Wireless Sensor Networks Lab Report

Onnig Brulez, Arthur Gautheron, Cyprien Heusse, Emily Holmes, Aude Jean-Baptiste, Romain Moulin, Marco Ribeiro-Badejo

GitHub [1]

Introduction

Wireless Sensor Networks (WSN) applications share a lot of common characteristics, such as a need for low power consumption. Yet, each application remains unique enough that “off-the-shelf” protocols are often unsatisfactory. Unique features such as data rate, security, node mobility, latency or scalability must be fine-tuned for the application. Many of those elements come in the form of a trade-off, justifying further the need for adapted and adaptive solutions.

It quickly becomes apparent that a custom protocol might allow us to optimize performances. We have chosen to work on a smart home application where various sensors are used to monitor an elderly person’s health.

This report introduces some of the technical choices we made based on the chosen application. We then explain the design of the physical layer, Medium Access Control (MAC) layer and gateway link. The final product is presented along with some key application examples.

Technical Constraints for Our Protocol

In the following section, we introduce the application we chose to work on as well as key criteria to judge its efficiency.

Chosen Application

We chose to start from the smart medical home application. As we wanted to make sure to have a functioning system by the end of the allocated time for this project, we decided to focus on a system with only an uplink and no downlink. Thus, we removed the need for actuators.

Our system is a house equipped with multiple sensors. These sensors will help track the health of a sick or elderly person. These sensors may send data irregularly and do not follow a specific pattern (aperiodic). Such sensors can be:

- Emergency sensors, such as fall sensors in the form of bracelets worn by the user. Other sensors are unwanted presence detectors, for example placed in a garden so authorities may intervene quickly.

[1] https://github.com/Aude510/be-wsn/

- Aperiodic (event-driven) sensors for drastic changes in temperature, humidity, or light. Since we want to focus on low-energy consumption, we believe that aperiodic sensors are better to use. Indeed, the radio part of a sensor is a huge consumption post, and we do not need to send temperature data every minute, for example. The aperiodic nature of the signals suppose a need to have some computation power on the sensors to determine when to send the data, but this is outside the scope of this project.

The sensors send their data to a gateway connected to the internet. We consider that this gateway (for example the internet router of the house) has no power-related constraints. A server located on the internet displays the sensors' data.

The overall organization of our application is displayed in Fig. 1.

