the Chirpstack gateway bridge. Therefore, we have to enter the IP address of our Chirpstack gateway bridge into the config file of the HAL as well as the (freely chosen) ID of that particular gateway. Since the Chirpstack gateway bridge and the HAL are still located on the gateway itself, which is a Raspberry Pi, they are hosted on the same device. Inside the Chirpstack gateway bridge, we now enter the IP of the Chirpstack Server hosted on the TU Darmstadt servers. For the packet transmission authenticated MQTT is used. Switching over to the configuration in Chirpstack, we first created service and device profiles for our devices. We chose the LoRaWAN Medium Access Control (MAC) version of 1.0.2 and used ABP as join method. On top of that, we created gateways with their corresponding gateway ID as well as applications with their corresponding device EUI, device address, network session key and application session key. If everything is configured correctly, new packets will now be received from the corresponding gateway and allocated to the right application. Finally, in order for our application server to receive all packets that the Chirpstack network server receives, we make use of the HTTP integration in Chirpstack. It can be configured for every application profile in Chirpstack and only needs the IP of the corresponding application server.

4 IMPLEMENTATION

In this chapter, we take a deep dive into the chosen architecture and explore the use cases, capabilities and downsides. At the end of this chapter there should be a clear understanding of the system and our path to the current state of development.

4.1 OVERVIEW AND ARCHITECTURE

To understand how we got to our current system, it is necessary to describe the following sections. The motivation that defined the underlying topic and goal of this project. Based on the goal, we defined a multitude of requirements that had to be accomplished. And those requirements outlined the scope of the system, its context and constraints. Furthermore, we split our system into three major components namely hardware, backend and frontend where the most important aspects are explained. At the end, there is also a discussion of our architectural decisions in a more high level manner. We want to measure the real world performance in terms of range with LoRa. A lot of research has been done about the long range capabilities of this technology. Additionally, there are only few papers using the LoRa technology for moving objects, e.g. [21]. For those scenarios the end device is mounted on top of a car's roof-rack, trucks, cranes or single assets [23]. It is important to remember that these measurements may be accurate and reproducible with the same hardware and in the same area, but for our use case and environment there a multiple factors to account for. Our measurements take place in the Rhine-Main Area inside the city center of Darmstadt. We also use different hardware and software setups as the already-mentioned papers. The Radio Frequency (RF) noise environment, topography and placement of the gateways are different, and therefore our results differ from those papers. Our motivation is not only creating reproducible results for this area with the used hardware, but also being able to do it in a much more efficient way. To be more precise, our goal is to create a real-time LoRa GPS tracking web application which is capable of basic analytics, can be used by multiple users simultaneously and is able to export the results for later use in common geographic information system applications. While researching the whole topic, we realized that there is virtually no existing application which satisfies our requirements. There are multiple applications that allow for GPS tracking of certain objects or mapping out the coverage of LoRa gateways for `The Things Network` or `Chirpstack`. The closest application, we found is

the TTN-Mapper [32]. Still, it is lacking many of our requirements. Our solution provides extensive filter functions that can be used to make rough statements about the coverage and performance of each node. For example, it is possible to filter the dataset for a specific time range, RSSI range, SNR range, altitude range, spreading factor range, visible clients or gateways. We not only want to measure and evaluate the performance of our LoRaWAN setup, but also provide a custom tool that reliably assists us while doing measurements.

Requirements

Requirements consists of functional features that must be present or should be available. Nonfunctional requirements do not specify certain functions, but describe the expected behavior, robustness and usability aspects. In our case both types of requirements are bound to our goals and questions we attempt to answer.

Functional requirements

- The system must be able to render incoming measurements as a layer on top of a world map.
- The system must provide real-time access to measurements made by active nodes.
- The system must be able to receive POST requests from the application server.
- The system must be able to store incoming data.
- The system must be able to serve multiple clients with real-time information about ongoing measurements.
- The system must be able to provide historical results to every client.
- The system must provide data integrity and correct values.
- The system must provide filtering functionality.
- The system must provide a filter by time and date function.
- The system must provide a filter by one or more clients.
- The system must provide a filter by one or more gateways.
- The system must be able to do the already mentioned requirements simultaneously for many clients.
- The system should be able to allow for exporting data to a RFC 7946 [25] compliant format for external use.

- The system should be able to allow for multiple methods displaying results on a map.
- The system should be able to display a rough heatmap of the covered area.
- The system should be able to display the number of visible data points.
- The system should be able to be scalable to multiple regions, application servers, gateways, nodes and users.

Nonfunctional requirements

- The system must be able to provide an assistance for those that measure LoRa RF coverage.
- The system must provide easy filter and exporting functionality to the user.
- The system should enable the user to make statements about coverage based on a contextual visualization.
- The system should provide a seamless experience for multiple device formats.
- The system should provide an easy way to realign the map to the GPS location of the latest received packet.

System Scope, Context and Constraints

The scope of our system consists of its three main components. The hardware is communicating one-way over LoRa to the gateway. The gateway receives the data point and forwards it to the application server. The application server sends the collected data from one or many gateways POST it to our backend HTTP server. The backend now saves the data permanently for later analysis and forwards it to multiple web socket clients. Domains are clearly separated, and each domain can be replaced if the interface requirements are met. In its current form, the system only uses standardized internet protocols such as HTTP, web sockets and LoRaWAN as communication medium. The overall context is able to visualize LoRa coverage measurements in real-time and evaluate the performance and properties of different modulations i.e. spreading factors as well as providing tangible results and tooling to improve methods of LoRa coverage analysis. The whole system is tested and evaluated only in Darmstadt, Germany. The real-time aspect is one of our most important functions. Moreover, we want to clarify what our system is not capable of in the current state. Our web application does not replace professional geographic

information applications or analysis tools. It is not competing against the TTN-Mapper project or any other LoRa GPS tracking application. The real-time aspect and feedback of our application enables us to view incoming data and to plan where we move next. Now the next move can be planned depending on the recent signal strength or link quality. In the current state of development we have no back-channel that allows for bidirectional communication between the application server and end device, although the foundation for such an extension has already been built and can be implemented without trouble. This will be discussed further in the following sections.

4.2 HARDWARE

As for many other projects using LoRa technology, development boards like STM32, ESP32 or small computers like a Raspberry Pi play a major role. In this section we explore multiple software libraries and hardware configurations for our mobile end device which continuously transmits its GPS location.

Microcontroller

As in section 3.1 mentioned, we use the Nucleo STM32L152RE microcontroller from ST Microelectronics with a SX1276MB1MAS LoRa shield from Semtech. We tried this combination of microcontroller + shield with Arduino Platform and a RIOT implementation. We went through multiple iterations and modifications and encountered several issues that had to be addressed. In figure 5 the wiring for connecting the `GPS module` and the `SD card module` to the Nucleo STM32L152RE microcontroller.

Arduino approach

Arduino is a physical computing platform for microcontrollers with many digital and analog General Purpose Input Output (GPIO) pins. The programming language is C and C++ and Arduino as a framework is based on `processing` with its typical `void setup()` and `void loop()` functions which are called once and looped over. Easy access for beginners, a large community and a lot of libraries are just a few of the benefits using Arduino. The reasoning behind going for the Arduino platform, was the vast community supporting it, such that in a case where things do not work, there are many people online who could help. There is also a large amount of libraries for almost every need available.

GPS

The Global Positioning System is a network of many satellites which constantly broadcast their position back to earth. Due to the high altitude of approximately 20,200 km on the Medium Earth Orbit (MEO), the received signal on earth is very weak and can even be blocked by light obstructions. In order to calculate the current position of the receiver, it listens for at least 3 to 4 satellites until it is able to calculate the position. This can take up to a few minutes if a receiver's last power up is a while ago.

