Zoltan Gere

Implementation of LoRaWAN from end-device to application

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor’s Thesis

11 November 2021

Abstract

Author:	Zoltan Gere
Title:	Implementation of LoRaWAN from end-device to application
Number of Pages:	29 pages + 1 appendices
Date:	11 November 2021

Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Smart Systems
Supervisors:	Joseph Hotchkiss, Project Engineer Keijo Länsikunnas, Senior Lecturer

LoRaWAN is long range, low power, wireless communication technology for Internet of Things devices. It has an increasing number of applications in various industries, such as manufacturing, smart cities, and agriculture.

In this project a complete LoRaWAN network is implemented. The resulting system gives an infrastructure for application developments and experiments. The network consists of three main components. The Chirpstack open-source LoRaWAN server stack is running on a personal computer. The LoRaWAN gateway is a separate network device. The reference end-device is built with widely available, inexpensive components. The project contains the schematics of the end-device for easy reproduction. The project also contains a software library implemented in MicroPython language. The library handles all the necessary LoRaWAN related communication.

Keywords: LoRaWAN, IoT, Network, Gateway, Raspberry Pi Pico

Contents

List of Abbreviations

ABP:	Activated By Personalisation.
ACL:	Access Control List.
ADR:	Adaptive Data Rate.
AES:	Advanced Encryption Standard.
CMAC:	Cipher-based Message Authentication Code.
CRC:	Cyclic Redundancy Check.
DevEUI:	Device Extended Unique Identifier.
FEC:	Forward Error Correction.
GUI:	Graphical User Interface.
IoT:	Internet of Things.
LoRa:	Long Range.
LoRaWAN:	Long Range Wide Area Network.
MAC:	Medium Access Control.
MIC:	Message Integrity Code.
MQTT:	Message Queuing Telemetry Transport.
OTAA:	Over-The-Air Activated.
SPI:	Serial Peripheral Interface.
SQL:	Structured Query Language.
TLS:	Transport Layer Security.
UDP:	User Datagram Protocol.

Glossary

docker:	Virtual machine environment, where the operating system and applications run in containers.
LORIX One:	LoRaWAN gateway device, developed and manufactured by Wifx Sàrl.

1 Introduction

Internet of Things (IoT) stand for small sized, networking, electronic devices with specific purpose, for example telemetry, surveillance or remote control. They are key components in smart technologies. Various technologies have been developed to create reliable communication between IoT devices. One of the most popular technologies is Long Range (LoRa). It is a wireless, secure, low energy, low data rate networking technology with long range coverage. The protocol can provide roaming and localization services. [1] [2]

IoT and smart technologies are rapidly developing Information Technology fields. Home automation has been increasing in popularity and smart home appliances can be found in growing numbers in households. Industrial IoT devices have been increasing in numbers as companies employ energy and resource saving smart technologies and factories implement more effective manufacturing processes. [3]

The fast growing IoT industry has an increasing demand for qualified technical personnel. This thesis has been conducted for Helsinki Metropolia University of Applied Sciences to support wireless communication technology education. The end-device software library has been developed to provide a foundation for students with beginner and intermediate knowledge.

This thesis explores the LoRa based communication and presents a reference Long Range Wide Area Network (LoRaWAN) network. The application server was configured with Message Queuing Telemetry Transport (MQTT) integration. The LoRaWAN server infrastructure was implemented in docker environment. A LoRaWAN gateway was configured and connected to the internal network. An end-device hardware was constructed and a LoRaWAN handler library was developed in MicroPython language for easy end-user development and application.

In the next chapter the project’s starting specification is explained. In chapter 3, the research and development methods are described. Afterwards the operational principles of the LoRa and other project related technologies are presented and the used hardware and software assets are further explained. In chapter 5, the resulting hardware system and the software library are presented.

2 Project Specifications

As mentioned in the previous chapter, the industry has an increasing demand for IoT technologies. To increase the students' professional experience the Helsinki Metropolia University of Applied Sciences creates new wireless communication based projects. The LoRa and LoRaWAN based communication network is an industry standard, and a widely used technology. It has a broad application area and it provides a good foundation for many interesting experiments. It is an ideal choice because the long distance coverage makes it available anywhere inside the building and the campus premises (compared to other indoor technologies, such as Wifi). The drawback is that it needs a supporting back-end network and centralized server hierarchy. The LoRaWAN server infrastructure should be properly installed and secured and it demands periodical administration and maintenance.

To mitigate these maintenance and security challenges the server is planned to run in a virtual environment. In docker the operating system and applications are running in separated, maintained containers. The server components run in individual containers and they are communicating in a virtual network. The server hierarchy has only the necessary access to the host computer and the physical network. Therefore the internal communication is secured against threat from the external network.

The containers are read-only file system images, the persistent configuration files are stored on the host machine. This approach makes the configuration and future migration tasks easier. To increase the resiliency of the server hierarchy all inter-server communication is performed through Transport Layer Security (TLS) connection.

The LORIX One LoRaWAN gateway provide the connection between the end-devices and the local network. A device with outdoor antenna was provided at the project launch meeting. Altough, as the device was used earlier, it was in an unknown state.

For cost-efficiency and component availability reasons the Raspberry Pi Pico board was selected. Combined with the Lambda62-8S radio module they provide a basis for the end-device. The MicroPython is simple, user-friendly language ideal choice for beginners. The MicroPython command line provides a convenient way to program the board and experiment with the language. The hardware components for the end-device were not available at the project launch meeting. The components were purchased at a later time.

In the next chapter the development process is explained and a short description of the available technical manuals is presented.

3 Material and Methods

3.1 Initial briefing

The product was developed for Helsinki Metropolia University of Applied Sciences. The development was started with a meeting, where the instructors outlined the project goal. During this meeting the requirements were discussed and the end-device hardware composition was decided upon. At the end of the meeting the currently available devices were provided. This hardware package consisted of the aforementioned LoRaWAN gateway and an STM32 LoRa discovery kit. The STM32 microcontroller was provided for development and testing purposes, but it is not part of the agreed project goal.

3.2 Researching and Planning

After the initial briefing a research on the possible technologies started. First research target was the LoRaWAN network, how does LoRa based communication work, what component are required and how do they interoperate. Out of the available LoRaWAN servers the Chirpstack server package was choosen. It is a user-friendly, open-source and modular server stack, available as downloadable docker image. It seemed to be a good fit for the project, and it provided a reliable foundation.

Next, the Chirpstack server components were studied. Its key features were examined, and the required dependencies gathered.

Finally, notes were taken and a draft network hierarchy drawn with all the necessary components included.

3.3 Network implementation

The installation process was performed in an agile manner, in other words first a minimal set up was created. When the services were running adequately, additional components were added.

The network service installation started with Chirpstack server's Docker Compose based set up. This install contains the necessary PostgreSQL, Redis and Mosquitto MQTT broker services. During the installation the component's configuration folder was moved from the default docker folder to a user specified folder. This allowed an easier project archiving later on. Next, the server component configurations were adjusted to the local network's requirements. To evaluate the outcome of the basic installation the server logging output was examined and affirmed that no severe errors were present.

In the next development cycle, the LORIX One LoRaWAN gateway was set up. As it was mentioned in the previous chapter, the gateway device was in an unknown state. Therefore a new operating system had to be installed. The Chirpstack server's Gateway OS software seemed to be a proper candidate. After the installation, the packet-forwarder was configured between the Gateway OS and Chirpstack Gateway Bridge. The gateway device addition-process was concluded by acknowledging succesful gateway status updates in the Chirpstack Network Server logging output.

The last cycle in server stack implementation secured the inter-server communication channels. The Chirpstack server suite supports this through the TLS protocol. The server software automatically enforce TLS based communication when certificates are generated and included in the configuration files. The certificates generation scripts were downloaded and the generator's configuration files were adjusted to the local network parameters. After the installation of the required softwares the certificates were generated. Subsequently the certifications were copied into the server configuration folders. Finally the certificates were enabled in the Chirpstack server configurations.