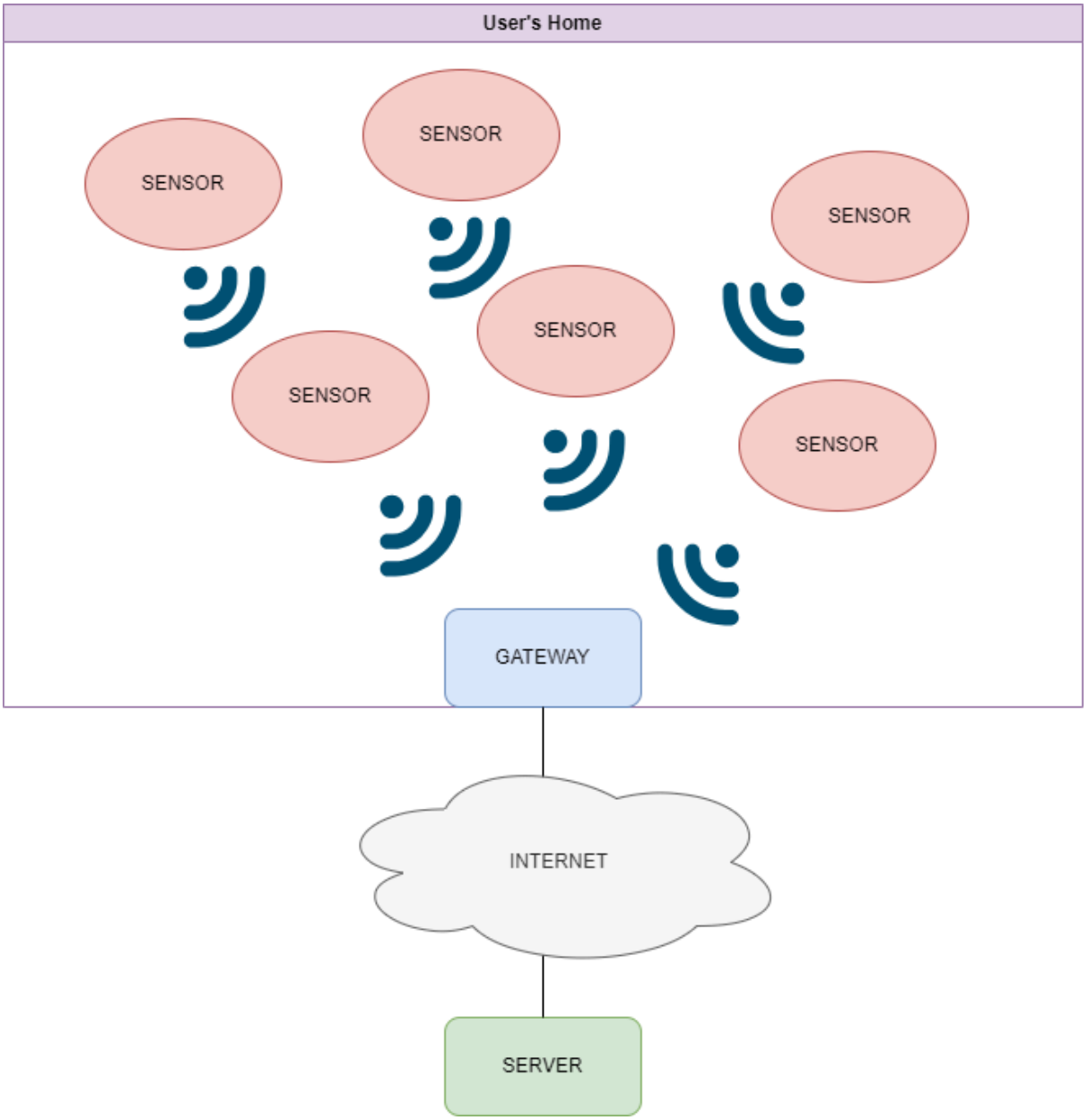

Figure 1: Architecture of our application

Characteristics of our Communication Layers

This application calls for a low data rate, but a need for reliability. We must make sure that the information is received. Ideally, the system should be reactive and as fast as possible.

Here are our decisions for some of the key features of a communication protocol:

- Security: Security is typically implemented at a higher level and not on the physical and MAC layers. However, we do include sequence numbering, so replay attacks are theoretically impossible.
- Quality of Service (wait time, latency, data rate): As we are working with health applications, wait time should be minimized as much as possible. Data rate is quite low, which also limits collision possibilities.
- Power consumption: Power usage should be as low as possible. We could also consider energy harvesting methods to power the sensors. Multiple harvesting methods from the human body exist, such as Shuvo et al.[2] that could be reused.

[2] https://www.mdpi.com/1996-1073/15/20/7495

Design of the Physical Layer

In this section, we present our technical choices and the implementation realized on GNU Radio below.

Frequency and bandwidth Constraints and Choices

The choice of our frequency band was driven by the fact that we need to use an ISM (Industrial, Scientific, Medical) band. Indeed, our application could never be commercialized if users had to support the cost of a licensed-band like in cellular technologies (several hundreds millions of euros for 5G bands [3]).

Amongst the different ISM bands available, we chose to use the 868MHz band, ranging from 863 to 870MHz. This band is already used by protocols widely used in Wireless Sensor Networks like Zigbee or Lora. Indeed, it is perfectly suitable for short-range applications like ours. We based our choice out of the Decision n° 2021-1589 of ARCEP *(Autorité de régulation des communications électroniques)* [4].

We arbitrarily choose a center frequency in this band of 863.2MHz and we used it all along our experimentations. However, for a more advanced project, further reflection on center frequency and bandwidth would be needed. Indeed, we used a bandwidth of 1.5MHz, but this channel size was chosen to avoid undersampling on the USRP. Another solution to avoid this undersampling should be found if further work was accomplished on this application, as this contradicts the ARCEP constraints for this frequency band (between 600kHz and 1MHz for wide range data communications). Moreover, the center frequency should be increased to avoid side channel interferences going out of the ISM band.

To summarize, we tried to base our frequency and bandwidth choices out of real world constraints. However, technical problems on implementation and lack of time makes our implementation imperfect and some further work would be needed to implement it in the real world.