SD Card logging

The SD card logging part of the project was at first a crucial element for our evaluation. The goal was to identify single packets by their frame count and timestamp if packet loss occurred. There is no acknowledgement for each packet and if one would be available then there is still the possibility that it gets lost too. Logging to an SD card is only implemented with the Arduino approach. Considering the connection problems we had using the Arduino library, we eventually switched to the lora-gps-tracker [14] and in the end there was little time left to integrate the SD logging part into the new implementation. The logging works by simply opening a new file with incremental versioning and log the current position with frame count line by line. We took the example data logger from the SdFat library by Bill Greiman [12] and changed a few things. There was an issue with the SPI interface because the first port was already used by the LoRa Shield. A quick workaround was to set the following `#define USE_STANDARD_SPI_LIBRARY 2` inside the `SdFatConfig.h`.

RIOT approach

In a nutshell RIOT OS is an operating system for the Internet of Things. As they said on their website: "RIOT powers the Internet of Things like Linux powers the Internet" [27]. For this project, we only made slight changes to the already existing lora-gps-tracker [14] in order for it to work with our Nucleo Node which had just one button instead of three and mainly focused on building the platform for evaluation. We switched the library due to the fact that the Arduino `mcci-catena/MCCI LoRaWAN LMIC library` had issues with compatibility over a larger range of end devices, and we encountered an issue with the transmission power not affecting the RSSI and SNR value. The most probable cause was that a necessary build flag was not set that enabled transmitting over one or both external antennas.

4.3 BACKEND

The backend system is the heart of our real-time evaluation platform, and it took quite a while to figure out what our actual requirements are. Deciding on which transmission methods, interfaces, programming paradigm or data structures fits our needs the best. The following subsections, we dive into various methods of delivering information between systems connected over the internet and methods of handling parallelism.

History of approaches

We started setting up a Chirpstack instance with Docker on a Raspberry Pi with a LoRa Shield. After setting up the end device with Arduino we were able to receive GPS coordinates, RSSI and SNR inside Chirpstack. Our first approach was to use the MQTT interface from Chirpstack to receive and save the data into a file. The underlying principle of this messaging protocol is rather simple, we use a publish-subscriber architecture which allows for buffering and multicasting one message to a group of receivers. While this was rather simple to implement in python, we encountered another problem. If we plan to do anything more than simple logging and offline analysis as virtually every other LoRa RF coverage evaluation paper, then we have to send the data further. As already stated in the Section 4.1 we want to serve multiple clients at the same time, while receiving and logging incoming data. Thus, adding a synchronous web socket server for pushing data will not work in this case. We tried putting each service in one thread or process, but this was not compatible with any of the web socket implementations that we have used. Another proposal might be to run two python scripts that are independent of each other. These could be connected via ZeroMQ, a messaging protocol designed for horizontal integration into backend services. The communication worked but the script with the web socket server in it was not able to serve multiple clients. Finally, we decided to give python asyncio a try.

Implementation

The final decision to remove MQTT support and instead open up an HTTP server to receive post requests from the Chirpstack application server was made together with our advisor. This allows a tighter integration in the already existing infrastructure and keeps complexity to a minimum. Asyncio is a library for concurrent code using the async/await syntax. This enables for high-performance and especially Input Output (IO) intensive applications. Our system has no compute intensive tasks. The only waiting time is network delay, waiting until

data is fully received, waiting until data is written into a file or later a database.

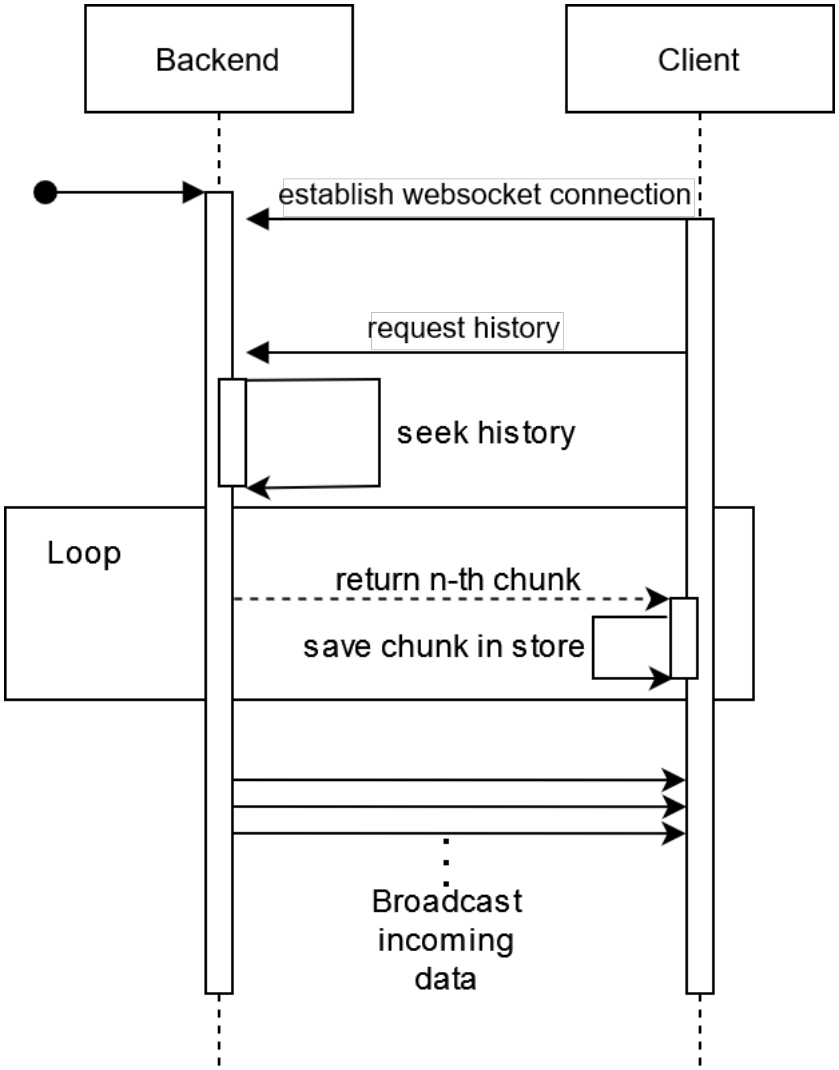

Figure 8: Backend procedure

As shown in Figure 8 the backend will execute the following procedure:

- Client connects to backend via web socket and a connection will be established.
- Client asks the backend `request_history` if there are any previous records.
- Server starts searching through the logging folder and will read every file ending with *.csv line by line.
- Server then sends the array with the recent history in chunks with a small delay to the client. The size of the history is always

part of each chunk, such that the client is able to verify that every record is transmitted.

- If a new packet arrives over the HTTP interface, then it will be instantly forwarded to every client.