In relation to securing the servers the Mosquitto MQTT broker had to be secured. During the unsecured way of connection the user is authenticated with a username and password pair. These credentials may come from two different sources. The first group of credentials belong to the internal servers, that is the Chirpstack Network Server, Chirpstack Application Server and the Chirpstack Gateway Bridge. These are persistent user names. Their password and Access Control List (ACL) are verified from local files. The second group of credentials belong to the Chirpstack Application Server users. Their username and password are verified from the Structured Query Language (SQL) database. The Mosquitto MQTT broker does not support authentication from SQL database. To make this feature available an additional authentication plugin is required. For the project the Mosquitto Go Auth plugin was selected. As the Mosquitto Go Auth plugin is available with Mosquitto MQTT broker as a docker image, integration is performed by replacing the default docker image.

The network implementation process concluded with the testing of the whole system. During these tests, the STM32 LoRa discovery kit was programmed to connect to the network and periodically send messages. These tests highlighted errors in the network communication. Further investigation traced back the error to packet-forwarder related problems. To correct the problem the operating system on the LORIX One LoRaWAN gateway was replaced. The new operating system, the LORIX OS was a superior choice to the previous one. It is an intuitive, stable and secure software. [4] Following tests proved that the operating system replacement solved the network communication problems.

3.4 End-device Development

Research continued with the internals of LoRa radio communication. Early tests and experiments were conducted on the STM32 LoRa discovery kit. When the hardware components for the end-device had been received, the electronic development has started. Connector headers were soldered onto the microcontroller and the radio module. Then a prototype device was built on a solderless prototyping board. This way the pin connections between the controller and radio module could be reconfigured easily.

Next, the radio module driver developments was started. The Lambda62-8S radio module driver is based on an open-source, freely available software. Because there are differences in MicroPython implementations on different platforms, adjustments were added to fit the software to this particular prototype device. The driver is written exclusively in MicroPython. During the driver development, the Serial Peripheral Interface (SPI) communication was monitored with a logic analyzer.

The STM32 LoRa discovery kit was used as a debugging tool during driver development. It was programmed to send or receive LoRa frames. The radio waves could not be inspected in a similar way as the logic analyzer can monitor an electronic channel. When the prototype device and the STM32 device were directly communicating, mistakes could be logically deducted and the proper radio operation could be confirmed.

The LoRa software library contains utility classes for creating and processing LoRa packets and frames, processing payload, generating and verifying Cipher-based Message Authentication Code (CMAC). The library development was started from an open-source project. However, the available implementation is written in Python. There are differences between Python and MicroPython, most importantly in the available packages. The cryptography library is not available in MicroPython. Therefore an open-source Advanced Encryption Standard (AES) implementation was inserted into the project.

The LoRa software library provides only elemental LoRa frame generation. For the LoRaWAN functionality an additional module had to be developed. The design process had started with a thoroughful study of the LoRaWAN standard specification. Afterwards, a flowchart was planned, which follows the description in the specification. The flowchart is presented in Chapter 5. Proposed solution. Based on the design a LoRaWAN handler class was implemented. During the development the necessary changes were updated in the design.

3.5 Testing

The development concluded with a thorough testing phase. As the MicroPython is not interpreted language, the only way to fully test the LoRaWAN library is to run every possibly code path. On the other hand Chirpstack Network Server does not support simple way of packet forging. The test packet creation requires additional software technologies. Because the extent and the time requirement of the test packet creation, the detailed software testing is put out of the project's scope. Based on the field test it can be concluded that errors are unlikely.

The end-device and LoRaWAN communication is tested by performing measurements and transmitting the results. The received measurements were logged on the server computer. The measurements were running for more than 24 hours. The end-device was tested both on wall charger power source and on battery power source.

3.6 Software installation on target computer

When the project was finished, the whole system was demonstrated to the instructors. As a final step the server suite had to be installed on a Helsinki Metropolia University of Applied Sciences' computer. First the installation was simulated in a virtual environment. This had revealed problems, possibly bugs, in the latest software images. Some components stopped working properly. To solve the problem, the software components had been bound to specific versions and releases. This is further explained in the last section of Solution chapter.

4 Theoretical background

LoRaWAN based networks employ various technologies on multiple layers to create a reliable communication media. It consists of numerous software protocols and hardware components developed and maintained independently. The LoRa and LoRaWAN specification describes the requirements the final product must conform to, but the actual implementation is left to the manufacturer. In the next section the theoretical specification is presented, while the later sections of the chapter describe actual hardware and software elements.

4.1 LoRa and LoRaWAN

LoRa is a proprietary, royalty free, wireless communication technique. It is based on spread spectrum modulation to transmit elemental data fragments called chips. Each chip is transmitted over constantly increasing frequency. LoRa is the physical layer of LoRaWAN communication. During transmission the frequency continuously increases within a specified bandwidth around the central frequency. The central frequency in the EU region is from 863 MHz to 870 MHz, which is in the unlicensed Industrial, Scientific and Medical band. The available bandwith in the SX1262 modem is from 7.81 kHz to 500 kHz. [5]

Chips are transmitted multiple times to increase robustness and reliability. The Spreading Factor parameter means the number of chips that represent a single bit. The value of the Spreading Factor can be selected between 6 and 12. Higher number means more repetition, therefore the speed of data transmission decrease, but robustness improve. [5]

The frequency and the Spreading Factor together specify the speed of data transmission, called Data Rate. Accrding to the standard the minimal, mandatory set is from DR0 to DR5. The actual bitrate at DR0 is 250 bit/s, at DR5 is 5470 bit/s. [6]

LoRa modems use Forward Error Correction (FEC) algorithms to improve the link’s resilience to interferences. Before transmission additional bits are calculated and added to the data stream. The price of better noise immunity is longer transmission time. The LoRaWAN standard uses the Coding Rate 1, which means 1 additional bit for every 4 bits of data, therefore the data payloads are 25 precent longer. [5] [2]

Key features of LoRa is energy efficiency and long range. Battery operated devices using LoRa based communication can possibly operate for years. The typical communication distance is more than 10 kilometers. [3]

LoRaWAN specification is a networking protocol based on LoRa. It specifies the end-device operational principles and requirements to connect IoT devices to the Internet. It provides key features required for reliable IoT operation such as bi-directional communication and encrypted messages. Additional features are available such as multi-cast address groups, roaming (between networks) and geolocation. [1]

The LoRaWAN network topology is a star-of-stars layout. The network servers communicate with numerous end-devices through multiple gateways. LoRa gateways convert between wireless LoRa frames and wired User Datagram Protocol (UDP) packets transparently. The gateways communicate with end-devices directly on a bi-directional, half-duplex channel. [1]

There are 3 classes of end-device communication to satisfy different requirements in the wide range of applications. These classes provide options to balance between communication latency and energy-efficiency.

Class A - All is the default communication method. It must be implemented in all LoRaWAN devices. During class A communication transfer is always initiated by the IoT device, allowing it to follow an energy efficient scheme. The end-device can go into sleep mode between transmissions for an indefinite time. After transmission the device must open two receive windows to check for possible downlink frames. The application server must enqueue and send downlink frames

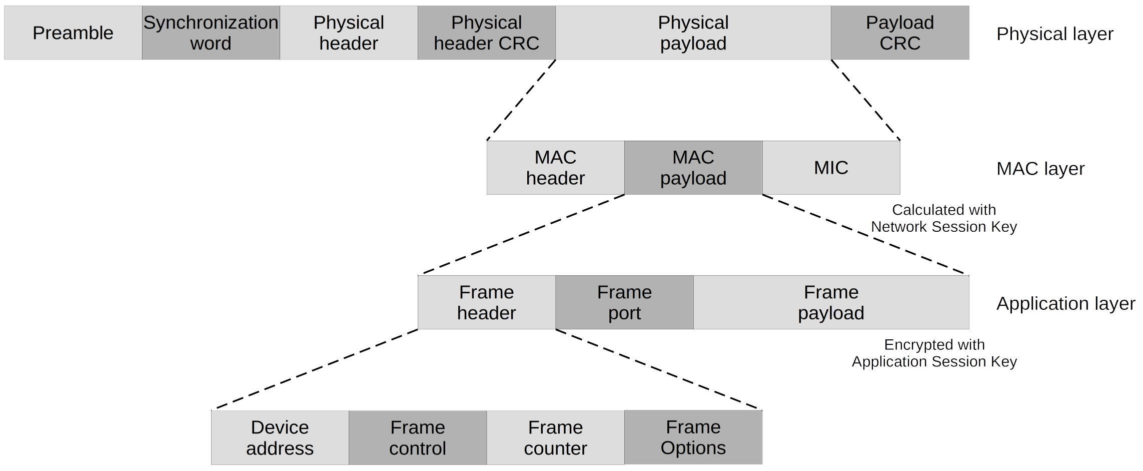


Figure 1: LoRa and LoRaWAN packet structure [5] [6] [7]

after reception.

