LEM-KIT

LORA Environment Measurement KIT

FINAL REPORT

I.BA_IOT.H1801

David Schafer, Pascal Inäbnit, Stefan Küng, Roman Schraner

HSLU

1 Table of Contents

2 Abstract

The following documentation describes the project our Group created in the module “Internet of Things” in the Fall Semester 2018. In this Project, our group of four students did implement an Environment Measurement Kit, which gathers air quality, temperature, humidity, pressure and GPS data from different sensors. The Edge Device then sends these data via a Low Power Transmission Protocol (LoRa) to a gateway. The gateway redirects the data through WiFi to a Cloud Backend Service, which displays the data in the form of different graphs and other measuring displays.

3 Introduction

Our group did not have a specific problem, which we wanted to solve, to begin with. Our main motivation was to use the LORA technology, because we are fascinated about the manifold possibilities with this kind of communication protocol. So, we used a top-down approach to think of possible problems we could solve with LORA. After some discussions we talked about environment monitoring and how it is made today.

Most of todays established environment monitoring solutions are based on the fact, that you build something (a weather station or a building) with monitoring in mind. For example, in a new building, you would have dedicated space, power and network-access for monitoring components. However, these resources are often missing in existing buildings and other places. Adding such resources afterwards is very time-consuming and costly.

Another aspect is the use of environment monitoring in secluded areas like mountains or rural areas. In those territories, it is often not possible to add such resources at all. Our main goal in this project was to solve these kinds of problems.

LORA as the transmission protocol was predestined, because it was specifically built for low power applications. Basically, our project was the creation of an Ad-Hoc Environment Monitoring Kit, based on LORA, with low-power requirements, so that it could be operated with a battery.

4 Related Work

The basic idea of this project started with a YouTube-Video from Andreas Spiess called "LoRaWAN Demistifyed[1]". He gives a good introduction to the LoRa and LoRaWAN technology and its advantages. After this 'teaser', the group had to deep dive into the topic and gathered a lot of information. The challenge was to limit ourselves to the essential factors for our project and to sort out information which was either too detailed or irrelevant.

Dragino HAT Documentation[2]

Dragino's hardware documentation is rather insufficient. It helped us, as it provided fundamental information like a pinout-diagram, but it did not give good examples on how to create stable and well documented code. Despite our reservations to the HAT, we bought it because Andreas Spiess recommended it in one of his videos. In retrospect this was a bad idea, which we will discuss later in detail.

Heltec ESP32 LoRa WiFi Documentation[3]

Heltec provides their documentation about the Platform through GitHub. They also uploaded some useful example code to test basic functionalities.

Report about LoRa & LP-WAN from "Swiss Radio and Television"[4]

The report from SRF gave us a good impression about the possibilities of a Low Power Sensor Network. But the report did not help to get detailed information that would help with the concept and implementation.

Arduino LoRa Library Documentation[5]

We struggled to find a simple Library to use the functionality of the LoRa-Chipset. But after a couple of hours research, we found the Library written by Sandeep Mistry. This Library helped us a lot to make fast progress because it is much easier than other Libraries.

Azure Iot Edge LoRaWAN Starterkit[6]

After we have decided to proceed with LoRa and an Environment Monitoring Solution, we had to choose where the data should be stored. We found the Azure IoT LoRaWAN Starterkit by accident and thought it would be great to go with it. But because we overlooked one of the major requirements, we had to switch to another solution.

Semtec SX1276/SX1278 transceiver Datasheet[7]

The datasheet of the Semtec LoRa Transceiver helped us to understand some key functionalities about the transmission technology, the chip and everything involved around it. Due to the high level of detail it sometimes was hard to find the truly important information.

Adafruit BME280 Documentation[8]

The Adafruit documentation has proven itself very useful, because it contains wiring diagrams, the library documentation as well as code examples.

GPS NMEA data[9]

This documentation helped us to understand the syntax of the GPS NMEA messages that are used by the Adafruit GPS Sensor.

[1] https://www.youtube.com/watch?v=hMOwbNUpDQA&t=2s
[2] https://wiki.dragino.com/index.php?title=LoRa/GPS_HAT
[3] https://github.com/Heltec-Aaron-Lee/WiFi_Kit_series#instructions
[4] https://www.srf.ch/kultur/wissen/internet-der-dinge-vergessen-sie-den-vernetzten-kuehlschrank
[5] https://github.com/sandeepmistry/arduino-LoRa
[6] https://github.com/Azure/iotedge-LoRawan-starterkit
[7] https://cdn-shop.adafruit.com/product-files/3179/sx1276_77_78_79.pdf
[8] https://learn.adafruit.com/adafruit-bme280-humidity-barometric-pressure-temperature-sensor-breakout/arduino-test
[9] https://www.gpsinformation.org/dale/nmea.htm

5 System Design and Implementation

This chapter describes the final system design with a short description of the components.