Any further communication is currently limited to the broadcasting of new records from end devices registered by the application server. The real-time aspect is realized through the constant stream of messages over web sockets. For example, if a client has to reconnect to the backend, it will remain up to date because after each reconnect the history is transmitted again. The web socket client we use is `autobahn web socket` [3] and its asyncio implementation allows us to integrate it with the HTTP and logging component. To get the backend up and running it is necessary to edit the `config.py` file with the ID, name and location of the gateways. When a packet arrives, the backend appends the `gwcoord` and current timestamp to the JavaScript Object Notation (JSON) string. If there is a case where multiple backends have to be up and running and receive packets from the same Chirpstack instance, then the timestamps can be slightly different. But for our use case, it does not matter because we can identify each packet by its frame count. Something that could be changed is the way how the history is transmitted. Currently, it is based on order which files are read first and not by date. It might be useful to restore the history sorted by date starting with the most recent record back to the least recent one. We have made a few assumptions while developing the software components. For example, we assume that the backend has enough storage capabilities to save every incoming packet from every application server. Then we assume that the network capabilities of every client are sufficient to load the complete historical dataset as JSON strings and is capable of processing the data locally. Especially receiving and processing is something that can be optimized further in Section 4.6.

Extending Backend

The backend works with an `Asynchronous context manager` which allows for using multiple co-routines. Under the hood of asyncio, there is an event loop with a set of tasks. There is a smart execution, such that when a task is waiting for something, another task can proceed its execution. The constant switch between tasks leads to a very efficient sequential execution, which acts like a parallel execution. This is useful for IO intensive operations rather than compute intensive workloads. To extend the backend with additional tasks and functionality it is necessary to implement the following steps:

- In or outside the `async_backend` function it is necessary to initialize variables or objects.

- Create a task with `asyncio.create_task(coroutine)`.
- Add the co-routine to the task set of the event loop by calling `tasks.add(new_task)`
- Use `async def` keyword and use `await` keyword to return or execute blocking code.

4.4 FRONTEND

The frontend is our interactive real-time LoRa RF coverage mapping and analysis tool. Its core functionality is to assist while doing surveys and show real-time statistics about received packets, used gateways, active clients and an easy to use interactive map. According to the set requirements, the frontend should be able to provide a basic set of filter functions for simple evaluation.

Context

While researching we found basically no examples of an online web application which is able to plot RSSI and SNR in real-time onto a map including filter functions. There are multiple methods of solving this problem. Firstly, we could plot the incoming data with a library within python and push the static image to a web server, but then filtering is missing. The web application should be easy to use, provide a real benefit for the surveyor and be expandable. The frontend will not replace professional Geographic Information System (GIS) tools, nor is it an all-round solution for every mapping project. But the architecture and chosen approach seems very promising for future projects.

Implementation

We implemented the frontend in ReactJS. React is a JavaScript based library for building user interfaces. It has a component system and a new way of writing functional code as Hooks. Hooks solve the problem of reusing stateful logic between components and reflect more of what React really is. React also supports server-side and client-side rendering. This can be used to speed up the loading and processing of huge tables, but for our case the client side rendering is perfectly fine. We chose MapboxGL over Leaflet and HeatmapJS, because of its extensive set of ready to use visualizations, but also for the reason that we are still able to use our own tile server. This is not done yet, but a possible solution for larger projects with many more clients due to the fact that displaying a map into a browser is not trivial. A map consists of multiple tiles, those tiles are rendered JPGs out of compressed geographic data from OpenStreetMap. For this project we decided to stick to the token system by MapboxGL

and use their tile server. Now that the basics are explained, we can go through the lifecycle of an incoming packet.

- In the beginning the packet arrives at the web socket callback that decides if it is a broadcast or a chunk of history i.e. measurements in the past.
- Afterwards the action function will format the incoming data into the corresponding GeoJSON format and calls the dispatch to store.
- The chunk will be loaded into the reducer by choosing the right method through a simple enum to put the new record into the store.
- Every selector that selects the whole store, the object in which the data is nested, or the data object itself, will be notified of its change.
- `useEffect((),{object})` hooks in every component will be triggered and the component gets re-rendered.

We use a simple grid layout and theming from `material ui`. The most important components are the map, filter settings and tables. When a filter is applied, a copy of the full history will be made, filtered through every step and then saved as `visible_data`. Every record is saved into an array of features which represents a feature collection. Most of the UI Elements depend on this step to compute or display their values including the map.

Extending Frontend

The frontend is built around the heatmap component of `MapboxGL` and the react-redux store. For incoming data the packet lifecycle mentioned in Section 4.4 leads the following steps to extend the functionality of the frontend. Changes in the interface between frontend and backend on the side of the frontend has to be made starting in the `heatmapGL.tsx`. Three `dispatch()` functions are called to hand over the received data. Then inside the `src/actions/history.tsx` there are two functions (`appendBroadcast()` and `updateHistory()`) that should be modified to serve the updated interface. If there are major changes in the data structure then the `history` reducer should be adapted too. The reducer lying in `src/reducer/history.tsx` heavily uses JavaScript spread operator to ensure that the new object contains all previous entities. This is necessary because objects are immutable and have to be replaced. For better separation of context, multiple reducers could be added to the `rootReducer` inside `src/store.tsx`. New components should make use of the custom `useAppDispatch()`

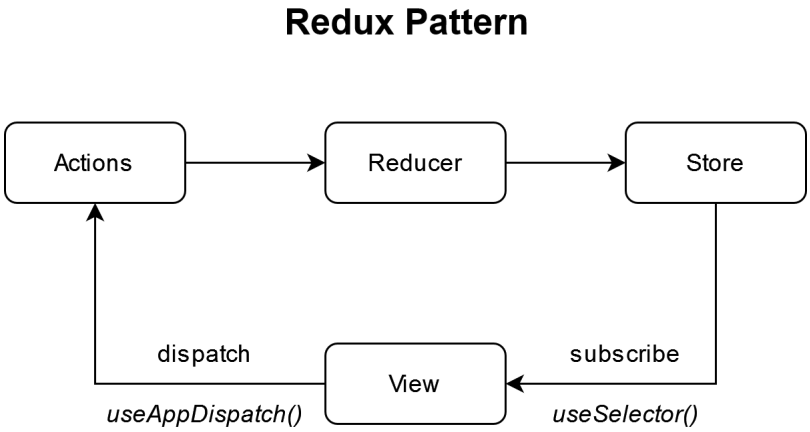

Figure 9: React redux pattern hinted with used functions

as `dispatch()` hook and `useSelector()` hook. Figure 9 roughly visualizes the components for the react-redux store. Filter functions can be added by extending the `applyCombinedFilter()` function inside `filterFunctions.tsx` with new `.filter((element) => return condition)` sections after the existing filters. Depending on the requirements it is advisable to reduce the amount of iterations made inside this function. Because in its current state, the function is called too often and the results are not cached. More to this issue in Section 4.6.

Challenges

A requirement in the development was that if one filter was changed, the whole application should react to it. Tables would change, numbers all over the app would change, and the map would look different and had to be re-rendered. This needs many global states which can be accessed from every component, but also should be mutable. The `react redux` store marks the perfect solution for that. The store holds the global state of the application including functions to access and change those values. The biggest difference to previous store types is that there is only one store i.e. root reducer which holds all states. This root reducer can be separated into multiple reducers for different parts of the application. The only way to trigger a change in the store is to call `dispatch(new_state_and_type)`. This dispatch is not called directly, rather it is hidden behind action functions that encapsulate the store from the consuming component. When a dispatch happens, then a `TYPE` is selected in a switch case containing the instructions to manipulate the store in the correct manner. This requires a lot of boilerplate code, but it enables us to create a truly interactive real-time web application.

