FACULTY OF ENGINEERING TECHNOLOGY
– GENT –

MOBILE COMMUNICATIONS

LAB

BOEIend!

GROUP 2

Birgen Vermang – Sander Thierens – Brecht Van Eeckhoudt
Louis Devreese – Sarah Goossens

MELICTETC & MELICTICT

November 26, 2018

Contents

1 Description of the problem

'BOEIend!' is a project about giving buoys the ability to communicate with each other, to make them 'smart', so they can **automatically detect and report if they are adrift**. They can also periodically report other useful data such as **wave amplitude and water temperature**.

The goal of the project (and the lab sessions) is to **develop the design and specifications for the communication element of such a 'smart' buoy**, and to **build and test a prototype**. Buoys generally have a lifespan of **5-10 years** so the design must be **energy efficient** and possibly make use of some energy-harvesting methods.

Since the scope of the lab is more about the communication-side, the **low-power aspect should rather be kept in mind than to be actively pursued**. We know however that due to the limited energy budget of some of the buoys, continuous collecting GPS measurements is unfeasible. It is thereby imperative that low-energy strategies are employed to ensure a long lifespan. We can also see that we **need to look more into other localization or proximity-based mechanisms**.

On figure 1 we see a diagram of the general problem. In this project, we assume the following distances:

- Coast ↔ Master buoy (A): 100 m – 1 km
- Neighboring (slave) buoys (B): 10 m
- Immobile buoy ↔ drifting buoy (C): $>$ 20 m

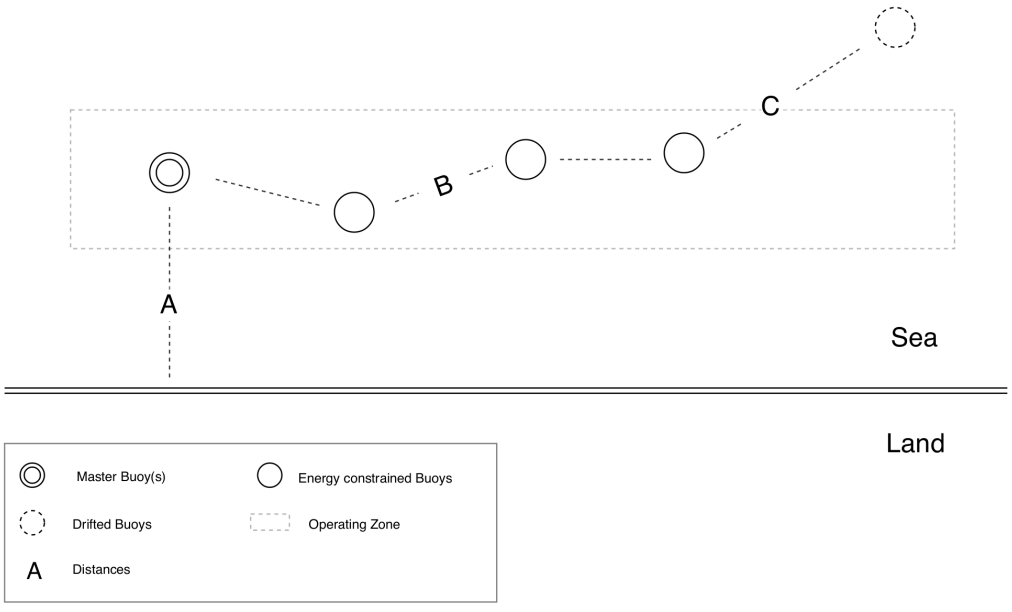

Figure 1: Visualization of the project.

A buoy is considered **out of place** when the distance to his original location is larger than a certain predefined threshold, which is in this case **20 meters**.

Figure 1 also tells us their will be **two types of buoys**:

Master buoy These buoys will be equipped with the technology required to communicate with both the mainland and the other (slave) buoys. It is important to use a technology that is capable of communicating over rather long distances for the connection to the mainland.

These larger buoys will be equipped with larger batteries. For the scope of this coarse we can assume they have 'unlimited power', so they can be used to do more energy-heavy tasks.