The Class B - Beacon communication method enhances the end-device availability. The devices are synchorized to the network and they open downlink windows at scheduled times. This way the communication becomes well determined, with increased energy consumption although still viable for battery operated devices.

As an alternative for Class B, Class C - Continuously listening devices keep downlink windows open all times between transmissions. Therefore the network server can send downlink frames without latency. Because the higher power consumption involved this method is viable for devices with continuous power source. [1]

There are many parameters regarding the transmission. The frequency, and the bandwith were already mentioned in the LoRa and LoRaWAN section. The data rate which consists of the frequency and the Spreading Factor, and the Transmit Power defines the transmission further. As many transmitting device can quickly fill up the available band, these parameters are controlled by the LoRaWAN Network Server. Consequently the transmissions are distributed over the band. This process is called Adaptive Data Rate (ADR).

AES algorithms with 128-bit keys are used in two layers during LoRaWAN

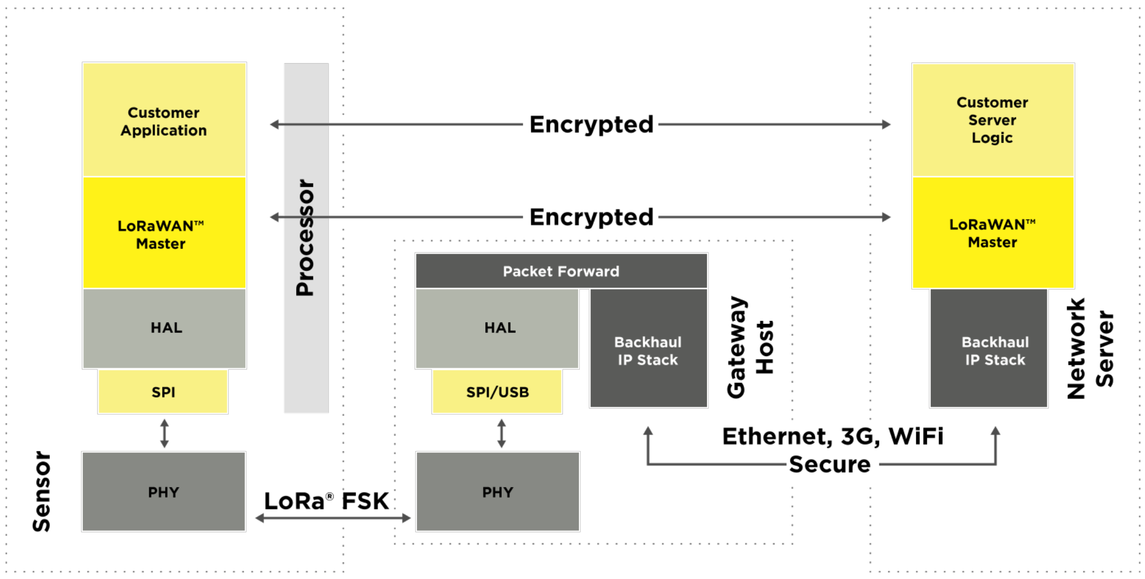


Figure 2: LoRaWAN protocol layout [1]

communication. The Network Session Key is used to authenticate and ensure integrity of messages between end-device and network server. The Application Session Key is used to encrypt payload exchanged with the application server. The two session keys are marked in Figure 1. In the Activated By Personalisation (ABP) method the keys are pre-set during production consequently they remain constant throughout the device lifetime. Alternatively, in the Over-The-Air Activated (OTAA) the two keys are generated and shared with a constant and pre-defined Application Key. [1] [7]

To increase resilience against replay attack and to eliminate message repetitions every frame has a counter number. Identical frames are sent with the same counter number. If the network server receives multiple frames from gateways sent by the same end-device, the frames with same counter number are filtered and forwarded only once. [7]

Figure 2 shows the application and network communication layers along with the underlying hardware layer. [1]

Figure 1 illustrates the LoRa packet layered structure with the fields used. Packets are created and processed in an embedded fashion, similar to the Open Systems Interconnection model. Data from application layer is encapsulated on the

Medium Access Control (MAC) layer into a frame which, in turn embedded in the packet on the physical layer. The LoRa modem handles the physical layer. The fields of this level are needed for transmission synchronization and error detection and correction. The LoRa gateway decides the forward direction based on the MAC header. Uplink packets are forwarded to the Network server on the wired network. Downlink packets transmitted to the end-devices on the radio. The Network Server verifies the Message Integrity Code (MIC) field. This field provides cryptography based message integrity check. The purpose of the field is to protect LoRa packets from tampering. Only frames with valid MIC are forwarded. The application layer is visible for both Network Server and Application Server depending on the content. Frames where port number is zero are intended for the Network Server and it contains LoRaWAN network management commands and answers. If the frame port number is a non-zero value the payload contains application data. These payloads are decrypted and processed by the Application Server. [5] [6] [7]

4.2 Chirpstack LoRaWAN Network Server Stack

Chirpstack project is an open-source network server suite. Together the software components provide complete LoRaWAN network server solution. As the components are created in an independent, modular fashion, they can be integrated into existing infrastructures. The Chirpstack server has a user-friendly graphical user interface for configuration, management and integration. [8] In the next subsections the server components and their internal communication are introduced.

4.2.1 LoRa Gateway

LoRa gateways listen on multiple channels and data rates simultaneously. Then, as mentioned in the previous section, they convert the received packets to wired protocol format and forward it toward the network server. The software handling the gateway’s communication on the wired network is called Packet Forwarder. The Semtech UDP Packet Forwarder and the Semtech Basic Station Packet

Forwarder are two commonly used implementations.

The LORIX One LoRa gateway is a commercially available, industrial grade device. It has its on Linux based operating system called Lorix OS. The system is configured and monitored through a Web User Interface. [9]

4.2.2 Chirpstack Gateway Bridge

The Chirpstack Gateway Bridge is running on a network server machine. It is the communication channel between the servers and the network. It transforms data between the Packet Forwarder’s format and the format used by the Chirpstack servers internally. The bridge is communicating with the Network server through the MQTT broker.

4.2.3 Chirpstack Network Server

The Chirpstack Network Server is an open-source implementation of the LoRaWAN Network Server specification. The software is performing LoRaWAN frame related tasks, such as de-duplication, authentication and scheduling of downlink frames. The Server is relaying the processed frames between the Gateway Bridge and the Application Server. The Network Server is capable to regulate end-device communication. With the MAC commands the network load can be distributed among frequencies and bandwidths. This way the LoRa network always performs optimally.[8]

4.2.4 Chirpstack Application Server

The Chirpstack Application Server is an open-source combined implementation of the LoRaWAN Application Server and the LoRaWAN Join Server specification. The server is responsible for LoRa device bookkeeping, presenting the device status, handling the end-device activation processes and performing application payload encryption and forwarding. The end-devices and LoRa gateways are grouped under services and application profiles. Information access is restricted on organization and user basis. The Application Server provides a web-based User Interface, where profiles and devices are managed, the integration of external services can be performed and access control tokens may be generated. Moreover, the Application Server periodically checks and records gateway and end-device availability and activity, thus providing a performance history. For developers it allows LoRa frame inspection for debugging purposes. [8] In the next section, the MQTT protocol is presented, which provides the internal communication channel for Chirpstack Network Server Stack.