5.1 System Overview

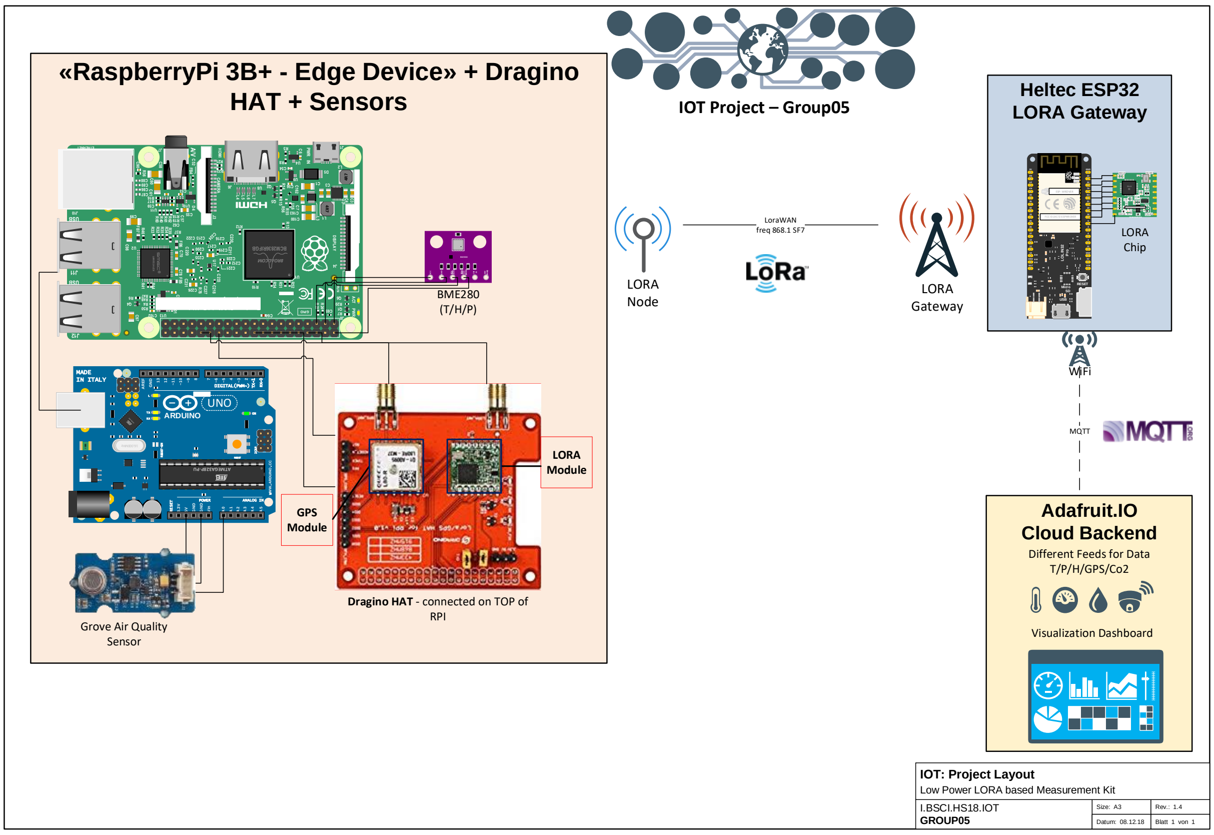


Figure 1: System Overview

The developed system is separated into three main components:

Edge Device: The Edge Device is a Raspberry Pi 3B+ with an attached Dragino LORA/GPS Hat. The Hat enables the LORA Communication with the Gateway. In order to do the environment sensing, the Edge Device was equipped with a BME280 (Temperature, Humidity and Pressure) sensor. Furthermore, there is a GPS Sensor built onto the Dragino HAT. This allows us to keep track of the accurate position of the Edge Device. Additionally, there is an Arduino with a Grove Air Quality Sensor attached.

Gateway: The Gateway is responsible for receiving messages from our Edge Devices. These messages are then processed and transmitted to the Adafruit.IO Cloud. To achieve the communication to the Cloud, the Gate-way uses the integrated WiFi module. A Heltec WiFi LoRa 32 with an inte-grated WiFi and LORA Chip makes it a perfect fit for the use as a Gateway.

Adafruit.IO: Receiving and visualizing the measured data is an important part of this project in order to generate value for the user. The Adafruit.IO Cloud provides an MQTT API. This API is used by our MQTT Client which publishes data to predefined Feeds. Received data will then be processed and visualized by Adafruit.IO. The dashboard is available at the following URL: https://io.adafruit.com/hslu_iot_hs18/dashboards/sample-dash-hs18

5.2 System Architecture

This chapter gives a more in dept insight about the used architecture.

5.2.1 Edge Device

Table 1: Edge Device Hardware Setup

Device	**Use**
Raspberry Pi 3B+	Computer, to control the Sensors and handle communication.
Dragino LORA/GPS Hat	Shield on top of the RPI. Enables LORA Communication and location sensing. Library (GPS): adafruit-circuitpython-gps
BME280	Temperature, Humidity and Pressure sensor. Library: adafruit-circuitpython-bme280
Arduino Uno R3	Gateway enabling communication between Air Quality Sensor and RPI
Grove Air Quality v1.3	Air Quality Sensor, reading an analog value.
Power Bank	Powers the Edge Device.

Most of the sensor libraries for the Raspberry Pi are written in Python. Therefore, the Python programming Language was used to realize the controlling of the sensors.

BME280 Setup: The BME280 is controlled over I^{2}C, simplifying the communication: After defining the appropriate I^{2}C address it is only a matter of calling the corresponding methods to read the values (temperature, humidity and pressure).

GPS Setup: According to the Dragino Wiki, it is required to connect Pins 22 and 24 in order to get the GPS module working with the Raspberry Pi. As soon as the Pins are connected, it is possible to talk to the module by UART. Appropriate configuration of the GPS Module is key: By sending PMTK messages to the GPS module, it can be adjusted what (and how frequent) data will be transmitted to the Raspberry Pi. Before the actual data (longitude, latitude and altitude) is readable, there must be a GPS fix. As soon as there is, the sensor writes the values to the UART interface.

LORA Setup: Because the Dragino Hat is still highly experimental, there was no Python library available to properly communicate with the Gateway. Therefore, a C program is used that can take a message (to be sent to the Gateway) as a parameter. The program configures the LORA module accordingly: The required frequency is configured, and the Parameters for the Spreading Factor are set.