Slave buoy These buoys need to be energy-efficient and could generate their own energy. They don't need to communicate over long distances.

2 Wireless communication technologies

Table 1: Possible feasible communication technologies with their most important characteristics.

Communication technology	Range	Bitrate	Power usage
Bluetooth	100 m	1-3 Mbit/s	1 W
Bluetooth Low Energy	50-150 m	1 Mbit/s	10-100 mW
ZigBee	10-100 m	250 kbit/s	10-100 mW
Wifi	30-100 m	150-1024 bit/s	/
Cellular	35 km	35-170 kbit/s	/
LoRaWAN	2-5 km	0.3-50 kbit/s	/
Ultra Wide Band [13], [14]	290 m	850 kbit/s	90 mA - 1 μA

The **choice** of the communication technologies used is **based on table 1, available hardware and the practical implementation of the technologies**. For communication **between the mainland and the buoys** we chose **LoRaWWAN** over the Cellular network (both have the right range-requirements) for it's possibility to easily visualize data in the cloud and lower power usage (see section 4: Strategies for energy efficiency).

For the communication **between the buoys** our first choice was Ultra Wide Band. We knew out of previous experience and the data gathered for a bachelor's thesis from one of our team members that a feasible solution was possible. We ultimately decided however, not to use UWB because we would not be able to get our hands on the required hardware in time to meet our deadlines.

Instead we choose to use the **ZigBee** technology. We chose ZigBee over Bluetooth Low-energy because we had the hardware available and because there is no need for high-bitrate connections.

2.1 LoRaWAN

For the data transfer **between the master buoy and the mainland** we need a connection that **can reach up to 1 km**. The data we want to send to the mainland is **the battery level of each buoy and the signal strength between neighbouring buoys**, so we can detect if any are drifting. With these requirements we decided to use LoRaWAN-connection.

Reasons to choose for LoRa in our case include the **great range and the low power-consumption**. On the master buoys we theoretically have an 'unlimited' amount of power, but a low power design is always preferred.

The LoRaWAN network by Proximus [1] covers almost the whole area of Belgium, as depicted on Figure 2. We can see that the network reaches a few kilometers in to the North Sea. This makes it a perfect candidate for our connection between the master buoys (equipped with a LoRa-module) and the mainland. The data coming from the master buoys will be transmitted over the LoRa network to 'the cloud', where it can be collected and further analyzed.

In the lab sessions we will experiment with a `EMF32 Happy Gecko` development-board with a `DRAMCO LoRaWAN RN2483` modem addon board.

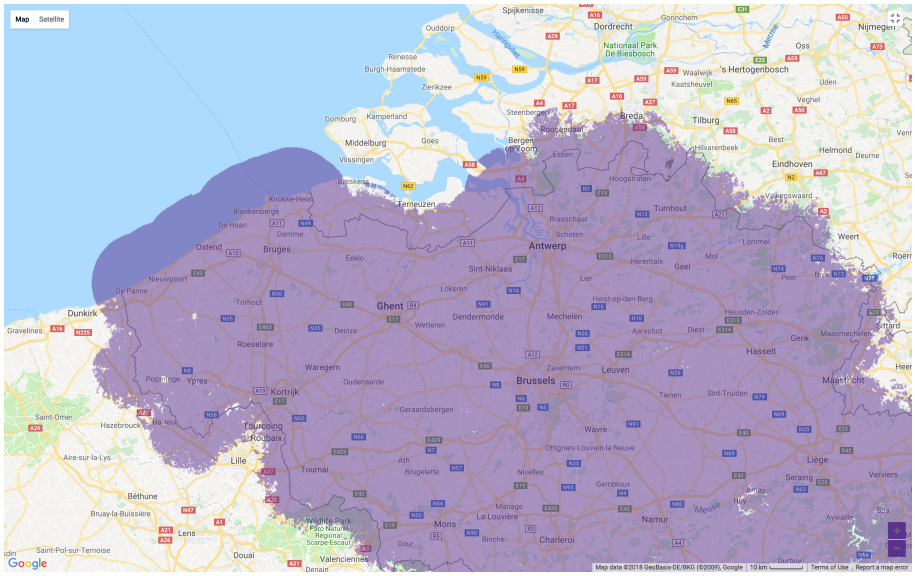

Figure 2: Coverage of the LoRaWAN-network by Proximus.