4.3 Message Queuing Telemetry Transport

The MQTT is a light-weight, simple, open, publish-subscribe messaging transport protocol with strong security. It is ideal to use in environment with limited resources such as IoT and Wireless Sensor Network. The protocol typically used over TCP/IP protocol. The MQTT protocol is used as application layer protocol. The publish/subscribe message pattern provides reliable one-to-many message distribution, even when the devices and applications decouple frequently. Another characteristic feature of MQTT is the transport being independent from the payload content. This means that any information can be sent in any format, it will not have effect on the delivery. The quality of service describes three different options. In case of ”At most once” the message is sent only once and messages are allowed to be lost. In ”At least once” scenario the message deliveries are guaranteed, but they may arrive multiple times. Finally, in ”Exactly once” case the messages are assured to be delivered exactly once. This order of quality marks the necessary effort from simple to complex. [10]

4.4 Transport Layer Security

The TLS protocol allows applications communicating over networks or the Internet in a secure way to prevent eavesdropping, tampering and message forgery. The protocol achieve this by creating a secure channel between the two communicating parties. The underlying transport must be a reliable, in-order, bi-directional connection. The secure channel provides authentication, confidentiality and integrity to the delivered messages. The authentication secures that the communication is not diverted and messages are delivered to the intended recipient. The server is always authenticated, while the client authentication is optional. Confidentiality means that, the message can be read only by the intended recipient. Integrity ensures the message cannot be altered by attackers without the recipient noticing it. According to the specification these properties hold even if the attacker has complete control of the communication medium. [11]

4.5 End-devices

An end-device in the LoRaWAN network is usually a battery operated microcontroller with scarce resources. They perform a simple task such as measurement or actuation. Applied in vast number they can control complex systems occupying large area.

4.5.1 Raspberry Pi Pico

Raspberry Pi Pico is a low cost and flexible development board. It has 26 general purpose Input/Output channel. Its power supply is flexible, it can be easily operated from micro-USB port, external power supply or batteries. The hardware platform is high quality cost-effective and available in large numbers. The software development environment is comprehensive, well documented. It has sufficient calculation capabilities and enough memory to reliably support the MicroPython environment and run the necessary drivers for the LoRa radio module. These properties make the board ideal for study purposes. [12]

4.5.2 LoRa Radio module

The Lambda62-8S LoRa radio module from RF Solutions is a cost effective radio module featuring the Semtech SX1262 chip modem. It is capable of ultra-long distance communication employing spread spectrum modulation based transmission and it has good interference immunity and minimal energy consumption. [13] The module support preamble detection, the packet engine has buffer for message up to 256 bytes with Cyclic Redundancy Check (CRC). It communicates on 868 MHz EU band. The modem offers a large amount of configuration parameters to meet the requirements of the applications based on the LoRaWAN standard. The radio module communicates with the microcontroller through SPI line. [5]

4.5.3 MicroPython

MicroPython is a Python 3 programming language implementation targeted to microcontrollers and constrained environments. It includes small portion of the Python standard library. Additionally it contains hardware specific modules for low level hardware access. The simplified, script-style approach makes the Python language ideal for learning purposes. [14]

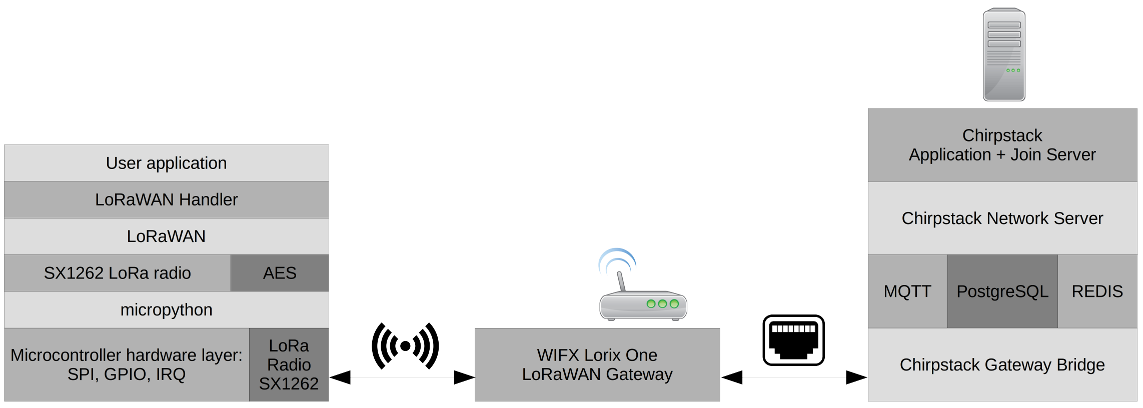

User application
LoRaWAN Handler
LoRaWAN
SX1262 LoRa radio
AES
micropython
Microcontroller hardware layer: SPI, GPIO, IRQ
LoRa Radio SX1262
WIFX Lorix One LoRaWAN Gateway
Chirpstack Application + Join Server
Chirpstack Network Server
MQTT
PostgreSQL
REDIS
Chirpstack Gateway Bridge

Figure 3: Component hierarchy

5 Proposed solution

5.1 LoRaWAN network hierarchy

Figure 3 illustrates the information flow path, from the user application to the Chirpstack Application Server. The message starts from the user application, encapsulated in a LoRaWAN frame, and the frame is encrypted. Afterwards the packet is sent through the SPI line to the LoRa radio, which, in turn transmits it. The LORIX One LoRaWAN gateway receives the packet and forwards it towards the Chirpstack Gateway Bridge. The Gateway Bridge converts the UDP packet to MQTT message. The Chirpstack Network Server receives the message and based on the packet's content either process it or forwards to the Chirpstack Application Server.

The LoRaWAN handler library has been written as part of the thesis work. Other software modules have come from open-source Github projects. They are modified and improved to fit the microcontroller and the project's objectives.

5.2 Chirpstack Server Suite

The Chirpstack Application Server has a Graphical User Interface (GUI) for configuration and device management. The interface is organized as user companies, network devices and applications. As shown in Figure 4 the operation