4.5 ARCHITECTURAL DECISIONS

The first and foremost goal to achieve was a real-time aspect of our application. The consumer should not have to know or ask when new data is available. It should just receive, when available. This approach is centered around our backend and poses some drawbacks that are discussed later on. As already mentioned, when data is available it flows from the application server to our backend and will be forwarded to every connected client. This PUSH type of service can be discussed further by thinking about other methods of delivering data. Let us say our backend is polling data from a gateway and then push it over to the connected clients. The main goal would still be achieved, but the drawback is that our backend has to know each and every application server that is in service. Now we have to think about where the complexity and scalability should go. Our approach was to keep the complexity for the backend as simple as possible, basically using it as a forwarder for the data. It should not have any knowledge of existing application servers or gateways. Additionally, the web socket connection to the client allows for rather efficient data transfer. A connection has to established once and can be reused without renegotiating the TCP Handshake. The channel is bidirectional and the real use case of the impact is much more than we currently aimed for.

4.6 POSSIBILITIES AND DRAWBACKS

It is important to understand the possibilities and limitations of the whole system. In its current form it is limited to displaying SNR and RSSI value only and plotting these onto a map, but it can be extended even further. The basic data structure, structure of the store can easily be modified to fit other needs for example, measuring the air quality throughout the city. Moreover, most important for our use case, we can provide a back-channel for the node. Consider the following, we walk through the city doing the survey and suddenly the LoRa connection drops. As we measure with ABP this should not be a problem because the node will continue sending data without knowing if it arrived or not. With our solution it is possible that the node can be connected to an internet connected device like a smartphone, that is already connected to the web application to display the current measurements, to communicate with the application server that a packet was sent and should have been arrived already. We could call this a more intelligent approach of measuring LoRa RF coverage. The actual impact of being able to provide a back-channel to the application server over the backend is massive. Now we could basically automate the survey, let the backend and node decide on what frequency, spreading factor, TX power factor or antenna should be used next. This kind of concept could even be used outside the LoRaWAN context and more in a

general sensor and location based surveying context. Right now, the back-channel is not used except for the initial request for history data from the client side to the backend.

Drawbacks of the chosen architecture

In the current state, we hand over the main complexity in computation, data handling and processing over to the client. Though it is possible to enable server-side rendering, we decided against it. The reason behind this is, that clients are getting faster, processing power is increasing, and further development would increase the rendering efficiency or performance of our web application. The next section will go more into detail about this pending issue. Additionally, the push architecture does not really scale well with massive amounts of data and many users. Due to the fact that at the current state the whole history is retransmitted at every connection and there should be a more intelligent approach than the current one. Holding a local state that is bound to an expiration date and a custom protocol that communicates with the backend can be useful.

Risks and Technical Debt

Current issues are that the timestamp is added at the backend and therefore is not precise and should be extracted from the packet that arrives from the application server. The location of the gateway is also hard-coded and should be extracted out of the `location` field in the packet. React web applications have a tendency to re-render too often and this will result in performance issues down the line. Re-render happens when a state changes and the component where the state resides in, will be re-rendered. This can happen multiple times a second and depending on the size and complexity of the component this can take a lot of time and reduce the performance even further. As a good user experience counts a stutter free and seamless interaction experience with the web interface and this is not guaranteed. A source for slowdowns is the filter function that is recomputed every time a packet arrives and for each change in the filter settings. Executing the filter functions also iterates over the whole dataset. There are multiple methods of reducing the amount of re-computation and therefore re-rendering inside react. It is called `React.memo()` and caches already computed values if they did not change.

5 EVALUATION

In the following few sections, we will go through the measurements we made and analyze them. We deployed a gateway at three different locations around Darmstadt and measured the coverage of each one using SF7 and SF12 with two different node setups. We also changed the antenna of the Nucleo Node to identify whether the range increases or not. For that, we chose two different antenna setups namely the default antenna which we will call Antenna A and a 20.5 cm long antenna called Antenna B. For the visualization, we use screenshots from our web application which uses the MapBoxGL tool set [19]. Moreover, the size of a circle is defined by the RSSI value. A better/higher RSSI value equals a bigger circle whereas a worse/lower RSSI value results in a smaller circle. The color of each circle is defined by the SNR value, which slowly transitions from green to red. Green represents an excellent/high SNR value whereas red stands for a bad/low SNR value. There is also a legend in each presented Figure. Gateway icon taken from [28].

5.1 GATEWAY PANKRATIUSSTRASSE

The first gateway is deployed in the cybersecurity (CYSEC) building of TU Darmstadt, located in the Pankratiusstraße at the university campus downtown. The exact coordinates are 49.87812, 8.65705. The gateway is mounted on the third floor above the ground floor outside a window, so there is nothing in between that would disturb our measurements. It is facing south-east, away from the downtown area. We did several measurements around the downtown area with SF7 and SF12 and switched between our two node setups. In Figure 10 and Figure 11 you can see the coverage area of both nodes for SF7 and SF12 respectively. We first analyze the coverage for both spreading factors and then compare the two nodes. You will also find Table 1, where we compare the different distance zones in terms of SNR and RSSI, which are also visualized in Figure 14. Beginning with SF7, the coverage area on the west was pretty much bounded by the end of

Distance (m)	#packets	SNR (avg) (dBm)	SNR (min) (dBm)	SNR (max) (dBm)	RSSI (avg) (dBm)	RSSI (min) (dBm)	RSSI (max) (dBm)
200	1144	5.94	−16.0	11.8	−89.41	−129	−43
400	910	−3.79	−20.8	10.5	−112.11	−132	−87
600	359	−8.03	−20.0	7.2	−117.19	−133	−96
800	43	−10.65	−19.8	3.2	−120.49	−132	−102
1000	3	−12.27	−17.8	−7.0	−123.0	−130	−119

Table 1: SNR and RSSI comparison of for Pankratiusstr. gateway

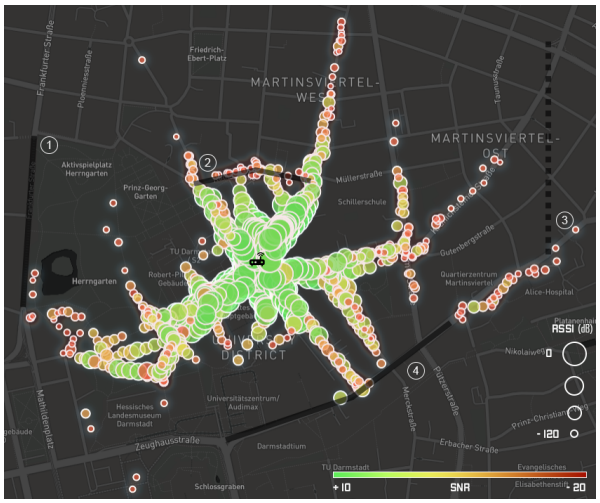

Figure 10: Pankratiusstr. coverage area for both nodes using SF7

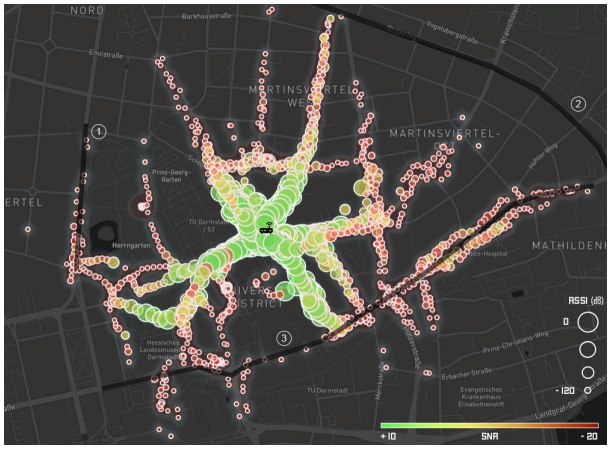

Figure 11: Pankratiusstr. coverage area for both nodes using SF12

the Herrngarten (marked with a 1 in Figure 10). Going over to the north, most of the packets did not make it further than straight behind the TU campus at Ruthsstraße (marked with a 2 in Figure 10). Only at the Pankratiusstr. the signal was still receivable almost up to the Rhönring. Looking at the east, the furthest packets made it as far as the Alice Hospital (marked with a 3 in Figure 10). The south was clearly bounded by the Darmstadtium at the Alexanderstr (marked with a 4 in Figure 10). We reached a maximum distance of 633 meters with an RSSI value of -117 dBm and SNR of -8 dBm for SF7. Considering SF7 only, approximately 95 % did not reach further than 450 meters around the gateway. Changing to SF12, increased our coverage area by some amount as seen in Figure 11. On the west side, packets were now still receivable a little further than the end of the Herrngarten (marked with a 1 in Figure 11). For the north, packets almost made it to the Rhönring at different directions behind the TU campus (marked with a 2 in Figure 11). The east now increased the range past the Alice