2.2 ZigBee

As stated before, we chose ZigBee as the technology used for the **communication between buoys**. It is a perfect fit being **very power-efficient, low cost and with a range up to 100m**. We will experiment with two `XBee Series 1 modules`, two `DRAMCO XBee shields` and two `Arduino UNO's`. To configure the XBee modules, we also used an `XBee Interface Board`. Next section will explain how these components are put to use.

3 Practical implementation

Starting with the data given in our assignment (also depicted on figure 1) we can see that each buoy and the next will be **10 meters** apart. If we determine that **every 15 buoys a master buoy** is needed, we can see that the master buoy needs a communication protocol which is able to communicate in a **star-network of up to 70 meters**. This means that every master buoy needs to communicate with 7 buoys to the left and 7 buoys the right. An overview of this proposed implementation is shown in Figure 3.

A communication technology that can handle these distances is `ZigBee`. This is an `IEEE 802.15.4`-based specification for a suite of high-level communication protocols used to create personal area networks with small, low-power digital radios. Use-cases can be found in home automation, medical device data collection and other low-power low-bandwidth needs. This communication technology can communicate between 10 and 100 meters with a bit rate of approximately 0,12 Mbit/s [2].

With ZigBee every 'slave' buoy will communicate with the master in a star topology. When all the buoys are in the right position and the distance between every master buoy is 150 meters, each slave buoy within a distance of maximum 70 meters (if he is at his correct position) can communicate with his designated master buoy.

This also means that **a drifting buoy** can still communicate with the master buoy[1] for some time. However, this becomes less reliable over a range of more than 100 meters, which corresponds to **30 meters drifting**.

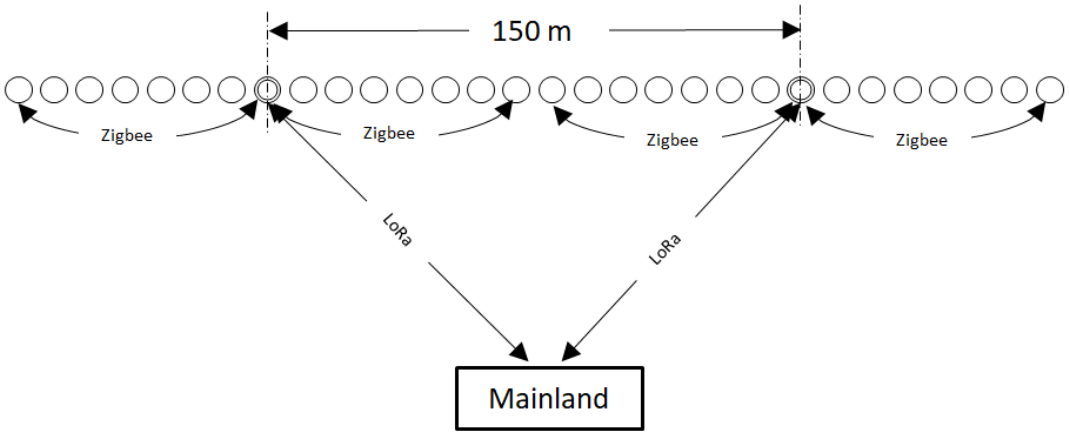

Figure 3: The overview of the proposed implementation.

[1] All master buoys communicate to the mainland over the LoRaWAN network.

3.1 Slave ↔ slave connection

As previously mentioned, a slave buoy's main components are an `Arduino UNO` and a `ZigBee Module`. When we configure our ZigBee modules to work in **API mode** we can simply **import a library into our arduino-code and use the ZigBee module**. A slave buoy knows his left and right neighbour by name, so to say. We use the `'MY field'` as a unique identifier for every ZigBee module. When we look back at figure 3 we see that a neighbour can be both a master or a slave buoy. Therefore **a master buoy has to implement all the functionality that a slave buoy has**.