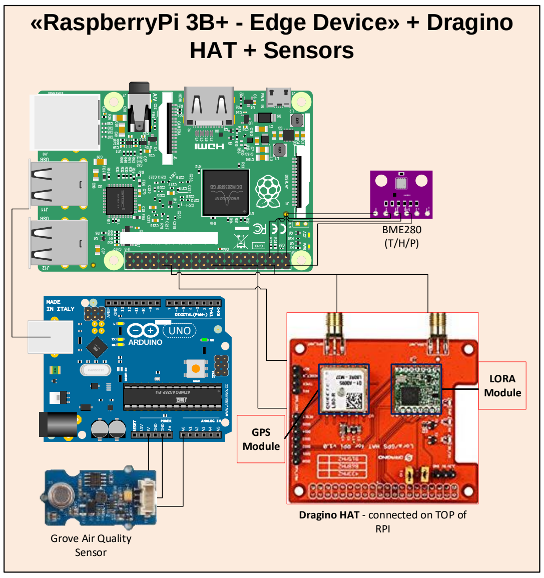

«RaspberryPi 3B+ - Edge Device» + Dragino HAT + Sensors
BME280 (T/H/P)
LORA Module
GPS Module
Dragino HAT - connected on TOP of RPI
Grove Air Quality Sensor

Figure 2: Edge Device Wiring

Arduino / Co2 Setup: The Air Quality Sensor returns a relative Air Quality Level based on the occurrence of different gases (Co, Alcohol, etc.). Because an analog input is required, an Arduino as a Gateway is used. This Gateway is connected to the Raspberry Pi over a Serial USB Connection.

Power Supply: A PowerBank delivered enough power to energize the Edge Device.

5.2.2 Gateway

Table 2: Gateway Hardware Setup

Device	Use
Heltec Wifi LORA 32 (v2)	Microcontroller, equipped with a LORA Transmitter, WiFi Module and an OLED Display.

As soon as the Device is started, the WiFi connection to a predefined Network will be established. The OLED Screen and the LORA Receiver will be initialized.

After the initialization has successfully ended, the Gateway is constantly listening for new LORA messages sent by the Edge Devices. As soon as an Edge Device initiates a connection, the Gateway parses all incoming packages until the sender is shutting down again. The fully received message will then be disassembled in order to determine which MQTT Feed (Adafruit.IO term for topic) to feed. After determining the correct Feed, the message is published over the MQTT API provided by the Adafruit.IO Cloud.

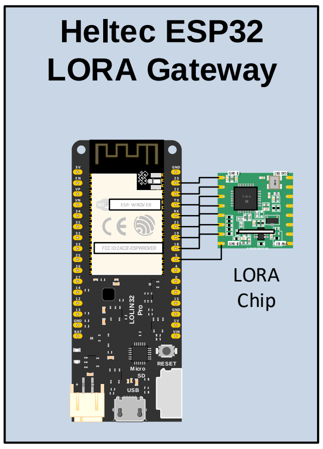


Figure 3: Gateway Wiring

OLED Setup: The Built-in OLED Display can be addressed by I^{2}C and the matching library. The specific I^{2}C ports must be connected according to the figure to the right.

LORA Setup: The LORA Chip only acts as a receiver; therefore, it is only necessary to set the Frequency (868.1 Mhz) and the Spreading Factor (to control the Bandwidth and Data Rate).

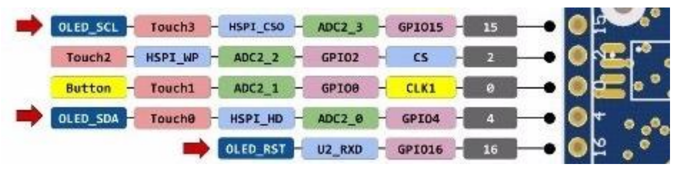


Figure 4: OLED Wiring

WiFi Setup: To gain access to a Wireless network, the WiFi Chip has to connect to a predefined network (requiring SSID and Password).

Adafruit.IO Server Access / MQTT: Because Adafruit.IO is a public Cloud Provider, a secured connection is necessary. In order to establish a connection, the following parameters are needed: Server, Port, Username and Session Key. With these parameters configured, a connection from the MQTT Client to the MQTT Server in the Cloud can be established. Furthermore, the Adafruit.IO is working with “Feeds”. These are MQTT Topics, that wait for incoming messages.

5.2.3 Adafruit.IO Cloud

Adafruit.IO provides different Visualization options that can be combined into Dashboards. Data is solely stored in the Cloud and there is no local Data-Storage necessary on our side.

The utilized free subscription model allows us to use it as follows:

- 10 Feeds may be defined
- 5 Dashboards
- Send 30 Measurements per Minute
- Store History Data for 30 Days

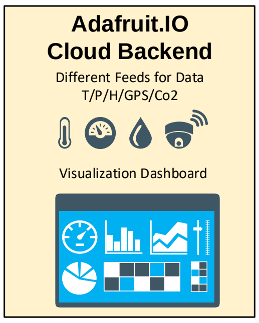


Figure 5: Visualization Layer

5.3 Software Architecture

Our solution consists mainly of the four modules “Master” (including: “Co2Sensor”, “BME280Wrapper” and “GPSWrapper”), “LoRa_sender”, “Read_co2” and “ESP32_Gateway_Master”. These modules run on different hardware platforms and are written in different languages. Communication between Raspberry and ESP32 is based on the LORA protocol. Between the Raspberry and Arduino, a Serial Interface is used.