Modulation Constraints and Choices

We chose a DBPSK modulation (Differential Binary Phase Shift Keying). This modulation does not allow big data rates as it has only two symbols (as data rate and symbol rate are identical). However, this makes it quite robust to interferences and should ensure a low Bit Error Rate (BER). The use of differential modulation makes the system even more robust to interference.

The choice of a robust modulation with low data rate, suited to our application needs, also allows us to minimize power consumption of the radio stage of the sensors. Indeed, sending at higher data rates consumes more power.

---

[3] https://en.arcep.fr/news/press-releases/view/n/5g-011020.html
[4] https://www.arcep.fr/uploads/tx_gsavis/21-1589.pdf

Emission

When it comes to emission, we used ZeroMQ (ZMQ). ZMQ is a messaging library similar to MQTT without a broker. By subscribing to a given address, the MAC layer uses ZMQ to send messages to GNU Radio, in which they are transmitted over the physical layer.
We could not get GNU Radio to work with transmission bursts. Instead, a continuous stream of data is transmitted on the wireless medium. Obviously, this is not representative of the real life case application, but it allowed us to prototype the protocol more efficiently. Fig. 2 features the workflow.

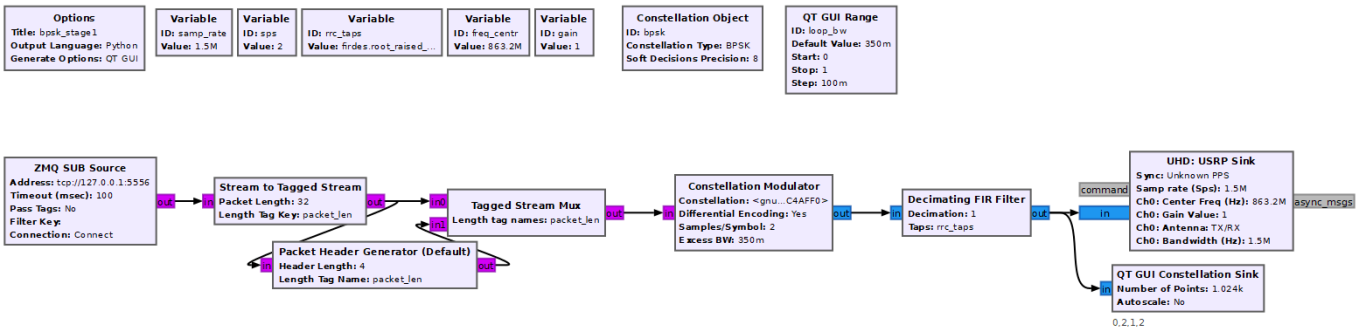

Figure 2: Emission diagram on GNU Radio

ZMQ Sub Source: Used to retrieve the message to send from the MAC layer.

Stream to Tagged Stream: Creates a packet of a defined length. Here, the length is 32 bytes.

Packet Header Generator: Adds a header of one byte to the beginning of each packet. This allows us to synchronize and recognize the start of a packet. GNU Radio generates default headers in the form of a series of 0's that are easily recognizable. We could have made a customized header but kept to the default for our prototype.

Tagged Stream Mux: Connects both the header and the payload of the packet to prepare for modulation and emission.

Constellation Modulation: Specifies the modulation scheme (DBPSK) as well as the number of samples per symbol. This is where the heart of the modulation occurs.

Decimating FIR Filter: A Finite Impulse Response filter, which helps with equalization and removing undesirable frequencies.

USRP Sink: The final signal is then sent via USRP to be received and decoded on the other end.

Reception

For the reception, we repeat similar steps in the reverse order as per Fig. 3.