To **detect whether a buoys is drifting or not**, we use the **signal strength** of the ZigBee communication. This physical quantity is presented by the **RSSI (Received Signal Strength Indicator)**. The greater the distance between two ZigBee modules, the bigger this RSSI-value will be. This is due to the fact that RSSI is a **negative-logarithmic unit**. When we look at the characteristic below in figure 4, we see this negative-logarithmic trend which is very important to interpret the results.

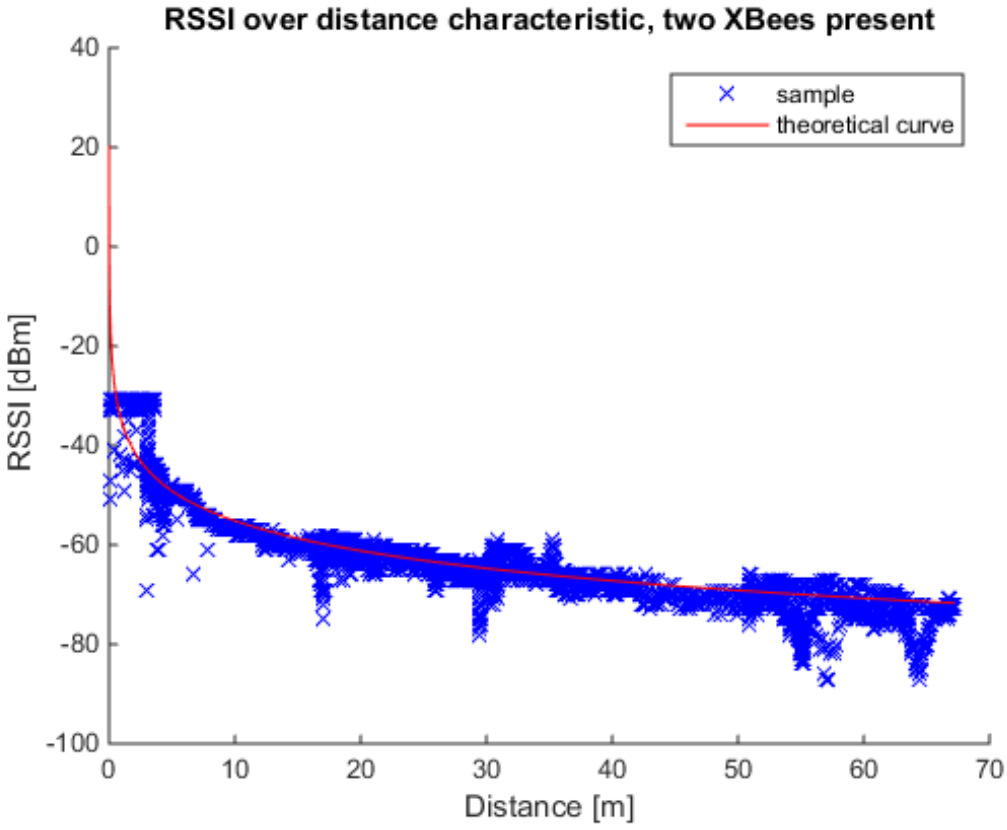

Figure 4: RSSI over distance.

We can use this method of detection because we **only have to detect if a buoy is more than 10 m away**. Imagine a buoy is adrift (20 m away from it's normal postion), the RSSI be would be around `-60 dBm`. When the buoy is still in it's original position it would be around `-55 dBm`. These distances are still distinguishable.

Now imagine we had a normal distance of 50 m between each buoy (around `-65 dBm`). When a buoy is drifting away here it would be more difficult to detect. If a buoy is 100m away, the RSSI-value is also around -65 dBm. In conclusion, **it is more difficult to use the RSSI-over-distance-method for larger distances because of the horizontal asymptote trend** [12].