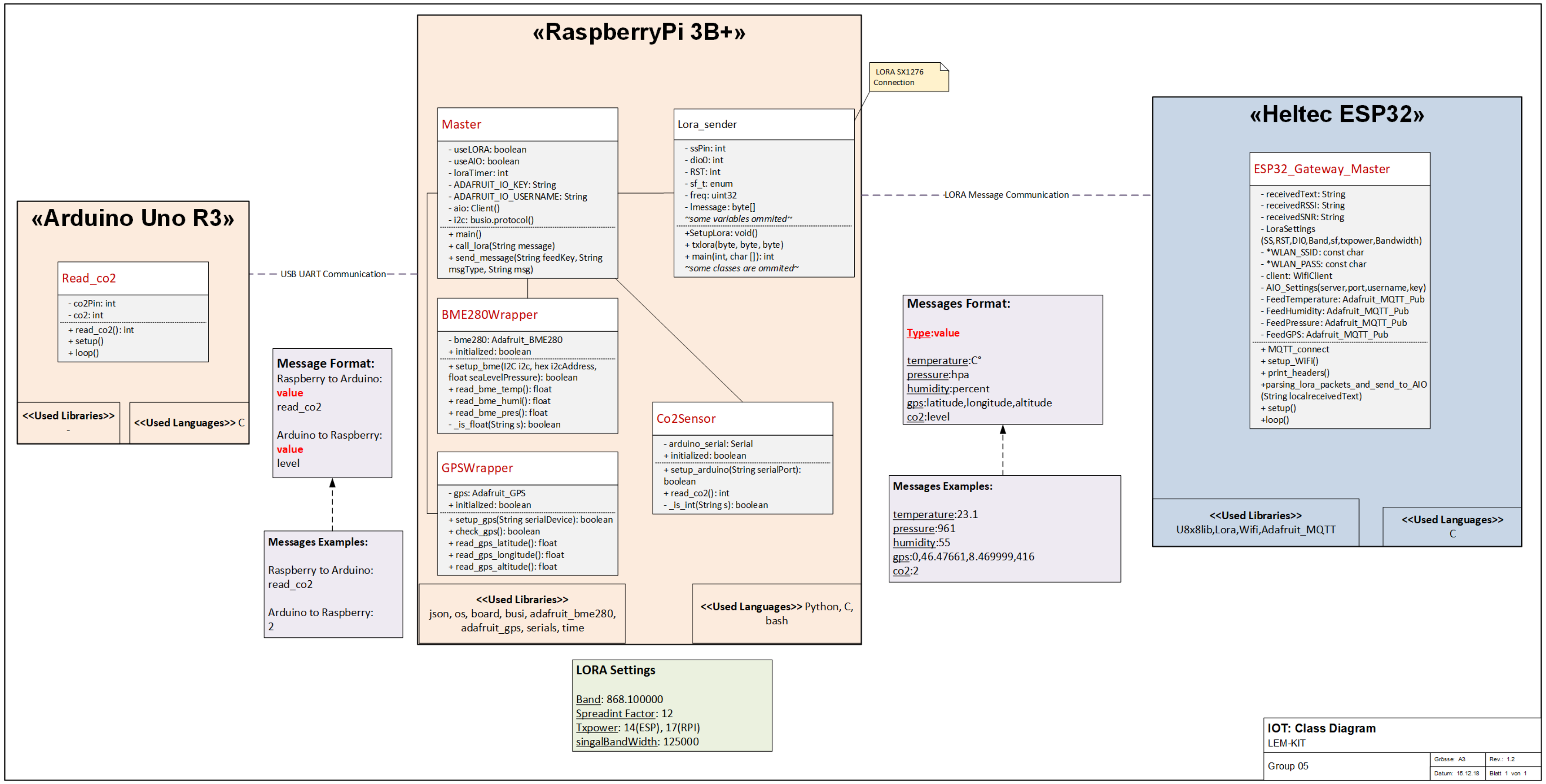


Figure 6: Class diagram

5.4 Interfaces

5.4.1 LORA Message Format

Messages Format:

Type:value

temperature:C°
pressure:hpa
humidity:percent
gps:latitude,longitude,altitude

The interchange of the LORA Messages takes place in a very simple format "type:value". The messages itself are basically encrypted by the LORA Syncword.

5.4.2 Serial Message Format

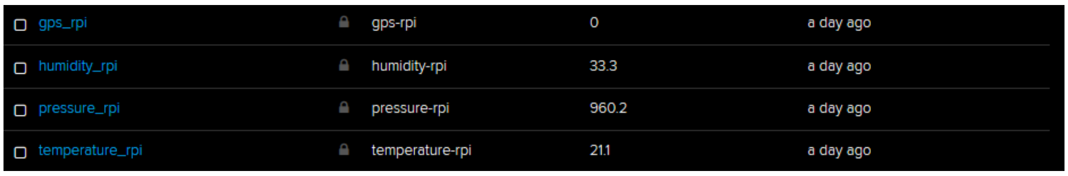
Message Format:
Raspberry to Arduino:
value
read_co2

Arduino to Raspberry:
value
level

The communication between the Raspberry and the Arduino is also based on very simple messages: The Raspberry only sends simple commands ("read_co2"). These commands are parsed by the Arduino and mapped to a specific method. As an answer, the Arduino returns an integer value. In this case it is the Air Quality Level.

5.4.3 Adafruit.IO Interface

The Feeds are the data sinks of Adafruit.IO. They are basically just containers for String values. These Feeds can be addressed via https or MQTT. We chose MQTT because of its low power aspects.



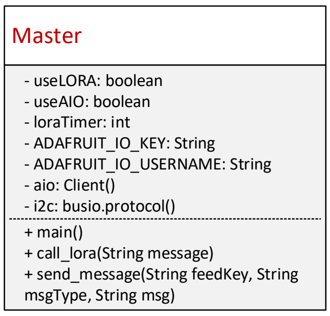
gps_rpi	gps-rpi	0	a day ago
humidity_rpi	humidity-rpi	33.3	a day ago
pressure_rpi	pressure-rpi	960.2	a day ago
temperature_rpi	temperature-rpi	21.1	a day ago

Figure 7: Adafruit.IO Feeds

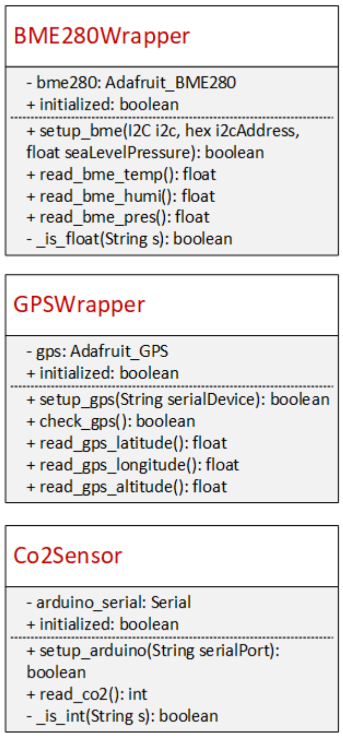
In the dashboard section, there are several predefined representations (gauge, graph, indicators, …). In the process of creating a representation, you have to map the corresponding feed and its values to different parameters or axles.