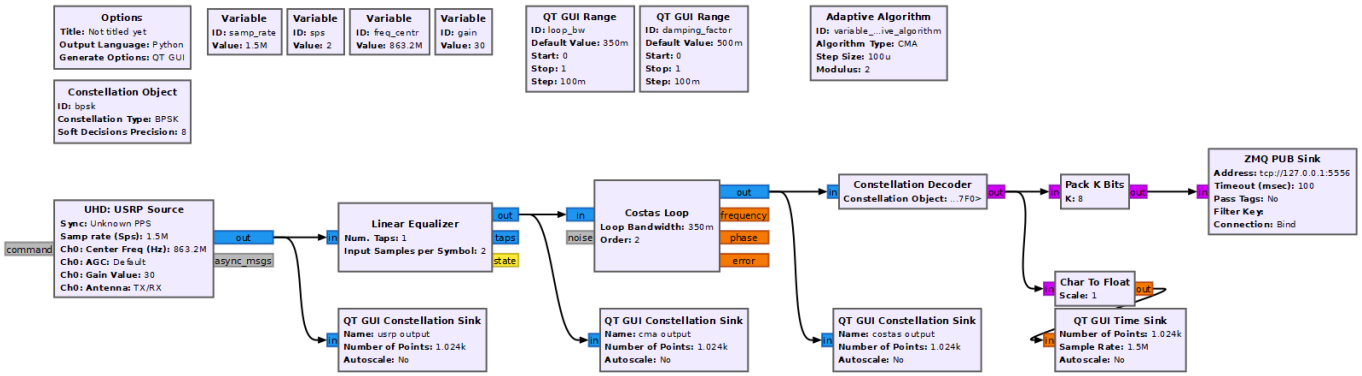

Figure 3: Reception diagram on GNU Radio

USRP Source: Retrieves the signal on the other endpoint. You can notice on Fig. 4 how the constellation points are far from -1 and 1 as they should.

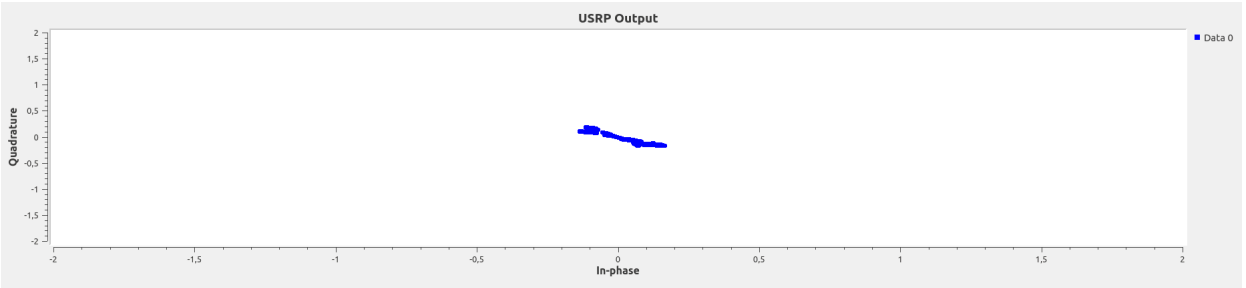

Figure 4: USRP output during reception

CMA Equalizer: Adjusts the gain to be constant over the signal, achieving the desired amplitude to decode the signal as you can notice in Fig. 5.

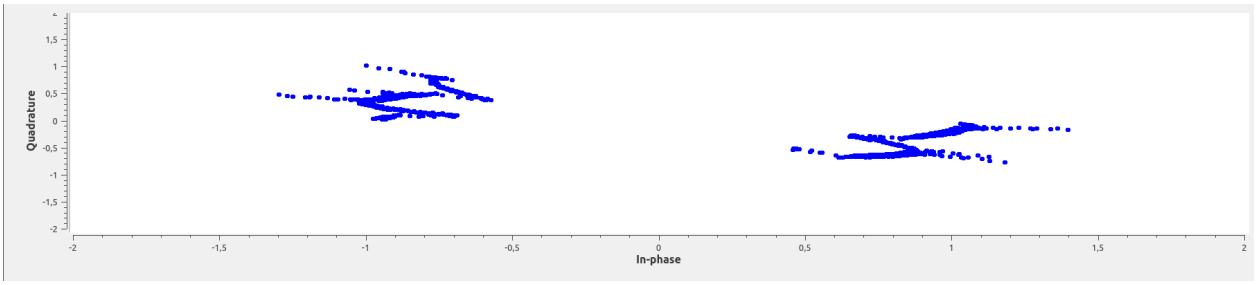

Figure 5: CMA Equalizer output during reception

Costas Loop: A second-order carrier recovery module. The second order enables BPSK recognition. It adjusts the phase offset, see Fig. 6.