Periodically a **slave buoy will let his neighbouring buoys know that he is still 'alive' by pinging to them.** The **neighbouring buoy will then calculate the RSSI value** of this message. This 'protocol' so to say happens at every slave buoy, which means that each buoy knows its distance to his left and right neighbour. After a set amount of time, a **slave will calculate the average RSSI-value of both of his neighbours** and **send this info and other sensor data** (see section 6) to the master buoy.

3.2 Proof-of-concept slave ↔ slave connection

To prove our concept for the slave ↔ slave communication, we used **two Arduino UNO's**, **two DRAMCO Zigbee Arduino shields** and **two XBee modules**. The first Arduino sends a message to the second Arduino by using the `'MY value'` of the XBee module on the second Arduino. The second Arduino then receives this message and calculates the corresponding RSSI value. The second Arduino then sends a packet back to the first Arduino. This packet consists of the **ID** of the second Arduino, the calculated **RSSI value** and the **voltage** on the 3,3V pin of the Arduino. The first Arduino then receives this packet.

In a later development stage this packet would be sent and received by the master buoy that is responsible for that specific slave buoy. The LoRa module will then send this data to the mainland where the data will be analyzed to detect if a buoy is drifting. *This flow will be shown in a demo.*

3.3 Master ↔ mainland connection

To **split the development of code** (for the proof-of-concept demo) **for the ZigBee and LoRaWAN connection**, we opted to use both an Arduino Uno (with the ZigBee shield) and a Happy Gecko development-board (with a LoRaWAN modem) **on the master buoy**.

The Arduino sends ZigBee bouy-data (buoy-ID, RSSI and battery voltage) in the form of `strings` over **UART** to the Gecko board. This captures these strings using interrupts, parses them back to numbers, packs them into a *Low Power Payload* and sends it over the LoRaWAN network to the cloud.

4 Strategies for energy efficiency

4.1 Arduino and ZigBee

If we use an Arduino Uno with a ZigBee module without any modifications, the Arduino Uno (`5V`) uses around `45 mA` [4], which is not very energy efficient. This also doesn't include the power consumption of the ZigBee module. If we implement that the **Arduino goes to deep-sleep mode after every transmission**, we can lower its power consumption.

To lower the power consumption caused by the transmissions of the ZigBee module, it might be better to **send all gathered data at once so the ZigBee module doesn't have to wake up for only one transmission**.

If we want an even more low-power setup, it's better to replace the Arduino Uno with e.g. an **Arduino Mini** (`5V`). This needs a power consumption of `23` μ`A` [4] when deep-sleep mode is implemented.

Because ZigBee has a sufficient range, we **first decided to 'connect' every slave buoy directly to a nearby master buoy**. With this method the slaves only need to send data, which is ideal when designing low-power applications because **listening and receiving data generally requires more power than sending**. When testing we noted however that for buoys farther from the master buoy, it became increasingly difficult to correlate a change in position to the RSSI value. This phenomenon is further discussed in section 5.1.

4.2 Happy Gecko and LoRaWAN modem

Since the `EMF32 Happy Gecko development-board` is geared towards low-power development it **has a lot of functionality to preserve power**. There are **five 'Energy Modes'** to choose from, each with less functionality available but more power saved. When the EMF32 board is in the deepest sleeping-mode (`shutoff mode - EM4`) it is only drawing `20 nA` [5]. There also exists functionality on the development board to measure the used power.

The `DRAMCO LoRaWAN RN2483 modem addon board` also features both **code-functionality to let the modem go to sleep** (which is implemented in the demo) and **switches on the PCB itself which can cut the power completely to the modem** when configured on the EMF32 [6].

Because the Happy Gecko module is mounted on the main buoy, we don't really have to go in these lower energy-modes since for these lab sessions this buoy has 'unlimited energy' available. **These energy efficiency concepts are thereby kept in mind but not implemented in our demo.**

5 Technical difficulties

5.1 RSSI

For the **detection of drifting buoys** we originally wanted to use the **signal strength** of the ZigBee modules. Therefore we used the RSSI (Received signal strength indication) as a reference for the distance between two buoys. In theory this could be a good indication for the distance, however, **when performing practical tests we found some problems with this approach**.

