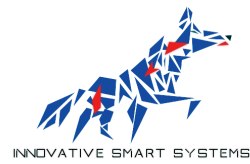




INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
TOULOUSE



BE WSN

Kenza Bouzergan
David Lacoste
Stig Griebenow
Lamiaa Housni
Alexis Payet
Thang Truong

Introduction

In our Wireless Sensor Networks lab for a smart medical home, we took on the comprehensive task of implementing the Physical, MAC, and Application layers of the WSN. This involved a detailed process of designing each layer to ensure seamless integration and optimal performance in a smart home environment. The focus was on creating a network that could effectively manage various types of sensor nodes and actuators, ensuring efficient communication, energy management, and responsiveness to cater to the specific needs of a smart medical home setting.

Requirements

This chapter describes the requirements for the log. The requirements consist of the following characteristics: Mobility, Security, quality of service, Power Consumption

System Overview

The WSN comprises 10 to 100 sensors and actuators strategically placed within a localized environment, such as a smart home. The central point of communication is the gateway, which facilitates bidirectional communication with the sensors and actuators within its range. Additionally, the gateway serves as the bridge to connect the WSN to the internet/cloud, allowing for remote monitoring and control.

Mobility

While the system is designed to operate within the confines of a house, mobility is not a primary concern. The focus is on maintaining connectivity within a fixed environment, ensuring seamless communication between sensors, actuators, and the gateway.

Security Measures

Security is a paramount concern in our protocol. To safeguard personal data and prevent unauthorized access, the entire system operates on a single user-generated key. This key is employed in cryptographic operations to ensure the confidentiality and integrity of the transmitted data. The protocol is designed to be resilient against replay attacks, further enhancing the overall security posture.

Availability Enhancements

To prioritize the response of emergency sensors, the protocol incorporates a mechanism to reserve specific frequencies or time slots.

This ensures that critical information from emergency sensors is promptly relayed to the gateway and subsequently to the internet/cloud, optimizing the overall system's availability. In contrast, the smart home functionalities do not require a high level of availability, allowing for a more flexible approach.

Power Consumption

Given the assumption that sensors are either plugged in or can be charged daily/weekly, power consumption is not a critical factor. The protocol is designed with the flexibility to support sensors with varying power needs, allowing for a balance between energy efficiency and system performance.

Quality of Service (QoS)

The QoS parameters include latency, wait time, and data rate. The protocol ensures a low latency in the order of seconds, meeting the real-time requirements of the application. Wait times are optimized to minimize delays in data transmission. Data rates are tailored to different sensor types, with periodic collectors, actuators, and event-driven sensors operating at a low data rate, while emergency sensors enjoy a higher data rate to relay critical information promptly.

I. Physical layer

The physical layer, often regarded as the foundation in communication systems, plays a pivotal role in the transmission of data between devices. It incorporates the concrete aspects of data transfer, dealing with the actual transmission and reception of signals over a communication medium. In the context of our smart home Wireless Sensors Network project, the physical layer serves as the base upon which the diverse nodes communicate and collaborate. Its design involves diverse considerations of modulation schemes, frequency allocations, and demodulations to ensure efficient and reliable communication. By addressing the requirements of periodic collectors, actuators, event-driven sensors, and emergency sensors, the physical layer facilitates a dynamic and responsive network within the smart home environment.

The primary decision in designing the physical layer revolves around the selection of modulation. Modulation plays a pivotal role in shaping the communication characteristics of a wireless sensor network. The chosen modulation scheme directly impacts factors such as data rate, energy efficiency, and robustness in signal transmission.

1) BPSK modulation

The modulation that we chose for our application is the Binary Phase Shift Keying modulation (BPSK) since it is the simplest and most robust modulation.

In fact, it involves only two phase states (0 and 180 degrees). This simplicity facilitates easy implementation and reduces the complexity of both the transmitter and receiver. Besides, BPSK is more robust in the presence of noise compared to more complex modulation schemes.

Nevertheless, an aspect we need to bear in mind is the limited data rates in comparison to more complex schemes. However, given that certain parameters, such as temperature, do not require frequent updates — for instance, having temperature readings every 30 minutes might suffice — the lower data rate should not pose a significant challenge.

Now, let's delve into the details of Binary Phase Shift Keying (BPSK) modulation before moving on to its implementation in GNU Radio.

What is BPSK?