5.5 System Implementation / Functional Software Architecture

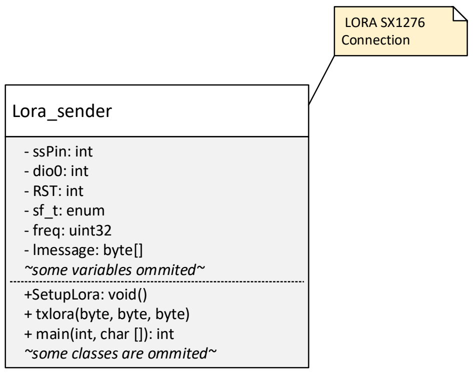
Master
The Master class is written in python and runs on the Raspberry Pi. It contains the different functions for reading out BME, CO2 and GPS Sensors (outsourced to classes), which we developed over this course. It also has a function to directly communicate with Adafruit.IO, in case there is WiFi available.


Master
- useLORA: boolean
- useAIO: boolean
- loraTimer: int
- ADAFRUIT_IO_KEY: String
- ADAFRUIT_IO_USERNAME: String
- aio: Client()
- i2c: busio.protocol()
+ main()
+ call_lora(String message)
+ send_message(String feedKey, String msgType, String msg)

Supporting Classes: BME280Wrapper, GPSWrapper, Co2Sensor
These classes are written in Python. They wrap around the Library implementation. This provides an even simpler way to read the values in the Master file.
The classes are responsible for reading the data from the according sensor and to communicate with the Arduino.

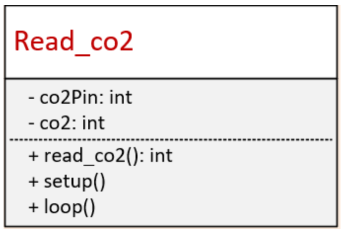

BME280Wrapper
- bme280: Adafruit_BME280
+ initialized: boolean
+ setup_bme(I2C i2c, hex i2cAddress, float seaLevelPressure): boolean
+ read_bme_temp(): float
+ read_bme_humi(): float
+ read_bme_pres(): float
- _is_float(String s): boolean
GPSWrapper
- gps: Adafruit_GPS
+ initialized: boolean
+ setup_gps(String serialDevice): boolean
+ check_gps(): boolean
+ read_gps_latitude(): float
+ read_gps_longitude(): float
+ read_gps_altitude(): float
Co2Sensor
- arduino_serial: Serial
+ initialized: boolean
+ setup_arduino(String serialPort): boolean
+ read_co2(): int
- _is_int(String s): boolean

LoRa_sender
The LoRa_sender class is written in C and also runs on the Raspberry Pi. It establishes a connection to the LORA SX1276 Module on the Dragino Hat and utilizes the chips LORA capabilities. The LORA settings are strongly based on the recommendation of the LORA-Allianz[10].

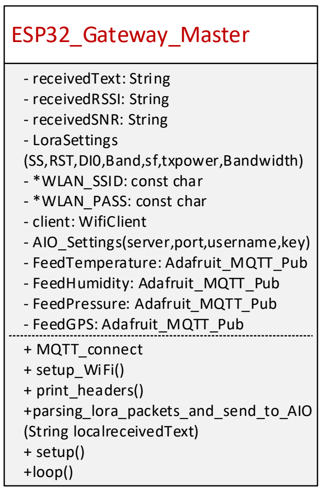


[10] https://LoRa-alliance.org/

Read_co2
The main program on the Arduino. This class is constantly listening for new serial messages sent by the Raspberry (loop method). On receival, it checks whether the message matches a predefined keyword. If so, the Co2 data is read and sent to the Raspberry Pi.



ESP32_Gateway_Master
The ESP32_Gateway_Master class is written in C and runs on the Heltec ESP32. In its Loop function, it continuously listens on incoming LORA messages. If a LORA Package is received, the program will analyse it and look for specific strings like “temperature”, “pressure”, “gps”, “humidity” or “co2”.
When one of these tags is discovered, the value part of the payload will be sent to the corresponding MQTT-Feed on the Adafruit.IO Backend.


ESP32_Gateway_Master
- receivedText: String
- receivedRSSI: String
- receivedSNR: String
- LoraSettings (SS,RST,DI0,Band,sf,txpower,Bandwidth)
- *WLAN_SSID: const char
- *WLAN_PASS: const char
- client: WifiClient
- AIO_Settings(server,port,username,key)
- FeedTemperature: Adafruit_MQTT_Pub
- FeedHumidity: Adafruit_MQTT_Pub
- FeedPressure: Adafruit_MQTT_Pub
- FeedGPS: Adafruit_MQTT_Pub
+ MQTT_connect
+ setup_WiFi()
+ print_headers()
+parsing_lora_packets_and_send_to_AIO (String localreceivedText)
+ setup()
+loop()