Because the RSSI is a **logaritmic unit**, we could clearly see a difference when the modules started to move away from each other. However, **when the distance between them increased, the change in distance became less and less noticeable**. We even got about constant RSSI values when we moved one module from 50 m to 100 m away. This behaviour was already mentioned in section 3.1.

When we calculate the RSSI value of a connection, it will always have a slightly different value. According to a study [15] we found, **there is an dependency between the RSSI and the temperature**. This is a negative correlation, as seen in figure 5. If we can predict the RSSI-change in function of the temperature, we can take this change into account for further analysis of the gathered data. This is kept in mind, but is not implemented in our design.

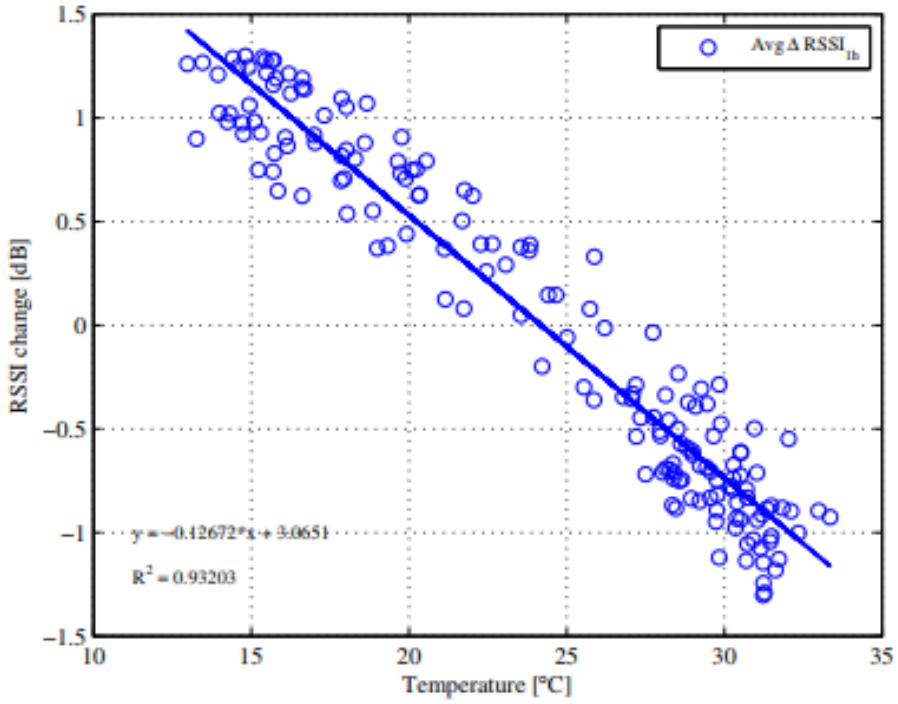

Figure 5: RSSI versus temperature [15].

5.2 UART

During the development of code for the `UART` connection between the Arduino and the Happy Gecko on the master buoy, we encountered a small problem because the **Arduino uses logic levels of 5V**, while the **Happy Gecko board uses 3,3V logic levels**. This could be fixed by using a simple resistor-based voltage divider or in our case a **level-shifter**, as shown on figure 6.