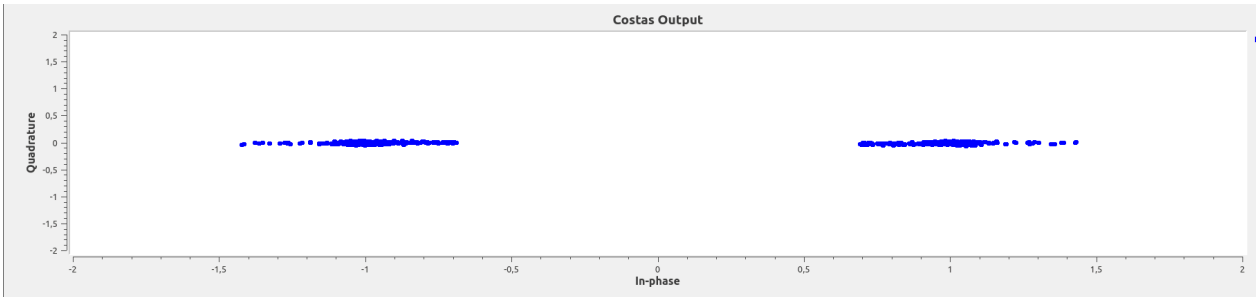

Figure 6: Costas Loop output during reception

Constellation Decoder: Decodes the BPSK modulation.

Pack K bits: The output bit stream is turned into byte packets for easier transmission and readability.

ZMQ Pub Sink: Finally, the output byte stream is sent to the MAC layer using ZMQ.

Due to limitations on GNU Radio, we chose not to remove the header at this stage. The header will be removed on the MAC layer, so it is sent with the rest of the stream.

On Connecting Reception and Emission

We ran into a few issues when connecting the reception and emission from GNU Radio. Extensive testing allowed us to see that packets were correctly formed, so the problem was related to the modulation and demodulation.

We made tests with wireless transmission (as in our application), with two USRPs connected with a cable (conducted signal) and on a unique machine (simulation of a transmission and reception) and on each of our tests, we faced corrupted data on reception.

We believe that error bits are accidentally introduced in the sequence and make it difficult to decode. We have not been able to solve this issue. However, we are confident that our architecture is still a meaningful representation of the process required to create the physical layer of a transmission protocol.

Design of the Medium Access Control layer

Channel Access Type Constraints and Choice

The MAC layer of our protocol is based on the Pure Aloha MAC protocol. It is a contention-based protocol on a single frequency channel. During transmission, data is encapsulated and sent without checking for presence on the medium. Aloha simply consists in transmitting data and waiting for an acknowledgement (ACK) from the receiver. If no ACK is received within a given time period, data is transmitted again until ACK is received.

This protocol fulfills the constraints of our system described above, including:

- One or multiple sensor nodes can send data to a gateway node and get confirmation of the reception;
- Data can be transmitted aperiodically with low rate and low predictability;
- Collisions are avoided by sending an ACK packet to the original sender when the transmission is successful.

The figure below describes a typical data transmission from a sensor node (called Device) and a Gateway node.

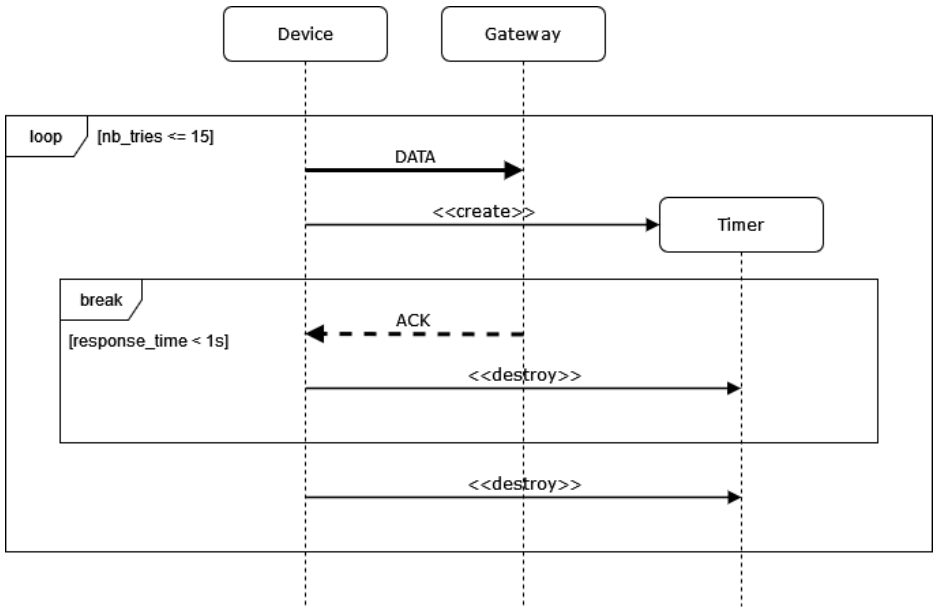

Figure 7: Sequence diagram of a typical communication between a sensor node and a gateway