6 Evaluation/Experiments/Results/Discussion

6.1 Getting Started with LORA

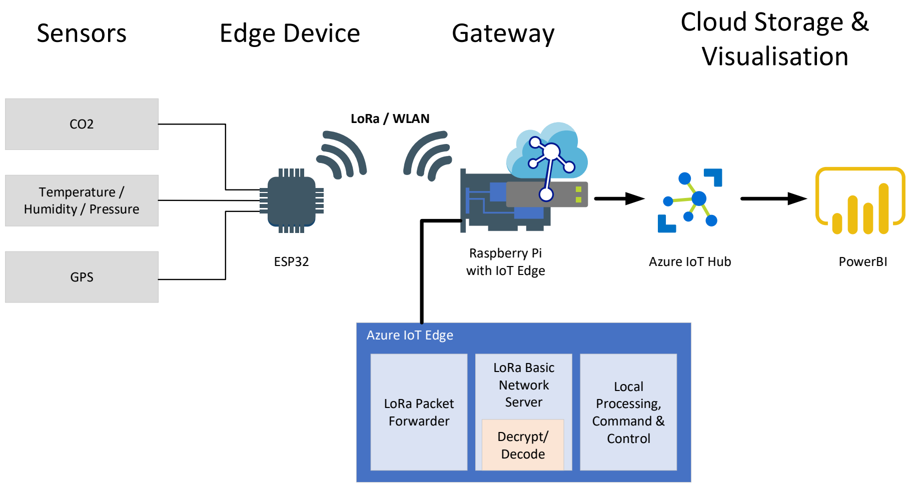

Sensors
Edge Device
Gateway
Cloud Storage & Visualisation
CO2
Temperature / Humidity / Pressure
GPS
LoRa / WLAN
ESP32
Raspberry Pi with IoT Edge
Azure IoT Hub
PowerBI
Azure IoT Edge
LoRa Packet Forwarder
LoRa Basic Network Server
Decrypt/ Decode
Local Processing, Command & Control

Figure 8: First Solution-Design, based on Azure

Our first design was strongly based on "Azure IOT Edge - LORAWAN Starterkit" by Microsoft. After reading the documentations and watching the tech-talks about the IOT Integration Features from Azure, we were fairly excited about their cloud-based solution. We tested the Azure integration with a Raspberry Pi and random generated Data, but without the LORA Module. The tests looked promising. We also added a PowerBI Integration to our Azure Setup for visualization

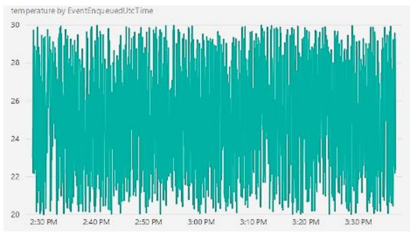


Figure 9: Sample Data, piped to PowerBI for Visualization

Soon after we received the LORA hardware and started with the implementation of the Azure based solution, we ran into several problems. Errors in the code, azure connection timeouts, configuration errors in the azure templates and so on. After a combined 24 hours of work, we decided to stop investing in this solution, because the whole Azure IoT implementation seemed rather experimental and not applicable in practice.

6.1.1 Switching to TTN and Windows 10 IOT

After we discarded our first design, we decided to switch to a TheThingsNetwork (TTN) and Windows 10 IOT based solution. We decided to switch to this solution because the hardware used in this project was the same as ours. We also wanted to test Windows 10 IOT operating system for the Raspberry Pi. After Setting up the Raspberry Pi 3B+ with Win10IOT (which included some heavy tweakings of the OS files), we tried to run our first example applications for the Dragino HAT. Although we were able to access the Dragino-HAT and read GPS data, sending and receiving LORA messages was not possible at all. We again, decided to not waste any more time because time was running out for us. We agreed upon switching to a Linux based solution again and to just focus on the core functionalities.

Figure 10 Successful deployment of Win10 IOT

6.2 Redesign

After discarding our previous solutions, we decided to switch back to a more basic approach. We started with tweaking the source code for accessing the LORA chip on the Dragino HAT from Dragino corp. After some heavy tweaking of the code, we were finally able to send LORA messages with the Raspberry, but unfortunately not receiving messages (see chapter 6.2.1).

6.2.1 Note about the chosen Edge and Gateway devices

As described in chapter “System Overview”, the Raspberry Pi acts as the Edge device in our architecture. This seems rather unusual considering that we wanted to build a low-power solution. The original plan was to use the Raspberry as the Gateway and the ESP as an edge device. Unfortunately, we did not manage to calibrate the Raspberry (or rather the Dragino HAT) to act as a LORA receiver. After hours of unsuccessful tests, it was then decided to change the roles of those two devices, since the Dragino HAT worked properly as a transmitter. In a future project we will definitely opt for more reliable hardware as a gateway.
Because of this late change of plans, we had to compensate for the shortcoming of a missing analog pin on the Raspberry Pi, which our Air Quality Sensor requires. We agreed to use an Arduino as a Bridge instead of utilizing an ADC (Analog to Digital Converter), because the Arduino was already available. It further gave us the opportunity to check how a Serial Connection is built between a Raspberry and an Arduino.

6.3 Testing & Optimizing

After getting our redesigned solution to finally send and receive LORA messages, we started refactoring our whole code, since the sensor implementations were already prepared for the ESP. The sensor part was rewritten in Python for the Raspberry and the Gateway/Adafruit Part was rewritten in C for the ESP. After getting the devices to successfully exchange the correct measurement data, we started with our first field tests, where we drove around in a Wardrive[11] like manner, measuring data and testing the maximal distance of the LORA communications.

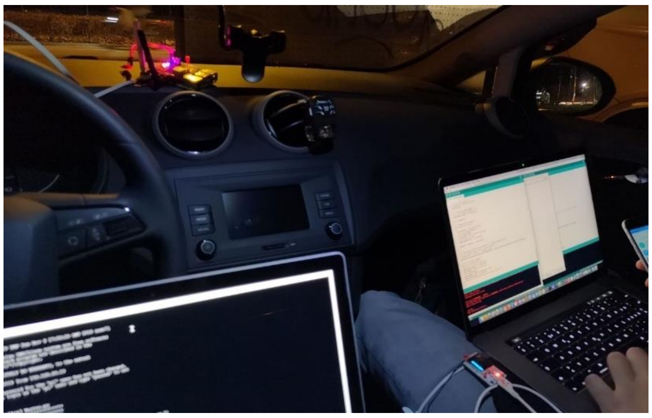


Figure 11: Fieldtesting LORA

While testing our Co2 Sensor we realized that the returned value is not meaningful. According to the documentation, the Sensor signal does not indicate an absolute Level of Air Pollution but rather returns a relative value indicating whether the Air Quality is good(1), bad(2) or very bad(3).