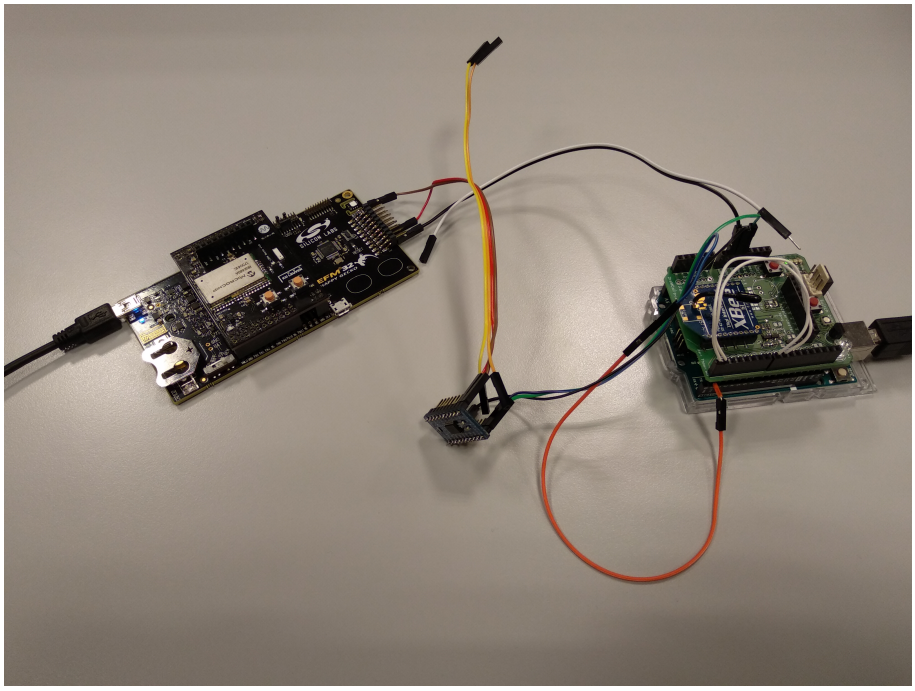

Figure 6: The 'master buoy' (Arduino Uno (right) → level-shifter → Happy Gecko).

5.3 LoRaWAN duty cycle

When using a LoRaWAN-connection **we cannot send data every time we want** because of restrictions put in place to ensure the stability of the network. **We can only send 1% of the time**, the other 99% we have to wait and do something else. In Europe, duty cycles are regulated by *section 7.2.3* of the `ETSI EN300.220` standard [16].

During the development of code for our proof-of-concept demo, we didn't really keep these restrictions in mind. Currently we try to send all of the data from every slave buoy via the master buoy to the mainland with LoRaWAN. On the mainland this information is observed in the cloud and we could use **triggers to detect if a buoy is drifting or a buoy's battery level is too low**.

This, in combination with the restricted length of the LoRaWAN payloads itself (see section 5.4), means that **we can only send the RSSI and battery values of two buoys every five minutes**.

During tests it became clear that this is not the optimal way of detecting drifting buoys. **It is better to collect the ZigBee data of every buoy on the Happy Gecko, detect here if a buoy is drifting or a buoy's battery level is too low and only then send a message to the cloud.** We could also periodically send the status of all the buoys and other sensor data every once in a while. This greatly reduces the amount of data we need to send over the LoRaWAN connection.

5.4 LoRaWAN Cayenne payload (types)

Since **the amount of data we can fit in the payload to send over the LoRaWAN network is limited**, we had to make some compromises.

In our proof-of-concept demo we send the **RSSI-value** (disgused as a *'digital input'* data type for the Cayenne cloud interface) to the cloud using a `uint8` (one byte) value. Since the string coming from the `UART TX` pin from the Arduino is (by design) only two decimal characters for the RSSI value, the maxium value can be `'99'`. This is enough for our application. The **corresponding buoy-ID** is also send using a `uint8` value, but since we reserved three decimal characters in the string coming from the Arduino it's maxium value can be `'255'`.

To send the **battery voltage** to the cloud we use a `uint16` (two byte) value disgused as an *'analog input'* data type for the Cayenne cloud interface. This means we can also reserve three decimal characters in the string coming from the Arduino, so we can display, for example, `3,25V` in the cloud interface. We **have to offset the buoy-ID by one for the battery voltage** however, since the data would otherwise be received by the cloud interface but the RSSI-value would be immedeately overwritten by the battery voltage since it is coming from the same 'sensor' because it has the same ID. **This means the total amount of possible buoy-ID's is divided by two.**

6 Possible sensors and data measurements

The scope of the lab sessions was to develop and design a communication setup between different buoys and the mainland. We didn't really look into all of the sensors we could use to measure, for example, the water temperature or the wave amplitude.

The only 'sensor data' we collect is the **air temperature of the main buoy** (which is read but **not send to the cloud** so we can send other data) and the **battery level of each slave buoy**. Nevertheless we could still add an **accelerometer or gyroscope** to measure the amplitude of the waves and a **temperature probe** for measuring the sea water's temperature. This will in turn decrease our battery-lifespan of since we need to use more power.