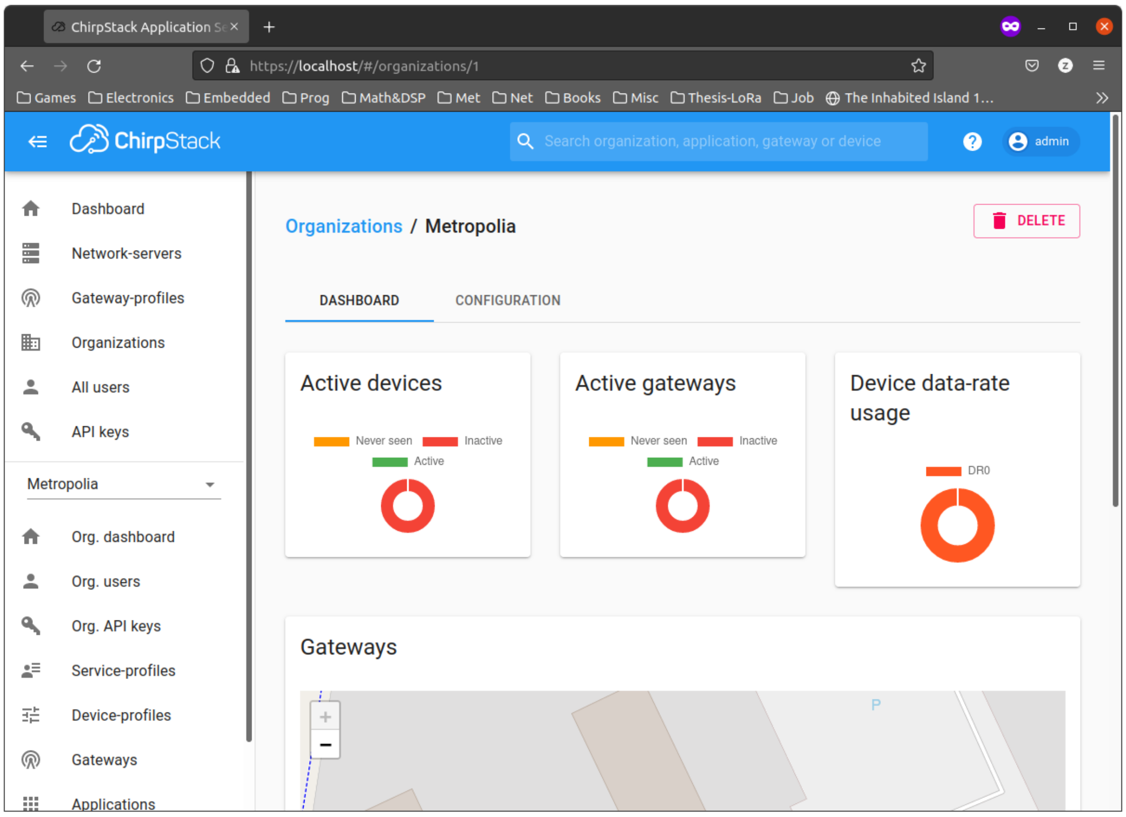
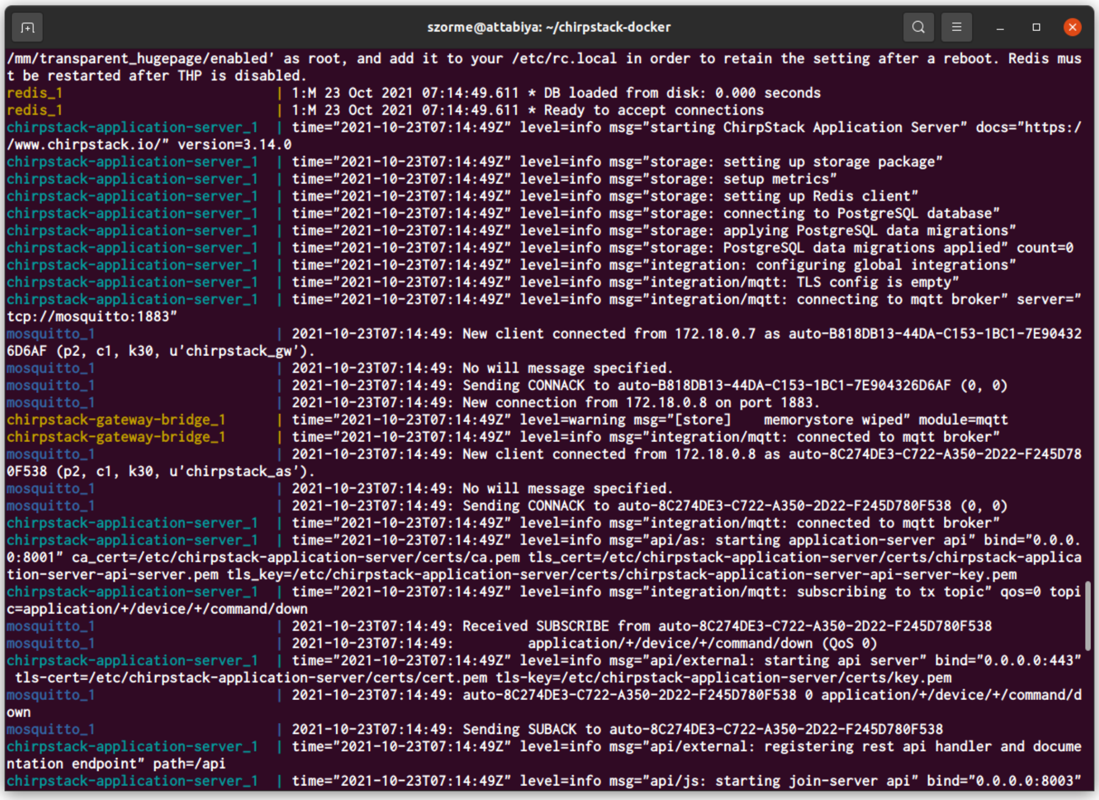


Figure 4: Chirpstack Application Server Graphical User Interface

of the devices can be monitored. New devices can be added and configured on the applications page.

Figure 5 shows the Chirpstack Server Suite running in terminal in separate docker containers. The server components provide log for monitoring the network traffic.

The MQTT broker is installed along with the LoRaWAN servers inside a Docker container. In case of the unsecured way of connection the user is authenticated with a username and password pair. The first group of credentials belong to the internal servers, that is the Chirpstack Network Server, Chirpstack Application Server and the Chirpstack Gateway Bridge. These are persistent user names. Their password and ACL is verified from local files. The second group of credentials belong to the Chirpstack Application Server users. Their username and password is verified from the SQL database. The application related connection is performed in secured way with certificates. These certificates are generated in Chirpstack Application Server's web-based user interface, under the applications/integrations page.



```
/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis mus
t be restarted after THP is disabled.
redis_1                          | 1:M 23 Oct 2021 07:14:49.611 * DB loaded from disk: 0.000 seconds
redis_1                          | 1:M 23 Oct 2021 07:14:49.611 * Ready to accept connections
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="starting ChirpStack Application Server" docs="https:/
/www.chirpstack.io/" version=3.14.0
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="storage: setting up storage package"
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="storage: setup metrics"
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="storage: setting up Redis client"
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="storage: connecting to PostgreSQL database"
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="storage: applying PostgreSQL data migrations"
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="storage: PostgreSQL data migrations applied" count=0
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="integration: configuring global integrations"
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="integration/mqtt: TLS config is empty"
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="integration/mqtt: connecting to mqtt broker" server="
tcp://mosquitto:1883"
mosquitto_1                      | 2021-10-23T07:14:49: New client connected from 172.18.0.7 as auto-B818DB13-44DA-C153-1BC1-7E90432
6D6AF (p2, c1, k30, u'chirpstack_gw').
mosquitto_1                      | 2021-10-23T07:14:49: No will message specified.
mosquitto_1                      | 2021-10-23T07:14:49: Sending CONNACK to auto-B818DB13-44DA-C153-1BC1-7E904326D6AF (0, 0)
mosquitto_1                      | 2021-10-23T07:14:49: New connection from 172.18.0.8 on port 1883.
chirpstack-gateway-bridge_1      | time="2021-10-23T07:14:49Z" level=warning msg="[store]    memorystore wiped" module=mqtt
chirpstack-gateway-bridge_1      | time="2021-10-23T07:14:49Z" level=info msg="integration/mqtt: connected to mqtt broker"
mosquitto_1                      | 2021-10-23T07:14:49: New client connected from 172.18.0.8 as auto-8C274DE3-C722-A350-2D22-F245D78
0F538 (p2, c1, k30, u'chirpstack_as').
mosquitto_1                      | 2021-10-23T07:14:49: No will message specified.
mosquitto_1                      | 2021-10-23T07:14:49: Sending CONNACK to auto-8C274DE3-C722-A350-2D22-F245D780F538 (0, 0)
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="integration/mqtt: connected to mqtt broker"
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="api/as: starting application-server api" bind="0.0.0.
0:8001" ca_cert=/etc/chirpstack-application-server/certs/ca.pem tls_cert=/etc/chirpstack-application-server/certs/chirpstack-applica
tion-server-api-server.pem tls_key=/etc/chirpstack-application-server/certs/chirpstack-application-server-api-server-key.pem
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="integration/mqtt: subscribing to tx topic" qos=0 topi
c=application/+/device/+/command/down
mosquitto_1                      | 2021-10-23T07:14:49: Received SUBSCRIBE from auto-8C274DE3-C722-A350-2D22-F245D780F538
mosquitto_1                      | 2021-10-23T07:14:49:         application/+/device/+/command/down (QoS 0)
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="api/external: starting api server" bind="0.0.0.0:443"
 tls-cert=/etc/chirpstack-application-server/certs/cert.pem tls-key=/etc/chirpstack-application-server/certs/key.pem
mosquitto_1                      | 2021-10-23T07:14:49: auto-8C274DE3-C722-A350-2D22-F245D780F538 0 application/+/device/+/command/d
own
mosquitto_1                      | 2021-10-23T07:14:49: Sending SUBACK to auto-8C274DE3-C722-A350-2D22-F245D780F538
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="api/external: registering rest api handler and docume
ntation endpoint" path=/api
chirpstack-application-server_1  | time="2021-10-23T07:14:49Z" level=info msg="api/js: starting join-server api" bind="0.0.0.0:8003"
```

Figure 5: Chirpstack Application Server Terminal Logging

The PostgreSQL database stores all data of the Application Server. It might be cleared by deleting the content of the database folder. During the next start the database server detects the empty folder and automatically initializes a new database from scripts. This way the Chirpstack Application Server starts anew. After the initialization the Chirpstack Network Server and the Gateway has to be reconnected, as described in the post-install instructions.