[11] https://de.wikipedia.org/wiki/Wardriving

We noticed that our transmission distance was very poor at first (~200m). Therefore, we tweaked our LORA settings for more distance (at the expense of bandwidth).

With a Spreading Factor of 12, we were able to reach distances over 500m on a flat area. However, the distance in densely populated areas like a city is limited if you are near the ground. The only solution for this problem would be to position the antennas on high ground, above any buildings/obstacles.

The pictures below show the distance over which we could successfully send messages. It was interesting to see how the RSSI of the incoming packets changed from -50 at start to -116 right in front of the train-station. We could have tested longer distances, but the time ran out and we had to come back to the classroom.

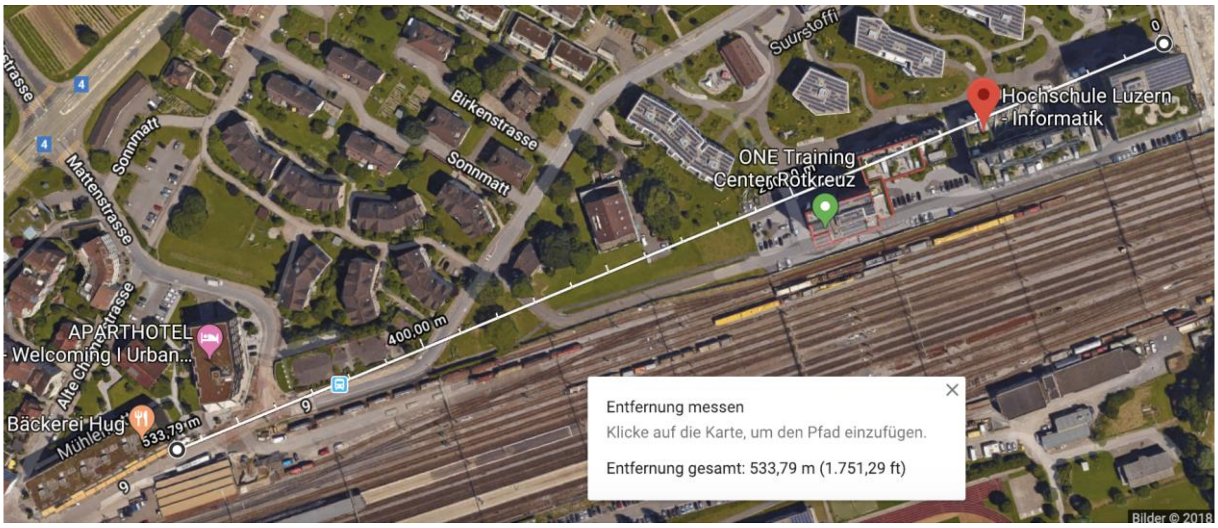

Hochschule Luzern - Informatik
ONE Training Center Rotkreuz
APARTHOTEL Welcoming I Urban...
Bäckerei Hug
Birkenstrasse
Sonnmatt
Mattenstrasse
Suurstoffi
400,00 m
533,79 m
Entfernung messen
Klicke auf die Karte, um den Pfad einzufügen.
Entfernung gesamt: 533,79 m (1.751,29 ft)
Bilder © 2018

Figure 12: Final Field Test at Rotkreuz – 533m

7 Applications

As described in the introduction, our goal was to build a Low Power, LORA based Measurement Kit (LEM-Kit) to carry out environmental measurements in remote areas or in old buildings. Thanks to the long range of the LoRa transmitter we can deploy our Edge Devices to remote Locations like forests, hills and even mountains, where access is limited. With the chosen devices and algorithms, battery operation over a longer period of time is no issue, which reduces maintenance efforts.

The final use scenario that we imagine is as follows: The gateway is placed at a central, high point (e.g. Lucerne city). The edge devices could then be placed all around in remote areas, for example on the surrounding mountains Pilatus, Sonnenberg and Rigi. The data collected on Adafruit.IO could then be made available to the tourism sector, for example, motivating tourists to explore the mountains around Lucerne. Additionally, the collected data can be used to determine what impact the traffic and industry of cities like Lucerne has on air pollution in the surrounding area.

8 Conclusion

The group reached its goal to implement an Environment Measurement Kit. The sensors gather different kinds of environmental data and send it via a low power transmission protocol to a gateway. This gateway uploads these data to a Cloud platform, which then displays this data in an appealing form.

Generally speaking, the project was a success. The team members learned a lot, especially about wireless data transmission, IoT development platforms like Arduino and Raspberry and the implementation of communication to different kind of sensors.

The two highlights during the project were the control of the different sensors and the data communication via LORA.

If more time would be available, the group would have put the devices in separate cases and try to deploy them outdoor. Likely, the enclosures could be 3D-printed at the HSLU, with the available resources at Rotkreuz. We would further like to optimize the Software in order to improve battery life of the edge device. One possible way of doing this would be to dynamically set the polling frequency of the sensors or by only sending data to the gateway, if there was a change greater than a predefined threshold.

One drawback was the work with the Dragino LoRa HAT. Because of its lack of detailed documentation, the group members lost a lot of time with “try-and-error”-kind of work. In another project, the group members would invest more time into platform-researching to circumvent situations like that.

9 Contributions / Acknowledgments

It was great to see that every group member was committed to this project. Therefore, every member did his work to successfully implement the final result. Because of this, the role description in Table 3 does fit the contributions of every team member.

10 Major Milestones & Deliverables

10.1 Team and Roles

This section describes the team, members roles, and what work packages the members are working on.