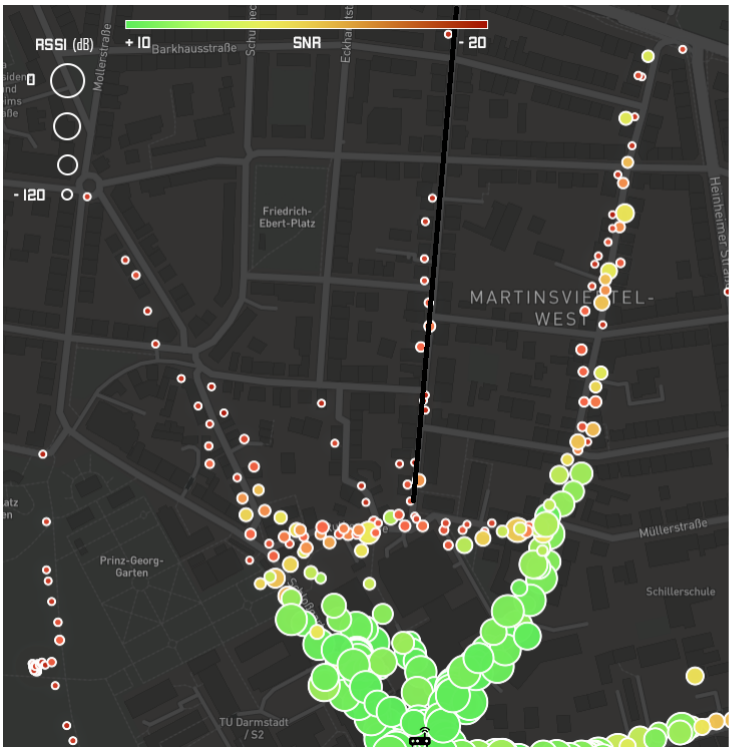

Figure 12: Coverage of Arheilgerstr. (marked in black) with Feather Node SF12

Hospital almost until the Spessartring (also marked with a 2 in Figure 11). The south was still mostly bounded by the Alexanderstr. (marked with a 3 in Figure 11) even though a bunch of packets made it a little further than that. Unfortunately, there was full packet loss at the Landgraf-Georg-Str., which is directly behind the Darmstadtium. For SF12 we, managed a maximum distance of 846 meters with an RSSI value of -130 dBm and SNR of -17.8 dBm. In contrast to SF7, where approximately 95 % of the packets did not reach further than 450 meters, SF12 only gave a slight increase of range, where now 95% of the packets lie within 550 meters around the gateway. We also switched between Antenna A and Antenna B on the Nucleo Node to see if a longer antenna increases the range. For that, we chose a specific location behind the CYSEC building up north, which already had bad coverage, namely the Arheilgerstr. While the Feather Node with SF12, using Antenna B, still had some packets coming through, the Nucleo node could not transfer any packets successfully with Antenna B using SF12. By switching to Antenna B, we managed to increase the signal range by a small amount, which is now more in line with the range of the Feather Node using Antenna B. An illustration of the Feather Node coverage using Antenna B with SF12 can be found in Figure 12 whereas the coverage of the Nucleo Node with Antenna B can be seen in Figure 13. We can summarize that a large portion of the TU campus downtown was covered by just this one gateway. Since

Figure 13: Coverage of Arheilgerstr. (marked in black) with Nucleo Node SF12 using Antenna B

the direction of the antenna is facing south-east, we reached further distances in that direction while the link quality behind the CYSEC building decreased faster. It might be possible to cover even more area by mounting the antenna higher or deploying another gateway.

5.2 GATEWAY MORNEWEGSTRASSE

The second gateway is deployed on the second floor above the ground floor, also in a university building, at the Mornewegstraße. The exact coordinates are 49.87252, 8.63576 and the gateway is mounted outside a window as well, thus preventing any disturbances of our measurements. Our antenna is facing east towards the downtown area of

Distance (m)	#packets	SNR (avg) (dBm)	SNR (min) (dBm)	SNR (max) (dBm)	RSSI (avg) (dBm)	RSSI (min) (dBm)	RSSI (max) (dBm)
300	904	4.01	−22.0	14.5	−101.32	−117	−57
600	986	−8.1	−23.5	12.2	−110.96	−117	−96
900	230	−14.15	−24.2	−1.2	−111.98	−117	−109
1200	67	−16.19	−23.8	−4.5	−111.85	−114	−109
1500	28	−16.05	−23.0	−8.2	−112.0	−114	−110
1800	8	−19.19	−24.0	−15.0	−110.62	−113	−109

Table 2: SNR and RSSI comparison for Mornewegstr. gateway

Figure 14: Different distance zones of Pankratiusstr. gateway in 200 meter steps

Darmstadt. We again did several measurements around the area with SF7 and SF12 using both node setups. The coverage area is visualized in Figure 15 for SF7 and Figure 16 for SF12. As seen in Figure 15, even with SF7 the coverage area was still quite large in comparison to the CYSEC gateway. You will also find Table 2, where we compare the different distance zones in terms of SNR and RSSI, which are also visualized in Figure 19. Starting with SF7 on the east side, we were pretty much bounded by the Kasinostraße and Neckarstraße (marked with a 3 in Figure 15). On the west side, it was almost impossible to reach behind the central station (marked with a 1 in Figure 15). Going south, the furthest range we were able to reach was around the University of Applied Sciences campus with some outliers going a little further. Finally, in the north the furthest packet was received at the Landwehrstraße (marked with a 2 in Figure 15). The longest distance covered by SF7 was 1.2 km away with an SNR of -9 dBm and RSSI of -111 dBm. Considering SF7 only, approximately 95 % did not reach further than 560 meters around the gateway. Now looking at the overall coverage area using SF12, starting on the east side, we were pretty much bounded by Kasinostraße and Neckarstraße again (marked with a 3 in Figure 16) but this time a bunch of packets made it a lot further than that, even though they seem to be outliers. Going over to the west side, we made a clear improvement to SF7 where we now reach behind the central station with a bunch of packets (marked with a 1 in Figure 16). Up north, we were mostly bounded by Landwehrstraße (marked with a 2 in Figure 16), but some packets made it slightly further. For

Figure 15: Mornewegstr. coverage area for both nodes using SF7

the south, packets were bounded by the Ahastraße (marked with a 4 in Figure 16). Overall, for the Mornewegstraße gateway our furthest link was 1.76 km away, close to the Darmstadtium building downtown. The SNR and RSSI value for that packet was -21.2 dBm and -110 dBm and again the coverage was the highest in the direction, where our antenna was mounted. In contrast to SF7, where approximately 95 % of the packets did not reach further than 560 meters, SF12 gave a greater increase of range, where now 95 % of the packets lie within 1110 meters around the gateway. Furthermore, we can quickly determine that the range of the Morneweg Gateway is higher compared to the CYSEC gateway. This might be due to the less built-up area compared to the downtown area where the CYSEC building is located. Comparing the two nodes we tested, the Feather Node reached a slightly higher range in comparison to the Nucleo Node. For our testing, we chose a particular walking route, what we found to be the highest range for the Feather Node. After walking that route with the Feather Node using Antenna B, we received more packets compared to the Nucleo Node using Antenna A, where almost no packet came through. A visualization of that route using the Feather Node and Antenna B with SF12 can be found in Figure 17. We again changed the antenna to Antenna B to be in line with the Feather Node. This seemed to improve the coverage and brought it back to almost the coverage level of the Feather node using Antenna B, as seen in Figure 18.