5.3 IoT end-device implementation

Figure 6 shows the end-device prototype used during development for testing the network communication. The device is created on a solderless prototyping board. This allowed quick reconfiguration and debugging during the early developments when the radio module was set up on the SPI line. The schematic of the prototype device can be found in Appendix 1.

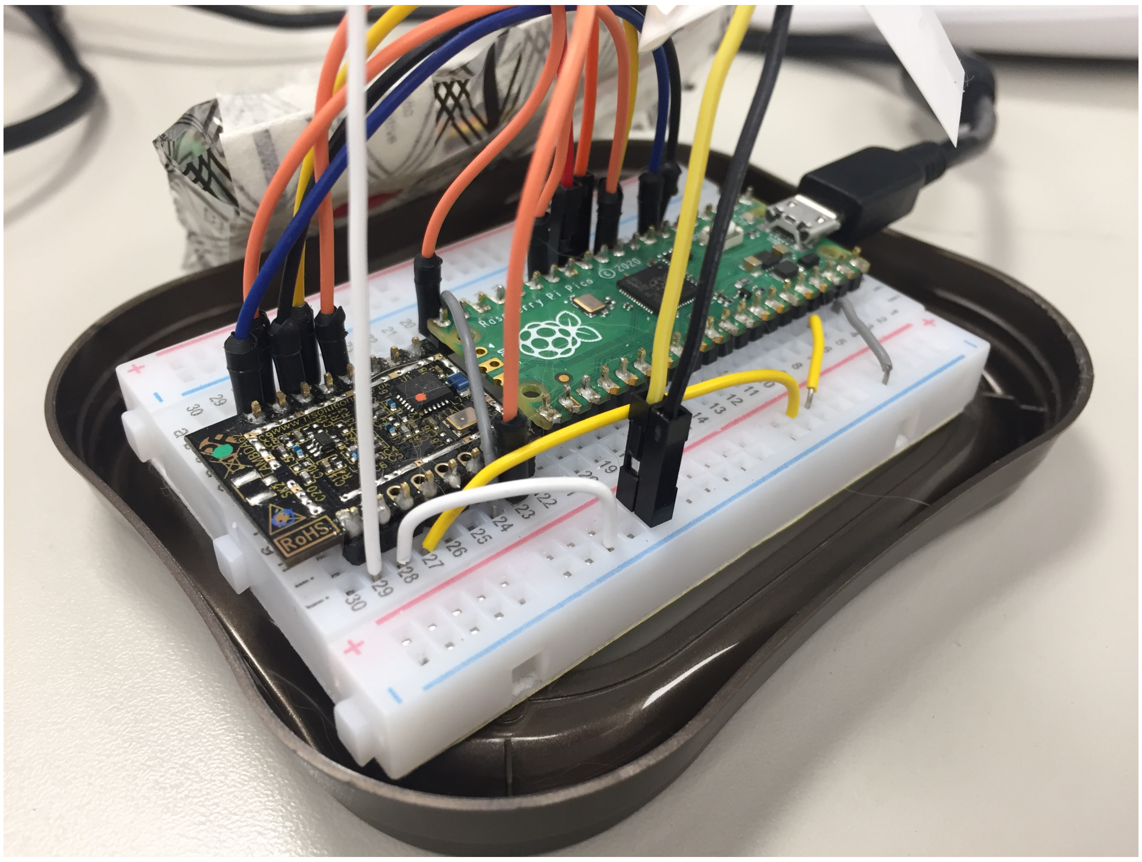


Figure 6: Prototype end-device

5.4 IoT end-device software library

The LoRaWAN handler class is built upon the LoRa library and the LoRa radio driver. It extends the communication libraries with the LoRaWAN functionalities, such as device activation, message retransmission, incremental frame counter handling, frequency hop, and MAC command processing. After instantiating the LoRaWAN handler class, the ’otaa’ method should be called. This initiates the device activation mechanism. The method sends the OTAA join request and waits for the answer. When the OTAA join accept has received, the method saves the device address, the network session key and the application session key. The application key and the Device Extended Unique Identifier (DevEUI) is saved in a Python class in a separate file. If no join accept received, the method continuously retries sending the join request message.

After the sucessful activation a message can be sent. The ’send’ method can construct both confirmed and unconfirmed uplink frames. If the success of the