On the receiving end of the communication, an ACK packet is sent in response only if the following conditions are fulfilled:

- The packet has been received (i.e. the preamble is intact and the received understands a packet is coming);
- The destination address of the packet is the address of the receiver device;
- The CRC checksum is correct.

Frame

In the wireless network, when two nodes try to communicate, they require information about the context of the communication, the purpose of the communication, the type of data transmitted, and whether the communication was successful or not. We chose to encapsulate data in the MAC layer within a packet to add metadata about the communication. The content of the frame within our protocol MamieMAC is described in Table 1.

Name	Size (in bytes)	Content	
Preamble	1	0x55	
Code	1	CODE_ACK CODE_DATA_FLOAT CODE_DATA_INT	0x00 0x66 0x69
Sequence	1	Sequence value (check for ACK)	
Destination Address	1	MAC address of destination device	
Source Address	1	MAC address of source device	
Data	4	data [31:24]	
		data [23:16]	
		data [15:8]	
		data [7:0]	
CRC	1	CRC	

Table 1: MamieMAC communication frame

Preamble: A 8-bit long field filled with alternating ones and zeros. When a receiver device detects a preamble, it goes into reception mode and starts reconstructing the packet byte after byte.

Code: After the preamble, the first byte is a code that indicates the content of the packet.
If the code is equal to CODE_ACK (0x00), then the packet data has no real meaning. It is supposed to confirm the reception of a data packet.
If the code is CODE_DATA_INT (0x69) or CODE_DATA_FLOAT (0x66), then the data must be interpreted as an integer or a float, respectively.

Sequence: The sequence number is an incremental identifier for the packet. It allows the end device to send an ACK packet to the transmitter with the same sequence number in the ACK packet to confirm that this specific packet has been successfully received.

Destination Address: A 8-bit long field for the destination device address. For sensors, this field is supposed to always be equal to 0x01, which is the address of the gateway.

Source Address: A 8-bit long field for the source device address. The source address is always equal to 0x01.

Data: A 4-byte long field for storing data. If the code of the packet is ACK, then the data field is not supposed to be interpreted and will most likely be filled with zeros.

CRC: A 8-bit long field that contains the Cyclic Redundancy Check checksum result. This value is checked at the end of the reception of a packet. If the check fails, then no ACK packet is sent in return.

Technical Implementation

The MAC layer is developed in Python. It is divided into multiple scripts:

- `encoder.py` manages **packet creation**. If the device is on the receiving end of a communication, it creates an ACK packet, else it encapsulates the message to be sent.
- `decoder.py` **extracts information** from received packets.
- `main.py` for sensor nodes creates a thread for **data transmission** and a thread for **ACK reception** and confirmation.
- `main.py` for gateway nodes creates a thread for **data reception** and **ACK transmission**. It is connected to the internet and sends HTTP requests to the server layer.