Figure 16: Mornewegstr. coverage area for both nodes using SF12

Distance (m)	#packets	SNR (avg) (dBm)	SNR (min) (dBm)	SNR (max) (dBm)	RSSI (avg) (dBm)	RSSI (min) (dBm)	RSSI (max) (dBm)
200	1547	9.23	−23.2	14.8	−58.22	−114	−19
400	1091	−5.64	−23.8	14.5	−98.77	−114	−59
600	73	−12.22	−23.0	3.8	−106.33	−115	−98
800	55	−14.49	−22.5	−3.8	−109.31	−116	−102
1000	29	−16.92	−23.8	−9.8	−110.52	−115	−101
1200	3	−21.93	−22.2	−21.8	−110.33	−115	−102
1400	7	−17.0	−22.2	−10.2	−108.14	−109	−108
1600	3	−19.97	−20.2	−19.5	−108.0	−108	−108

Table 3: SNR and RSSI comparison for Adelungstr. gateway

5.3 GATEWAY ADELUNGSTRASSE

The third gateway is deployed on the first floor above the ground floor in an apartment at the Adelungstraße, visualized in Figure 22. The exact coordinates are 49.8705, 8.6478. The gateway is mounted on top of a balcony border, so there are also no blockages in between and the antenna is facing east towards the downtown area of Darmstadt. Measurements were taken around the area with SF7 and SF12 using both nodes. The coverage area is visualized in Figure 20 for SF7 and Figure 21 for SF12. In comparison to the other gateways, the coverage area of the Adelungstr. gateway was the lowest. You will also find Table 3, where we compare the different distance zones in terms of SNR and RSSI, which are also visualized in Figure 23. For SF7, the coverage in the north mostly already ended at the Bleichstraße (marked with a 2 in Figure 20). Looking at the west, we were limited by the Hindenburgstr. (marked with a 1 in Figure 20) with some packets reaching further than that. The south was clearly shielded by

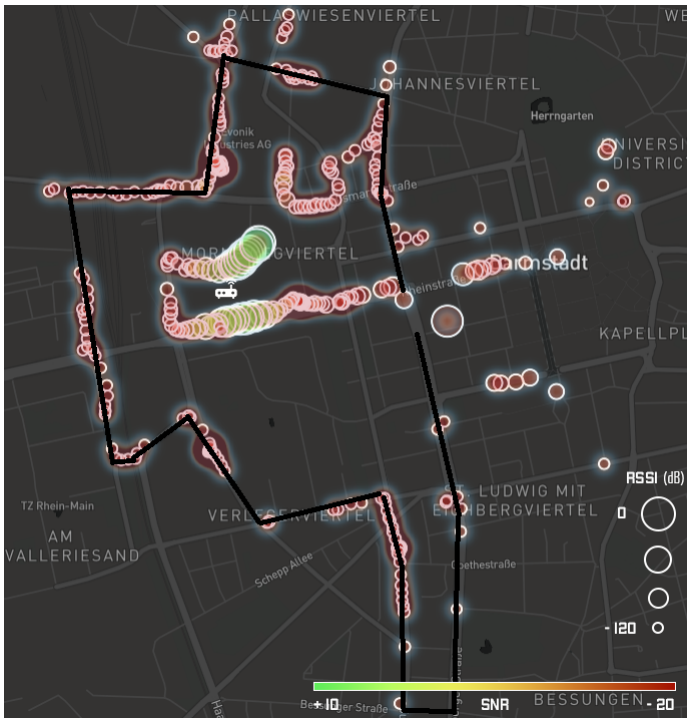

Figure 17: Mornewegstr. long distance route using Feather Node with SF12 and Antenna B

the Darmstadt state theater (marked with a 4 in Figure 20). Going over to the east, it was not possible to reach behind the Luisencenter (marked with a 3 in Figure 20). Only after switching the antenna of the Nucleo Node to Antenna B, we were able to get data packets from behind the center. Even though the antenna change increased the range, it did not grow by a lot, since the signal already was dark at the next street behind the Luisencenter, namely the Ernst-Ludwig-Str. The signal range on the east mostly ended at the Luisenplatz. The longest distance covered by SF7 was only 449.5 m away with an SNR of -8.8 dBm and RSSI of -104 dBm. Considering SF7 only, almost all packets, namely 95 % did not reach further than 330 meters around the gateway. Switching over to SF12, we get a slight improvement in the coverage area. For the north, west and south the coverage area mostly stayed the same compared to SF7. The east hit an astounding result with packets still being able to reach the gateway from as far as the Herrngarten and the TU campus downtown (marked with a 3 in Figure 21). The longest distance covered by SF12 was 1.52 km away with an SNR of -20.2 dBm and an RSSI -108 dBm. In contrast to SF7, where approximately 95 % of the packets did not reach further than 330 meters, SF12 gave a greater increase of range, where now 95 % of the packets lie within 720 meters around the gateway. Comparing the two nodes, the Feather Node using Antenna B had the higher range for both spreading factors, outperforming the Nucleo Node using Antenna A. This might be due to the bigger antenna size. Since

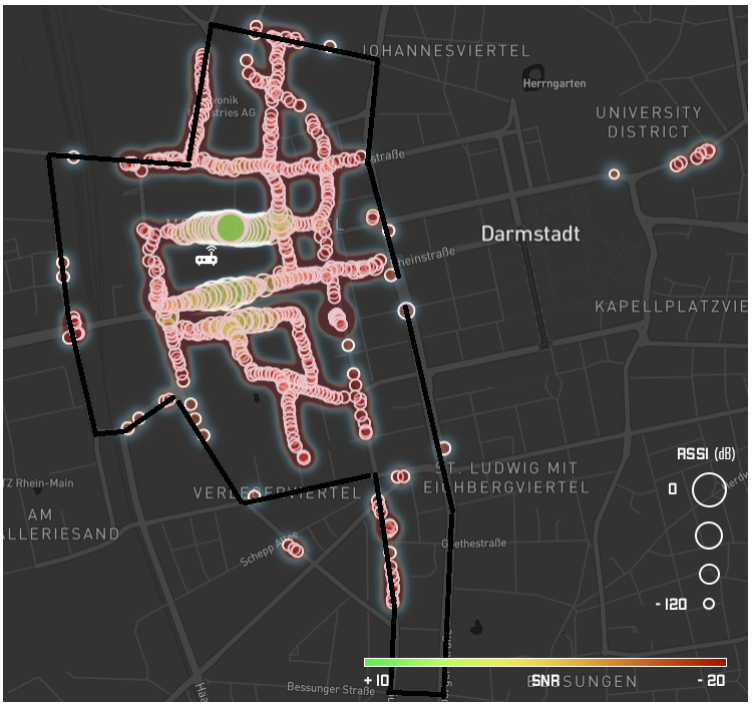

Figure 18: Mornewegstr. long distance route using Nucleo Node with SF12 and Antenna B

we only changed the antenna for specific locations we were not able to tell how Antenna B would perform for the whole area. But as we stated before, there was a clear increase in range with Antenna B for the Nucleo node when trying to reach from behind the Luisencenter. Taking a look back at our coverage maps in Figure 20 and Figure 21, there is a clear observation about the circle size which is defined by the RSSI value. For the Adelungstr. gateway there are hardly any small circles. This implies that for most of the packets, the SNR was simply too low for the package to be receivable.