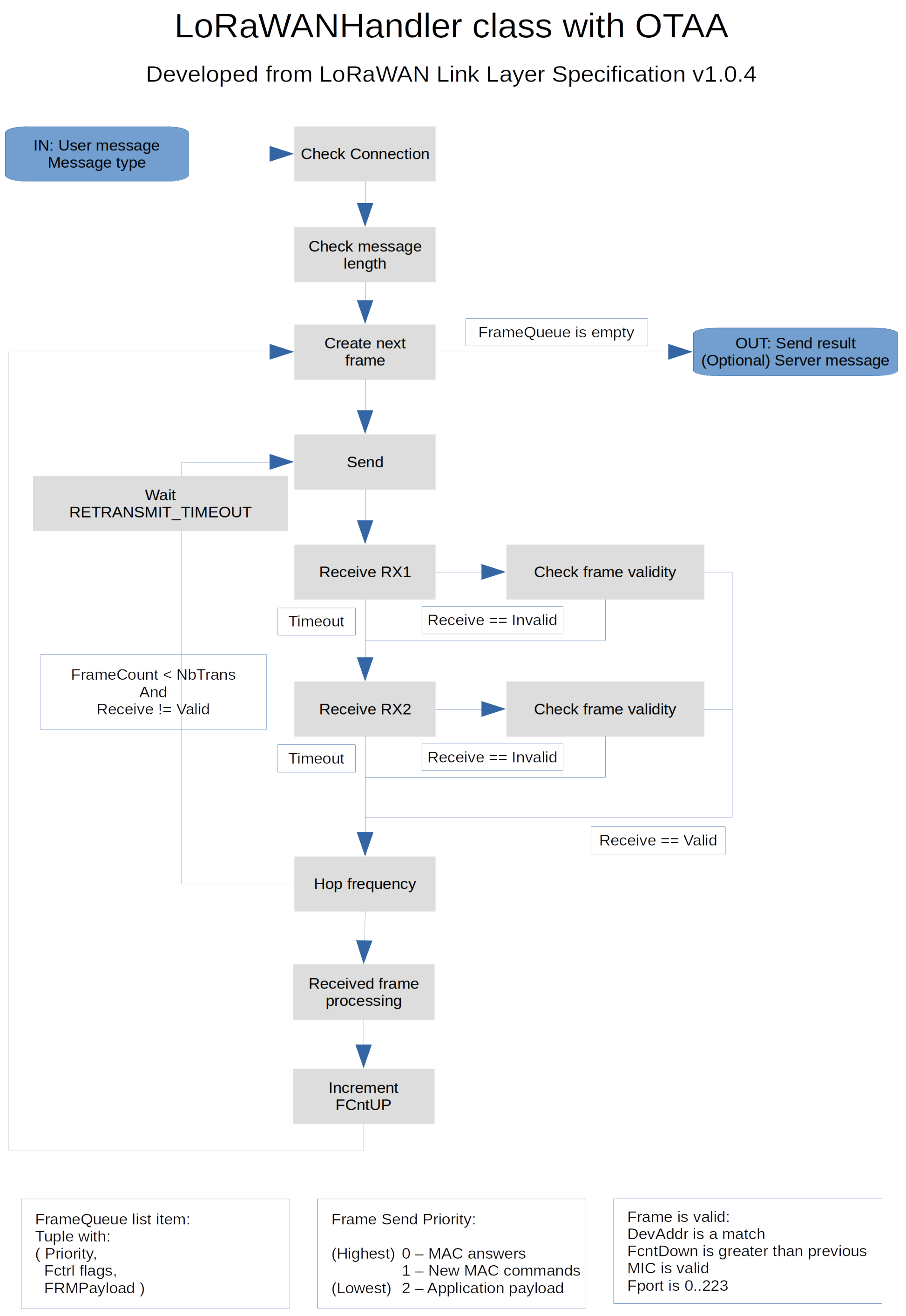


Figure 7: LoRaWAN Handler class send method’s flowchart

send can be deducted, the method returns ’True’ value, otherwise ’False’ value.

Figure 7 presents the operation mechanism of the send method. The send method implements the recommended message transmission mechanism set by the LoRaWAN standard. First the device activation and the validity of the message checked. Then a frame is created and repeatedly sent. The current repetition time specifies how many times the packet should be sent. The resending stops when a confirmation received, or a time-out occurs. This phase depends on the frame type (either a confirmed or an unconfirmed frame). Finally the frequency changed and the counters are incremented. In case of MAC command answer is in queue, waiting for send, but if no uplink frame is waiting for transmission, a new empty uplink frame is created. As a result the MAC commands are answered without delay and the requirements stated in the specification are fulfilled.

If any received downlink frame contains MAC commands, the commands are extracted and processed. The MAC command processing is implemented in the ’ProcessMACCommands’ sub-function. The possible commands and effect is illustrated in Figure 8. According to the LoRaWAN Link Layer specification v1.0.4 [7], the unknown commands are ignored.

5.5 Deviations from the specification

Altough the effort to create a full implementation of LoRaWAN, which is viable in production environment, available time frame is limited. As a consequence, there are a few missing features in the project. After the update on the Chirpstack Server modules the Secure connection between the Network Server and Join Server does not work over the TLS channel. This error has occured during installation onto the Metropolia provided computer, therefore it could not be further investigated. As a quick solution, the TLS is disabled.

Another error occured during the final install. The Mosquitto Go Auth plugin does not authenticate from files in the latest image. To fix this problem, the Mosquitto MQTT broker has fixed version and digest. During install the downloaded docker

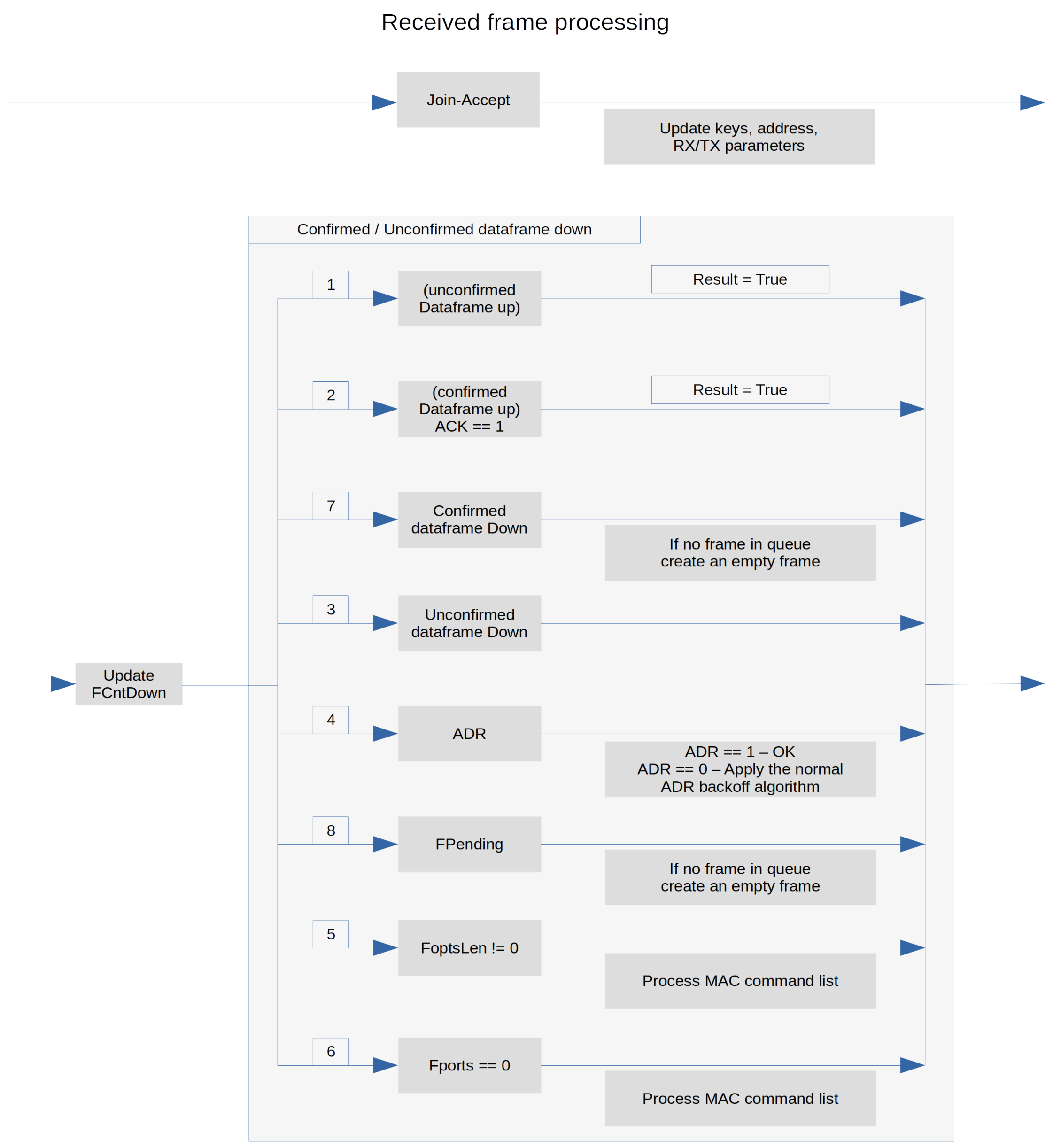

Received frame processing
Join-Accept
Update keys, address, RX/TX parameters
Confirmed / Unconfirmed dataframe down
1
(unconfirmed Dataframe up)
Result = True
2
(confirmed Dataframe up) ACK == 1
Result = True
7
Confirmed dataframe Down
If no frame in queue create an empty frame
3
Unconfirmed dataframe Down
Update FCntDown
4
ADR
ADR == 1 – OK
ADR == 0 – Apply the normal ADR backoff algorithm
8
FPending
If no frame in queue create an empty frame
5
FoptsLen != 0
Process MAC command list
6
Fports == 0
Process MAC command list

Figure 8: Received frame processing

image is replaced with a local copy.

The following are missing features, the network operates without them, but they could be implemented to further increase the usability of the network. Downlink communication with application is not possible. The end-device can send messages and process MAC commands. To make downlink message reception available, the Proprietary frame type has to be implemented. Consequently firmware update over-the-air does not work either.

No roaming is implemented. As only one gateway was provided, there was no possibility to experiment with roaming, altough possibly the required work hours exceeds the project's allocated time frame. Similarly no support for Class B and Class C end-devices has been implemented. This requires a separate packet receiving and processing mechanism.