Binary Phase Shift Keying (BPSK) is a modulation technique employed in communication systems to transmit information via a communication channel. In BPSK the carrier signal is modified by altering its phase by 180 degrees, for each symbol. A phase shift of 180 degrees denotes a binary 0 while no phase shift represents a binary 1. The BPSKs modulation process is straightforward and efficient making it suitable for scenarios where the communication channel suffers from noise and interference.

Modulation Process in BPSK

In BPSK we represent symbols as phase shifts in the carrier signal during the modulation process. A binary 1 is sent without any change in phase while a binary 0 is sent with a phase shift of 180 degrees. This information about phase shifts is encoded into the carrier signal to facilitate data transmission. The constellation diagram, for BPSK, displays two constellation points positioned along the x in phase). There are no points projected onto the y-axis (quadrature) as BPSK relies on one basis function. The phase of the carrier wave carries all the information being transmitted.

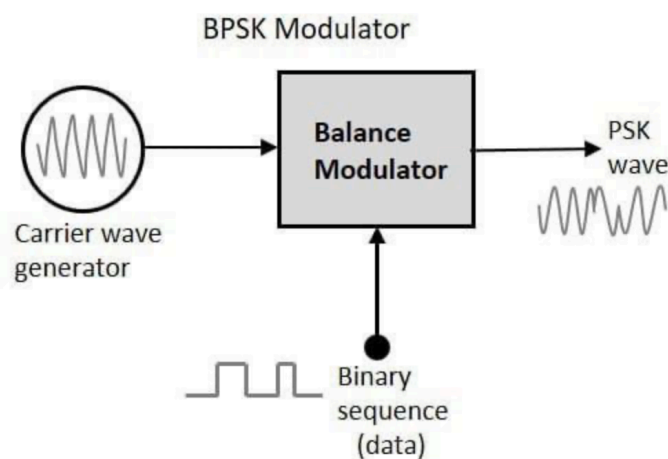


Figure 1 - BPSK Modulator – Modulation Process

BPSK Modulation and Demodulation

If we create message bits and apply BPSK modulation to them, then reverse the process by demodulating the signal we should obtain something like this :

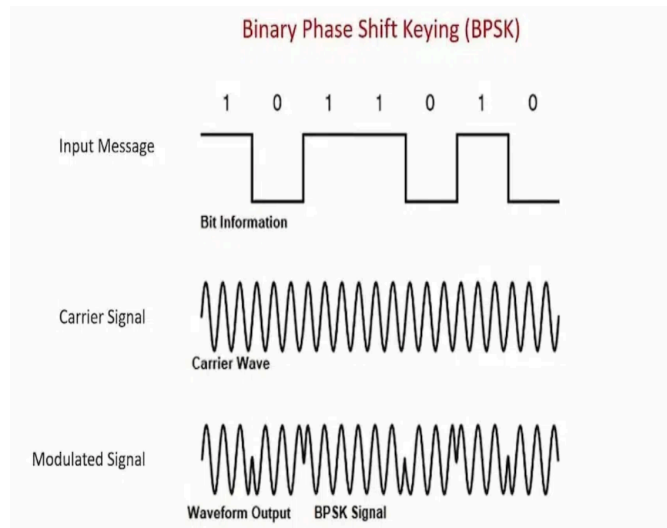


Figure 2 - BPSK Signal Waveforms – End-to-End

2) Simulation of the BPSK modulation in GnuRADIO

First of all, let's start with a simple simulation in GNU Radio without involving the PlutoSDR. This simulation will help us understand the basic functioning of BPSK modulation in GNU Radio.

Here's the flowgraph of the BPSK modulation on GnuRADIO :