We can't add an infinite numbers of sensors to this setup however, because we are limited in the payload of the LoRaWAN-connection (see section 5.3).

7 Conclusion

During the lab sessions we designed a wireless communication configuration (see section 3) between the different buoys. For the proof-of-concept of our design, we experimented with an EMF32 Happy Gecko development-board with a DRAMCO LoRaWAN RN2483 modem addon board that functions as the master-buoy. With this development-board we were able to send the received data from the slave-buoys to the cloud. The slave- and master buoys were equipped with a ZigBee shield so we could estimate the distances between each buoy.

During testing of our prototypes, we have experienced some difficulties in the communication setups (see section 5) like for example the limited amount of data we can send to 'the cloud' using LoRaWAN. We also experienced the negative logarithmic trend of the RSSI-value in real life tests (see figure 4). We saw that the RSSI-value is not always constant at the same place. This probably has something to do with some side effects of wireless communication e.g. *multipath fading*. We also learned that the ambient temperature has a significant negative influence on the RSSI-value.

For our final design, we need to combine all of this valuable information. A lot of work has already been done for the proof-of-concept demo, but there still is some work left until we can create an implementation ready to be put at sea.

References

[1] Proximus, *LORAWAN*,
https://www.proximus.be/en/id_cl_iot/companies-and-public-sector/solutions/connected-business/internet-of-things.html

[2] Wikipedia, *ZIGBEE*,
https://en.wikipedia.org/wiki/Zigbee

[3] SN2483, *RN2483*,
http://ww1.microchip.com/downloads/en/DeviceDoc/50002346C.pdf

[4] Home-automation, *ARDUINO*,
http://www.home-automation-community.com/arduino-low-power-how-to-run-atmega328p-for-a-year-on-coin-cell-battery/

[5] Mouser, *EM-states*,
https://www.mouser.com/pdfdocs/d0233_efm32wg_reference_manual.pdf

[6] Geoffrey Ottoy, *EFM32 RN2483 LoRa-Node schematic*,
https://github.com/DRAMCO/EFM32-RN2483-LoRa-Node/blob/tutorial/DOC/schematics/v4.0.pdf

[7] RS Components, *11 Internet of Things (IoT) Protocols You Need to Know About*, [20 april 2015],
https://www.rs-online.com/designspark/eleven-internet-of-things-iot-protocols-you-need-to-know-about

[8] LinkLabs, *The Complete List Of Wireless IoT Network Protocols*, [8 februari 2016],
https://www.link-labs.com/blog/complete-list-iot-network-protocols

[9] Eric Hines, *Z-wave vs Zigbee vs Bluetooth vs Wifi: Which Smart Home Technology is Best For Your Situation?*, [19 oktober 2016],
https://inovelli.com/z-wave-vs-zigbee-vs-bluetooth-vs-wifi-smart-home-technology/

[10] Brian Ray, *A Bluetooth & ZigBee Comparison For IoT Applications*, [28 oktober 2015],
https://www.link-labs.com/blog/bluetooth-zigbee-comparison

[11] Brian Ray, *Bluetooth Vs. Bluetooth Low Energy: What's The Difference?*, [1 november 2015],
https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy

[12] RSSI, *Use Xbee rssi for distance measurment*, [29 november 2016],
https://electronics.stackexchange.com/questions/272183/use-xbee-rssi-for-distance-measurement

[13] DecaWave, *DW1000 datasheet*,
https://www.decawave.com/wp-content/uploads/2018/09/dw1000_datasheet_v2.17.pdf

[14] DecaWave, *DW1000 Radio IC*,
https://www.decawave.com/product/dw1000-radio-ic/

[15] Jari Luomala and Ismo Hakala, *rssivstemp*,
https://annals-csis.org/Volume_5/pliks/241.pdf

[16] The Things Network, *dutycycle*,
https://www.thethingsnetwork.org/docs/lorawan/duty-cycle.html