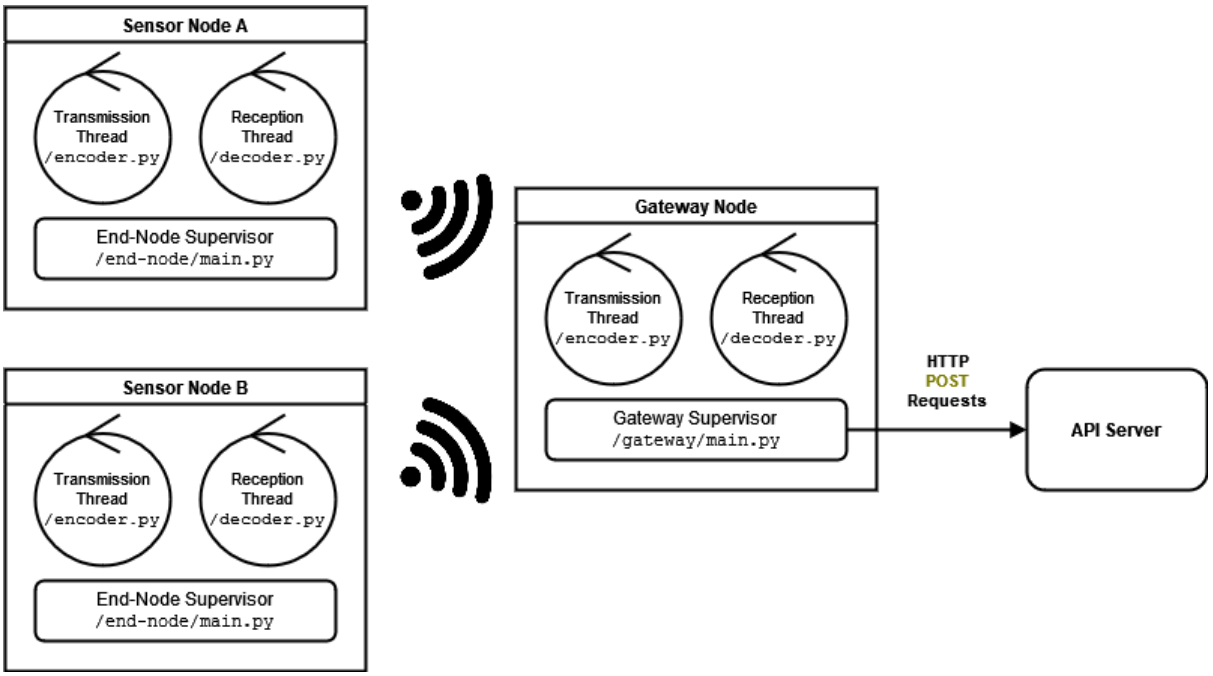

Figure 8: MAC layer implementation

Gateway Link

To establish a link between the gateway and the server, we implemented an API made in Python. It features requests such as “https://api.is-grandma-alive.obrulez.fr/mamie-debout” that consumes a message in JSON format (for example, “{"debout":"0"}“). Upon request, data is saved in a database. 0 means that the person fell down, 1 means that the person is fine.
This interface allows gateway nodes to perform API calls whenever a packet is received from the nodes, while the server saves and processes the received message.

Other API calls allow users to retrieve data to be used on a custom user interface (web or other).

Analysis and Future Works

Finally, we will analyze some of the work done for this project.

Our work is a very simplified version of a network. However, we tried to make all technological decisions as coherent and relevant as possible despite the time constraints and with the tools available to us. For example, while a real protocol wouldn’t have the physical layer and MAC layer talking via ZMQ, it was easy for integration and demonstrated inter-layer communication.

As we only focused on the physical and MAC layer, security options are limited. The integration of a sequence number should in theory prevent replay attacks and add security, but it is not exactly true in our implementation: if a “future” sequence number is received, the system will synchronize with this new number and assume it is the correct sequence. Thus, an attacker could still perform an attack by using a big enough sequence number. Typically, security measures are integrated at a higher OSI level, so this is not a key issue. Still, it was an important factor we kept in mind.

Regarding the work itself, the autonomous work on GNU Radio was difficult. We chose to be a team of students from various majors (IR-SC and AE-SE), but this did not help a lot. AE knowledge in wireless communication was very limited, and SC students have never worked with packets in GNU Radio before. Despite our best efforts, the lack of documentation for the software made debugging quite difficult. The versions also change a lot and have very little to no backwards compatibility.

Nonetheless, this project was an interesting learning experience in many aspects with a stiff learning curve.

Conclusion

This project was an opportunity to develop a wide array of skills and introduced project management tasks as we had to divide a group of seven students into smaller teams. Despite the initial difficulties in designing the protocol, we collectively landed on a solution we are satisfied with and that matches the application we chose.

We had the opportunity to conceive a meaningful protocol. This highlighted the advantages of such a method, by only implementing the tools we would actually need and thus removing “useless” features for our application case. We also saw some of the difficulties that come with it: using an “off-the-shelf” protocol provides easy-of-use, a solution that has been tested and debugged and improved by thousands of users, etc. Ultimately, the decision to use a custom protocol must be thought through at the beginning of a project. It is, nevertheless, an option engineers should consider before diving head first into well-known, heavier protocols that might reveal themselves unsuitable for some of the application features.