Figure 19: Different distance zones of Mornewegstr. gateway in 300 meter steps

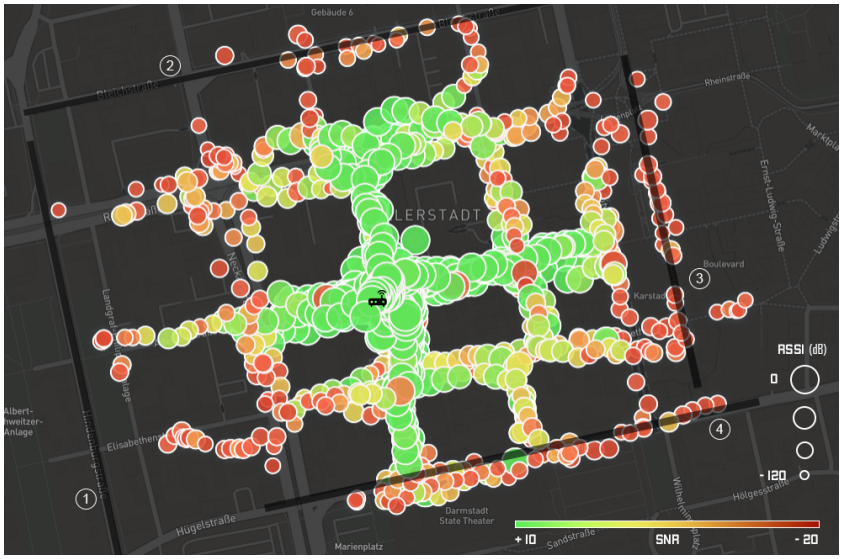

Figure 20: Adelungstr. coverage area for both nodes using SF7

Figure 21: Adelungstr. coverage area for both nodes using SF12

Figure 22: Adelungstr. gateway mounted on balcony

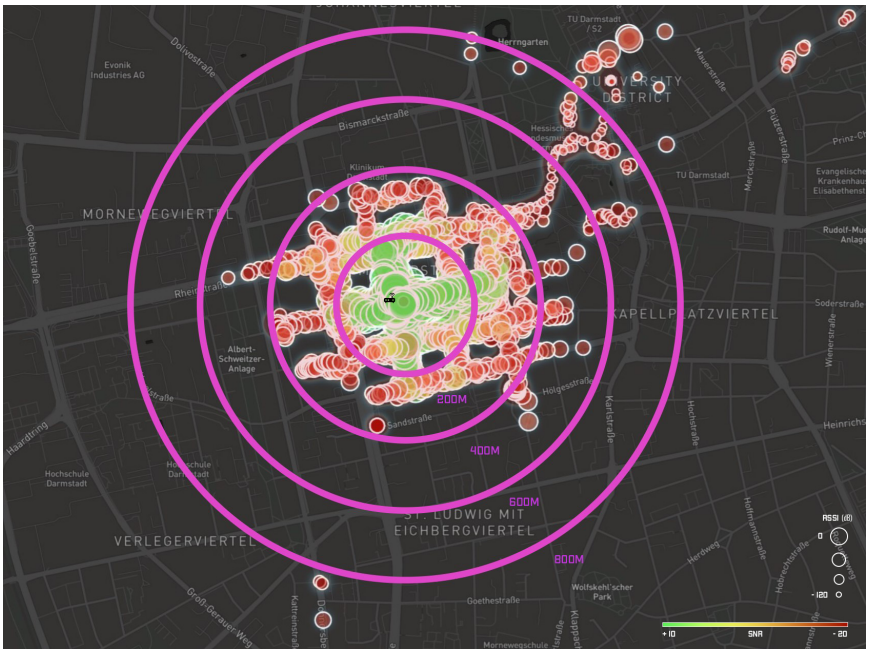

Figure 23: Different distance zones of Adelungstr. gateway in 200 meter steps

6 CONCLUSIONS

In this project, we build a web application for visualizing LoRa RF coverage in real-time and used it to create our own survey. We started by researching about LoRa technology, set up our gateways and nodes and received our first packets. We set up our own network and application server using the infrastructure of TU Darmstadt. Additionally, we build a backend server that logs and forwards incoming measurements to the clients. Using the application, we did a large scale field test. For that, three gateways were deployed around the city center of Darmstadt, and we used two different hardware setups to measure the coverage of each gateway using SF7 and SF12.

6.1 DISCUSSION

Building the real time application definitely helped us to make large scale field tests, as we were able to see ourselves on the map as a circle as soon as the packet was received. Thus, it was possible to clearly recognize areas with good or bad coverage in real time and where it might make sense to increase the distance even further. Therefore, improving the overall measurement efficiency. The useful filter functions also allowed us to keep a clear overview about the sheer mass of data. As without gateway or node filtering, it was not possible anymore to distinguish where measurements have already been made and by which node. This also marks the answer to our first research question from Section 1.2. The application is not really optimized for mobile end devices, but it was perfectly usable while on the go. Logging our received packets also made sure that there was no loss in data. Furthermore, our application is built on modern libraries and is highly extendible in terms of features. Going over to our second research question, our measurements took place at three different gateways with two different node setups. We visualized the coverage area for all three gateways in Chapter 5 and gave average, min, max values for the different distance zones of each gateway. The Mornewegstr. gateway for example had a higher range compare to the other gateways, but this may be due to the fact that there are less tall buildings in the surrounding area which would block the signal. The CYSEC gateway in contrast is in a highly urban area with tall buildings nearby. Thus, the overall coverage and link quality might be suffering. The Adelungstr. gateway also had a bunch of data points reached as far as 1 km and more but for the most part the coverage area was rather small. Because the area is also highly urbanized and the gateway was

not placed high enough, this might a reason for the lower coverage zone. Overall, the coverage of each gateway is highly unique and really depends on the surrounding area. Comparing the two nodes, we saw an advantage of the Feather Node, even though it was rather small. According to our data, there was an increase in range by switching the default Antenna A of the Nucleo Node to the longer Antenna B. This could be due to the bigger size of the antenna. It remains the fact that measuring radio coverage results in very different ranges throughout different cities, topographies and environments. Height difference in terrain can be exploited to cover larger areas by increasing the line of sight distance to the horizon, while reducing the amount of blockage to receivers on the ground. Now the question remains, which areas should be covered to which extent. For example, if we would place a gateway on the second floor in Frankfurt am Main downtown between the skyscrapers, the resulting coverage would be very different to the modest urban environment of the University district in Darmstadt. In our survey we used omnidirectional antennas only, that means that the radiation pattern is due to the dipole characteristic and antenna gain shaped like a donut. Due to that fact, we can observe that the signal reception is not bounded to one direction. It is visible that the area directly behind a building is much more attenuated than in front or near line of sight.

BIBLIOGRAPHY

[1] *ABP vs OTAA*. Accessed on 2022-08-10. URL: https://www.thethingsindustries.com/docs/devices/abp-vs-otaa/.