Table 3: Work Package Assignment

TEAM	PROJECT WORK PACKAGES	OWNER
Group 5	Final Report	Roman
	Edge Device (Sensors)	Stefan & Roman
	Edge Device (LORA)	Pascal & David
	Gateway (LORA)	David & Pascal
	Gateway (Adafruit.IO)	David
	Visualization (Adafruit.IO)	Group
	Presentation	Pascal

10.2 Project Planning, Timelines, Milestones & Deliverables

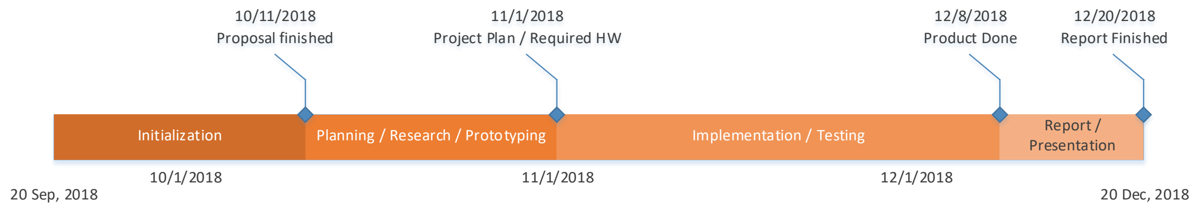


Figure 13: Project planning

To properly organize our project, we created a simple timeline with the main milestones we attempted to achieve. Furthermore, we created work packages, these enabled us to work on different problems autonomously.
Weekly meetings ensured, that we stayed on track. The meetings additionally gave us the possibility to exchange our gained insights and discuss further tasks and problems.

By using collaboration tools like GitLab, OneDrive and Telegram, we were able to communicate and share our progress instantly.

The project was split into the following phases:

Initialization:	Project setup, building groups and brainstorm possible ideas and use cases. By the end of this phase, we delivered the proposal for our project.
Planning / Research:	This phase was mainly used to further substantiate our idea and to do further and deeper research (what hardware to use, where to get the hardware, when will we be able to work on the project). We also created a timeline, set milestones and goals.
Implementation / Testing:	After completion of the research phase, we were able to implement our planned setup. The main target was to create a functioning product which we could present.
Report / Presentation:	We finalized our report and prepared the presentation, which will be held in January.

10.2.1 Actual Project Progression

In comparison to the planned process we had to slightly delay milestone 2 and 3. Our main problem with milestone 2 was, that the ordered hardware did not arrive before November 4th. Because of implementation problems discussed earlier, we had to delay milestone 3, but we were still able to finish the product on time.

The actual progression of the project is indicated by the blue bar:

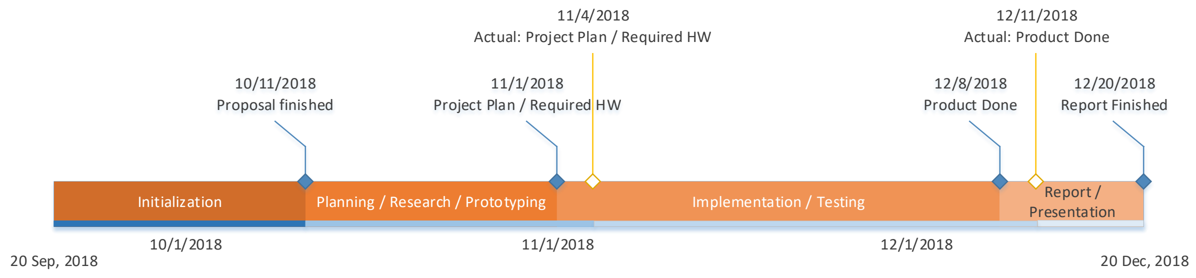


Figure 14: Actual Project progression

11 References

ada, l. (2018, 11 29). *Adafruit BME280*. Retrieved from Adafruit: https://learn.adafruit.com/adafruit-bme280-humidity-barometric-pressure-temperature-sensor-breakout/arduino-test

ada, l. (2019, 11 15). *Adafruit Ultimate GPS*. Retrieved from Adafruit: https://learn.adafruit.com/adafruit-ultimate-gps

Adafruit. (2014, 01 14). *MQTT Basics*. Retrieved from learn.adafruit.com: https://learn.adafruit.com/mqtt-adafruit-io-and-you?view=all

Bachmann, C. (2018, 05 25). *SRF Wissen*. Retrieved from https://www.srf.ch/kultur/wissen/internet-der-dinge-vergessen-sie-den-vernetzten-kuehlschrank

DePriest, D. (2019, 11 15). *NMEA data*. Retrieved from GPS Information: https://www.gpsinformation.org/dale/nmea.htm

Dragino. (2018, 05 11). *Github.com*. Retrieved from RPI LORA Transmitter: https://github.com/dragino/rpi-lora-tranceiver

Larsson, M. (2017, 05 08). *Hackster.IO*. Retrieved from Windows 10 IOT : https://www.hackster.io/laserbrain/a-lorawan-the-things-network-gateway-for-windows-iot-core-441210

Microsoft. (2018, 08 27). *docs.microsoft.com*. Retrieved from IOT Edge: https://docs.microsoft.com/en-us/azure/iot-edge/how-to-install-iot-edge-linux-arm

Microsoft. (2018, 12 12). *Github.com*. Retrieved from LORAWAN Azure IOT Edge Starterkit: https://github.com/Azure/iotedge-lorawan-starterkit

Sangesari, R. (2016, 09 05). *Hackster.IO*. Retrieved from https://www.hackster.io/idreams/getting-started-with-lora-fd69d1

Sangesari, R. (2017, 11 11). *Github*. Retrieved from https://github.com/sourceclimber/LoRa_TTN_Raspberry

Seeed Studio. (2018, 12 06). *Grove - Air Quality Sensor v1.3*. Retrieved from Seeed Studio: http://wiki.seeedstudio.com/Grove-Air_Quality_Sensor_v1.3/

Telkamp, T. (2018, 12 08). *github.com*. Retrieved from LORA single channel packet forwarder: https://github.com/tftelkamp/single_chan_pkt_fwd

Tellado, H. G. (2018, 06 13). *azure.microsoft.com*. Retrieved from IOT Hub: https://azure.microsoft.com/de-de/resources/samples/iot-hub-c-huzzah-getstartedkit/

12 Table of Figures

13 Table of Tables