From the aforementioned MQTT broker problem comes the question: Is Mosquitto Go Auth necessary? If the Go-Auth plugin is removed, there is no username based authentication. The Mosquitto MQTT broker can handle internal server connections from file. For all other connections the certification based connection is recommended. The certifications are provided on per application basis. As a result the username based authentication is removed. It depends on the usage methods whether this is an unused feature. This functionality was included to increase the number of possible use cases.

The tests are performed with one end-device. In production environment with hundreds of devices, unnoticed errors might show up stemming from saturated band. More end-device software library tests might be necessary in a network with a large number of end-devices.

6 Conclusions

The aim of the thesis was to develop a LoRaWAN network for experimenting purposes. The server stack is running in docker environment on a personal computer. The LoRaWAN gateway is connected to the network and it is reliably communicating with the server and the end-devices. The second aim was to create a prototype end-device and develop an examplary software to demonstrate the proper operation. The work is based on the available LoRaWAN specifications and hardware component datasheets.

As presented in the Solution chapter, the goals set for this study were achieved. The server stack is running and logging events. Through the user interface the Chirpstack server can be configured and monitored. The end-device was tested and it is working adequately. The schematics and the software library were provided for reproduction. Along with the server installation, additional documentation is provided to help later reinstall and configuration tasks. The resulting network provides a reliable foundation for further tests and developments.

There are still many unimplemented features left, as discussed in the specification section. Developing those features would increase the available tasks the IoT device can perform. However, the network is fully functional without those features.

References

1 Alliance®, LoRa. 2021. What is LoRaWAN®Specification. Online. <https://lora-alliance.org/about-lorawan/>. Visited on 04/16/2021.

2 Corporation, Semtech. 2021a. What is LoRa®? Online. <https://www.semtech.com/lora/what-is-lora/>. Visited on 04/16/2021.

3 — 2021b. Why LoRa®? Online. <https://www.semtech.com/lora/why-lora/>. Visited on 04/16/2021.

4 What is LORIX OS? 2021. Online. <https://iot.wifx.net/docs/lorix-os/latest/what-is-lorix-os>. Visited on 10/17/2021.

5 Corporation, Semtech. 2017. SX1261/2 Long Range, Low Power, sub-GHz RF Transceiver. Semtech Corporation. <https://semtech.my.salesforce.com/sfc/p/E0000000JelG/a/2R000000HT7B/4cQ1B3JG0iKRo9DGRkjVuxclfwB.3tfSUcGr.S_dPd4>. Visited on 10/14/2021.

6 Committee, LoRa Alliance Technical. 2020a. LoRaWAN Regional Parameters (RP002-1.0.2). LoRa Alliance. <https://lora-alliance.org/resource_hub/rp2-102-lorawan-regional-parameters/>. Visited on 10/14/2021.

7 — 2020b. LoRaWAN L2 1.0.4 Specification (TS001-1.0.4). LoRa Alliance. <https://lora-alliance.org/resource_hub/lorawan-104-specification-package/>. Visited on 10/14/2021.

8 Broocar, Orne. 2021. ChirpStack open-source LoRaWAN®Network Server. Online. <https://www.chirpstack.io>. Visited on 06/01/2021.

9 Discover the LORIX One 2021. Online. <https://www.lorixone.io>. Visited on 10/16/2021.

10 Andrew Banks Ed Briggs, Ken Borgendale & Gupta, Rahul. 2019. MQTT Version 5.0 Specification. Online. <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf>. Visited on 03/04/2021.

11 Rescorla, E. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. Online. <https://datatracker.ietf.org/doc/html/rfc8446>. Visited on 09/12/2021.

12 Ltd., Raspberry Pi (Trading). 2021. Raspberry Pi Pico Datasheet. An RP2040-based microcontroller board. Version 000dcb1-clean. <https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf>. Visited on 10/14/2021.

13 Solutions, RF. 2021. Lambda62. +22dBm LoRa 868/918MHz Transceiver. RF Solutions. <https://www.rfsolutions.co.uk/downloads/1623420782DS-LAMBDA62-6.pdf>. Visited on 10/14/2021.

14 George, Damien. 2021. MicroPython. <https://micropython.org>. Visited on 10/14/2021.

1 Schematic drawing of the end-device

The electronic connection of the end-device is presented in figure 9.

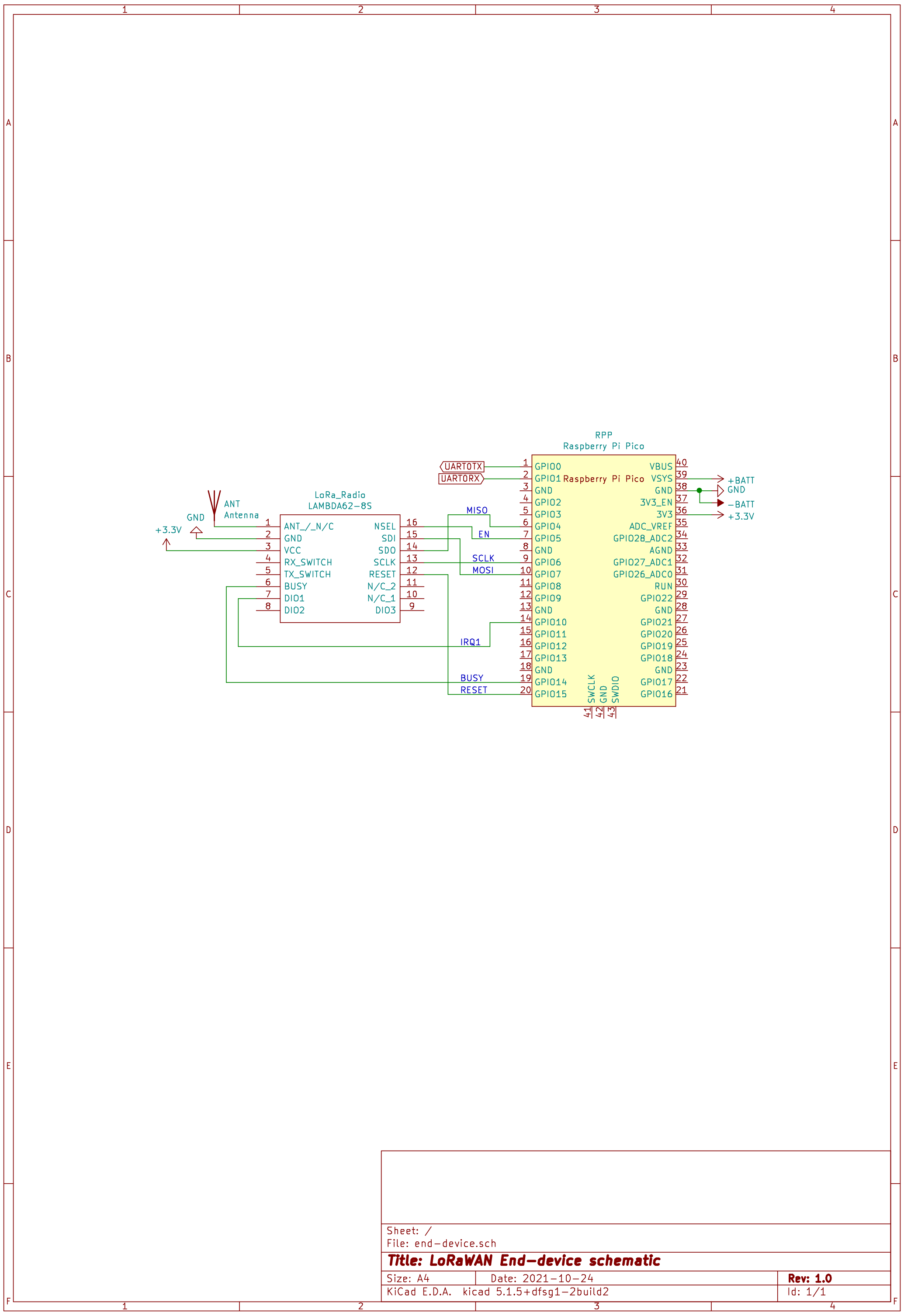


Figure 9: Schematic drawing of the end-device