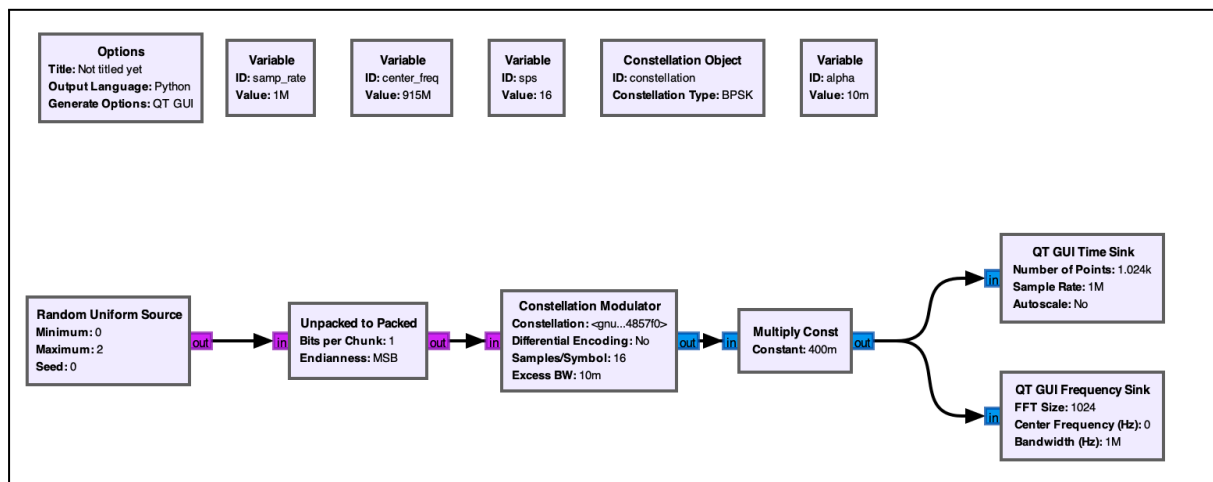


Figure 3 - BPSK flowgraph

When launching :

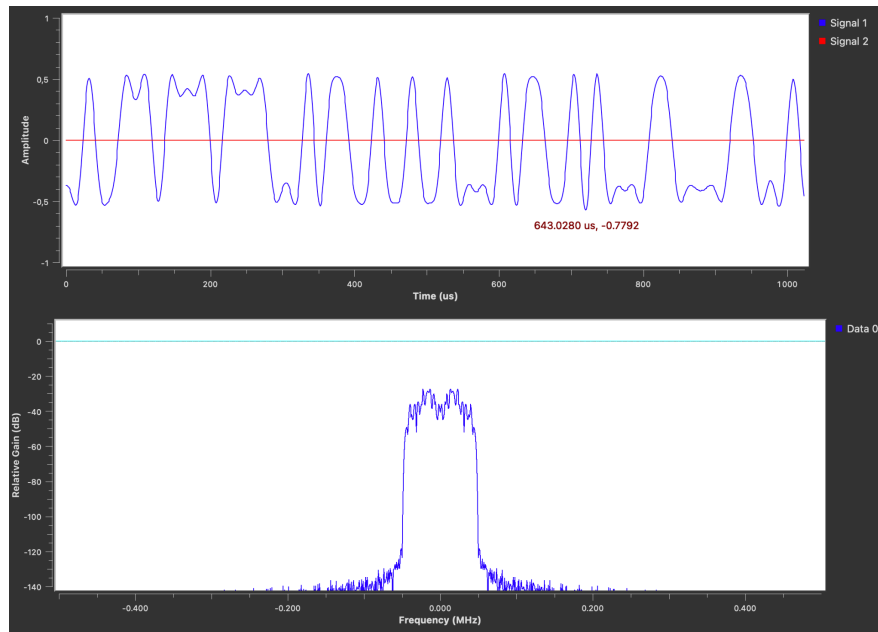


Figure 4 - Modulated signal

We can see that it matches what we're supposed to have according to figure 3.

3) Physical layer in GnuRadio

Now, we will implement the BPSK modulation using PlutoSDR in GnuRadio.

In our experimentation, we integrated the BPSK modulation flowgraph previously developed with a PlutoSDR sink. To complete the loop, we introduced a PlutoSDR source and linked it to a Costas Loop with a Loop Bandwidth set to $2 \times 3.14 / 100$. Subsequently, we employed a constellation decoder for the final step of demodulation.

To validate our data transmission, we added multiple time sinks throughout the block, allowing us to monitor and ensure the integrity of the transmitted data at various stages of the process.

For the test phase, we attempted to verify if the data sent as input was accurately reproduced at the output. However, we encountered difficulties in decoding the initially chosen random file, such as "Hello World." To overcome this, we opted for a Random Uniform Source, sending random sequences of 0s and 1s as input. Given the unpredictable nature of the data, we included a File Sink to write the input data to a file for reference.