[2] Mohamed Aref and Axel Sikora. "Free space range measurements with Semtech LoRa™ technology." In: *2014 2nd international symposium on wireless systems within the conferences on intelligent data acquisition and advanced computing systems*. IEEE. 2014, pp. 19–23.

[3] *Autobahn Python*. Accessed on 2022-08-31. URL: https://github.com/crossbario/autobahn-python.

[4] Alexander Victor Tietgen Bardram, Mikkel Delbo Larsen, Krzysztof Mateusz Malarski, Martin Nordal Petersen, and Sarah Ruepp. "LoRaWan capacity simulation and field test in a harbour environment." In: *Third International Conference on Fog and Mobile Edge Computing, FMEC 2018, Barcelona, Spain, April 23-26, 2018*. IEEE, 2018, pp. 193–198. DOI: 10.1109/FMEC.2018.8364064. URL: https://doi.org/10.1109/FMEC.2018.8364064.

[5] *Chirpstack Docker Installation*. Accessed on 2022-08-30. URL: https://www.chirpstack.io/project/guides/docker-compose/.

[6] *Chirpstack LoRaWAN Network Server stack*. Accessed on 2022-06-16. URL: https://www.chirpstack.io/.

[7] *Estimating the service radius*. Accessed on 2022-05-04. 2016. URL: https://www.thethingsnetwork.org/community/oxford/post/estimating-the-service-radius.

[8] *Explanation of Adaptive Data Rate*. Accessed on 2022-08-10. URL: https://www.thethingsnetwork.org/docs/lorawan/adaptive-data-rate/.

[9] *Explanation of Spreading Factors*. Accessed on 2022-08-10. URL: https://www.thethingsnetwork.org/docs/lorawan/spreading-factors/.

[10] *Google Scholar Results for LoRaWAN between 2015 and 2018*. Accessed on 2022-08-30. 2022. URL: https://scholar.google.com/scholar?q=lorawan&hl=de&as_sdt=0%2C5&as_ylo=2015&as_yhi=2018.

[11] *Google Scholar Results for LoRaWAN between 2018 and 2022*. Accessed on 2022-08-30. 2022. URL: https://scholar.google.com/scholar?q=lorawan&hl=de&as_sdt=0%2C5&as_ylo=2018&as_yhi=2022.

[12] Bill Greiman. *SdFat Data Logger Example*. Accessed on 2022-08-31. URL: https://github.com/greiman/SdFat/blob/master/examples/examplesV1/dataLogger/dataLogger.ino.

[13] *Hardware Abstraction Layer for SX1302 by Semtech*. Accessed on 2022-06-16. URL: https://github.com/Lora-net/sx1302_hal.

[14] Frank Hessel. *LoRa GPS Tracker*. Accessed on 2022-08-27. URL: https://dev.seemoo.tu-darmstadt.de/LoRa/lora-gps-tracker.

[15] Frank Hessel. *LoRaWAN Topology Map from Mobile Networking course*. Accessed on 2022-08-10. URL: https://moodle.informatik.tu-darmstadt.de/pluginfile.php/204366/mod_resource/content/0/lecture_mobnet_c09m04_lorawan-basics.pdf.

[16] *Image Source of different Spreading Factors*. Accessed on 2022-08-10. URL: https://www.sghoslya.com/p/lora-is-chirp-spread-spectrum.html.

[17] Nikola Jovalekic, Vujo Drndarevic, Iain Darby, Marco Zennaro, Ermanno Pietrosemoli, and Fabio Ricciato. "LoRa transceiver with improved characteristics." In: *IEEE Wireless Communications Letters* 7.6 (2018), pp. 1058–1061.

[18] Marine Loriot, Ammar Aljer, and Isam Shahrour. "Analysis of the use of LoRaWan technology in a large-scale smart city demonstrator." In: *2017 Sensors Networks Smart and Emerging Technologies (SENSET)*. IEEE. 2017, pp. 1–4.

[19] *MapBoxGL Website*. Accessed on 2022-08-31. URL: https://github.com/mapbox/mapbox-gl-js.

[20] *OTAA Example Github*. Accessed on 2022-05-10. URL: https://github.com/matthijskooijman/arduino-lmic/tree/master/examples.

[21] Juha Petajajarvi, Konstantin Mikhaylov, Antti Roivainen, Tuomo Hanninen, and Marko Pettissalo. "On the coverage of LPWANs: range evaluation and channel attenuation model for LoRa technology." In: *2015 14th international conference on its telecommunications (itst)*. IEEE. 2015, pp. 55–59.

[22] *Platform IO*. Accessed on 2022-06-15. URL: https://platformio.org/.

[23] Irfan Fachrudin Priyanta, Frank Golatowski, Thorsten Schulz, and Dirk Timmermann. "Evaluation of LoRa technology for vehicle and asset tracking in smart harbors." In: *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*. Vol. 1. IEEE. 2019, pp. 4221–4228.

[24] Arrief Rahman and Muhammad Suryanegara. "The development of IoT LoRa: A performance evaluation on LoS and Non-LoS environment at 915 MHz ISM frequency." In: *2017 International Conference on Signals and Systems (ICSigSys)*. 2017, pp. 163–167. DOI: 10.1109/ICSIGSYS.2017.7967033.

[25] *RFC7946 GeoJSON Format*. Accessed on 2022-08-31. URL: https://www.rfc-editor.org/rfc/rfc7946.html.

[26] *RIOT OS*. Accessed on 2022-08-27. URL: https://github.com/RIOT-OS/RIOT.

[27] *RIOT OS website*. Accessed on 2022-08-31. URL: https://www.riot-os.org/.

[28] *Router Icon*. Accessed on 2022-08-31. URL: https://www.flaticon.com/free-icon/router_3336923?term=router&page=1&position=25&page=1&position=25&related_id=3336923&origin=search.

[29] *Semtech Acquisition of Cycleo*. Accessed on 2022-05-10. 2012. URL: https://www.businesswire.com/news/home/20120307006338/en/Semtech-Acquires-Wireless-Long-Range-IP-Provider-Cycleo.

[30] *STM32 Nucleo-64*. Accessed on 2022-06-15. URL: https://www.st.com/en/evaluation-tools/nucleo-l152re.html.

[31] *SX1276MB1MAS LoRa shield*. Accessed on 2022-06-15. URL: https://www.semtech.com/products/wireless-rf/lora-core/sx1276mb1mas.

[32] *TTN Mapper*. Accessed on 2022-08-31. URL: https://ttnmapper.org/.

[33] *WM1302 LoRaWAN Gateway Module*. Accessed on 2022-06-15. URL: https://www.seeedstudio.com/WM1302-LoRaWAN-Gateway-Module-SPI-EU868-p-4889.html.

[34] *WM1302 Pi Hat*. Accessed on 2022-06-15. URL: https://wiki.seeedstudio.com/WM1302_Pi_HAT/.

ACRONYMS

ABP	Authentication By Personalization
ADR	Adaptive Data Rate
GIS	Geographic Information System
GPIO	General Purpose Input Output
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
HTTP	Hypertext Transfer Protocol
IO	Input Output
JSON	JavaScript Object Notation
LoS	Line of Sight
MAC	Medium Access Control
MEO	Medium Earth Orbit
MQTT	Message Queuing Telemetry Transport
OTAA	Over The Air Authentication
RF	Radio Frequency
RSSI	Received Signal Strength Indication
SF	Spreading Factor
SNR	Signal-to-Noise Ratio
SPI	Serial Peripheral Interface
TX	Transmit Power