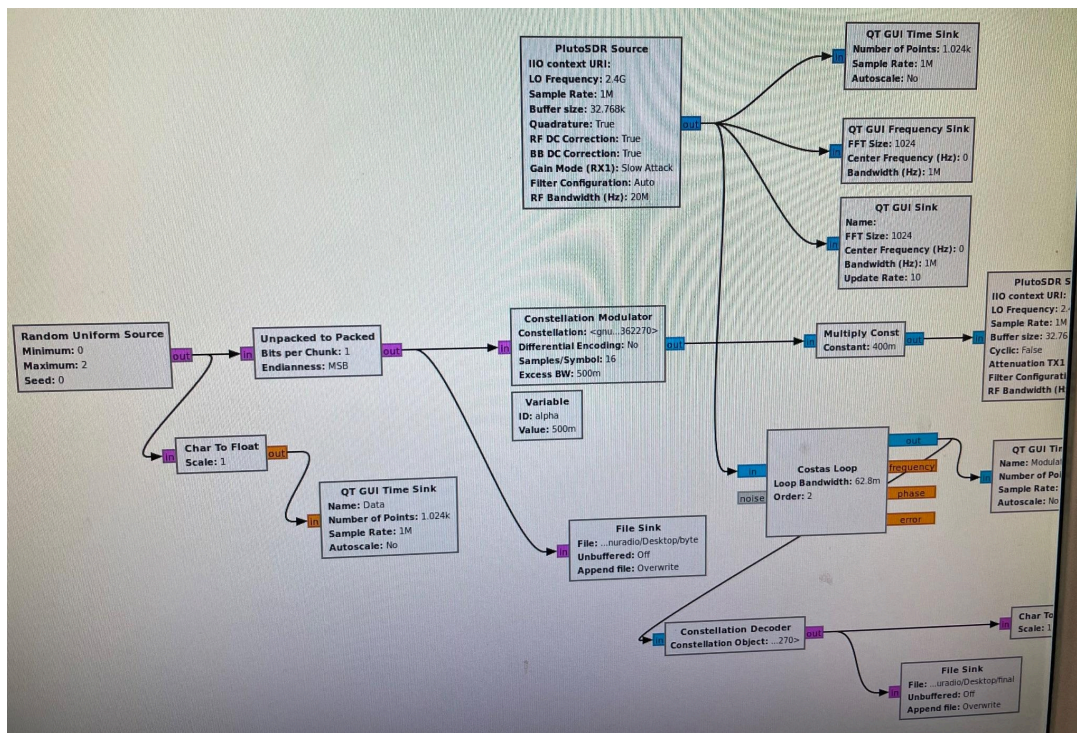


Figure 5 - Physical layer flowgraph part 1

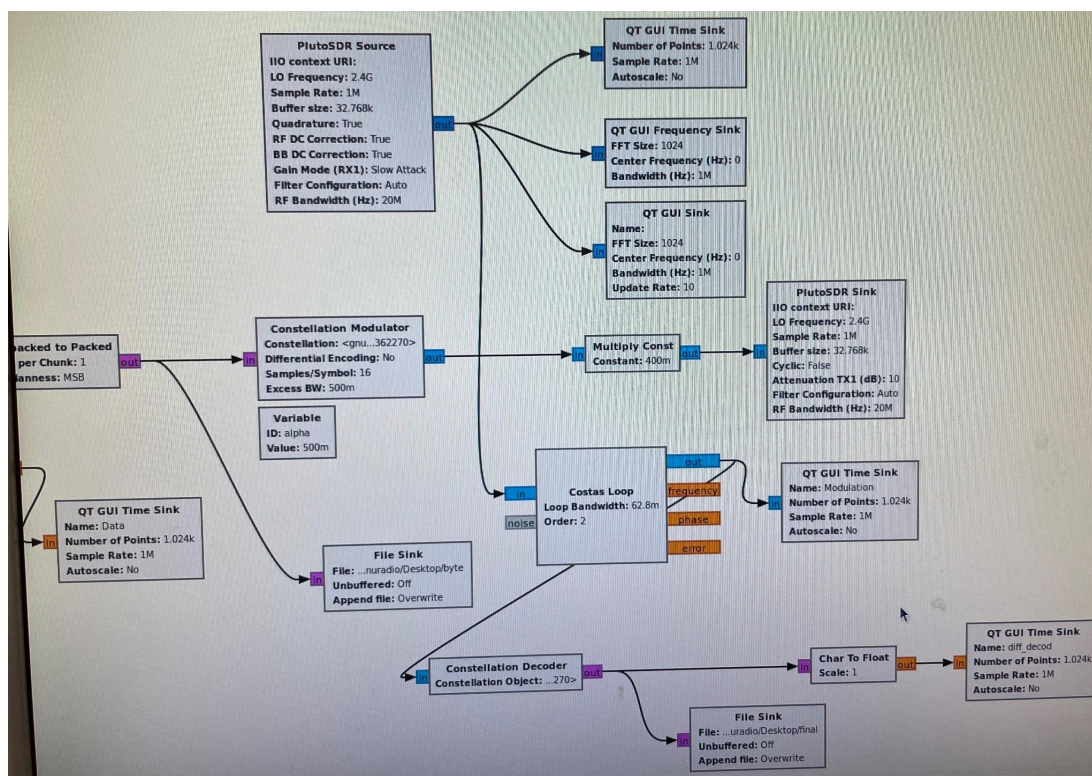


Figure 7 - Physical layer flowgraph part 2

The final step involved comparing the source file with the output file. As the output file was in binary format and couldn't be opened manually, we utilized Python to visualize its contents. This comparison between

the source and output files allowed us to assess the accuracy of the modulation process and ensured the fidelity of the transmitted data.

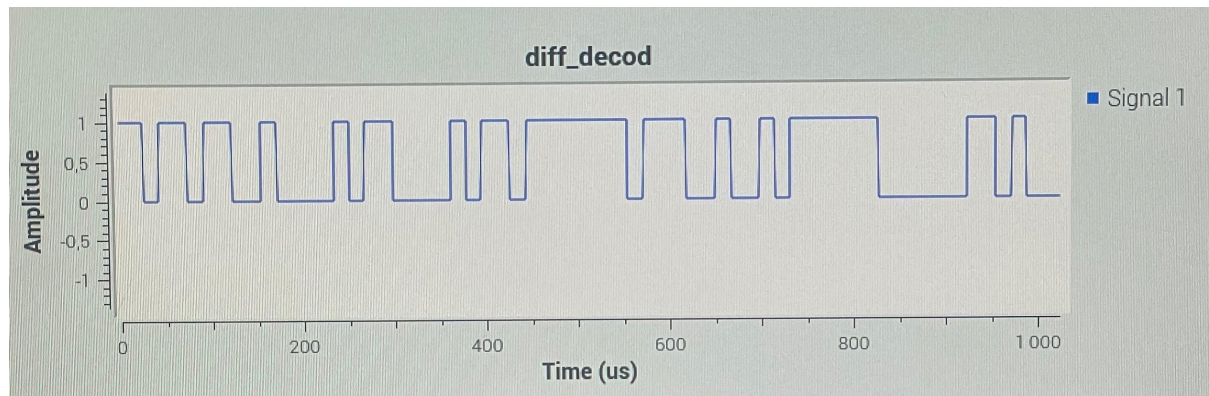


Figure 7 - Demodulated signal as output of the BPSK decoder

At the output of the BPSK decoder, we observe a binary sequence comprising alternating 0s and 1s.

II. MAC layer

The MAC (Media Access Control) layer is a crucial component in network communication, particularly in wireless contexts like a smart home wireless sensor network. It plays a central role in managing and coordinating how data packets are transmitted between devices within the network. The MAC layer's functionalities include addressing, channel access, frame validation, and error handling. It ensures reliable data transfer in environments with multiple network devices. By effectively orchestrating the interaction of periodic collectors, actuators, event-driven sensors, and emergency sensors, the MAC layer ensures orderly and efficient network communication, balancing the needs for timely data transmission and minimal interference. This balancing act is crucial for maintaining the network's overall efficiency and responsiveness, especially in a dynamic smart home environment where multiple devices with varied communication requirements coexist.

In designing the MAC layer for our project, the first critical decision was the choice of protocol to manage network access. After evaluating options, we chose Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) for several reasons. CSMA/CA was selected for its effectiveness in reducing data collisions, a common issue in dense network environments. By enabling devices to sense the channel before transmitting and delaying transmission if the channel is busy, CSMA/CA enhances network efficiency and reliability.

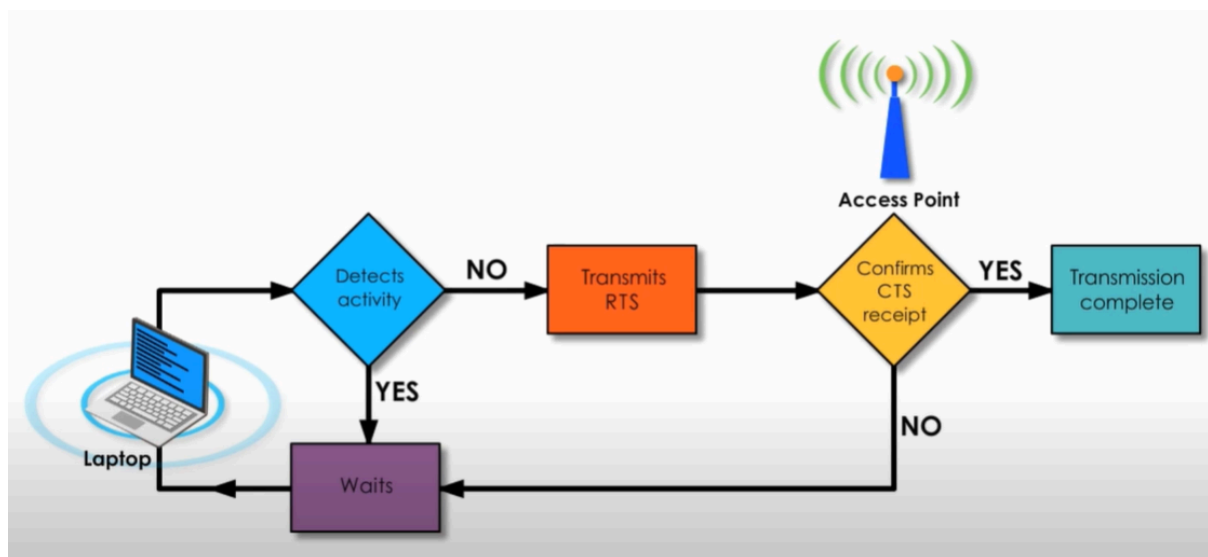
Unfortunately for us, GNU radio does not implement a block for CSMA/CA. So we had to create the block ourselves in python using the GNU Radio python api.

After each signal sampling phase, the “work” function is executed:

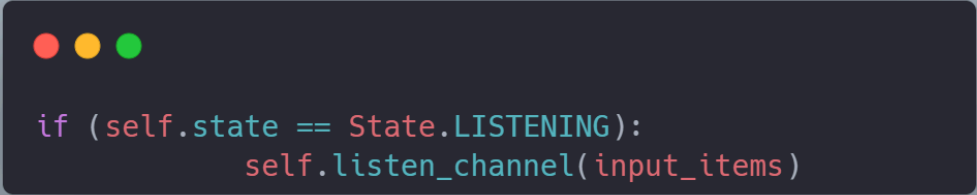
```
def work(self, input_items, output_items)
```

- input_items: samples
- output_items: samples modify

with that, we can now implement the CSMA/CA algorithm

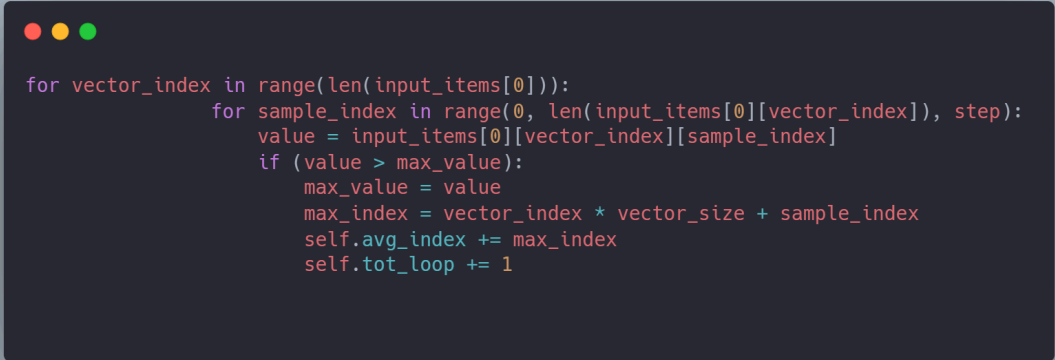


The first step is to listen the channel:



```
if (self.state == State.LISTENING):  
    self.listen_channel(input_items)
```

To do that, we do a FFT on the samples:



```
for vector_index in range(len(input_items[0])):  
    for sample_index in range(0, len(input_items[0][vector_index]), step):  
        value = input_items[0][vector_index][sample_index]  
        if (value > max_value):  
            max_value = value  
            max_index = vector_index * vector_size + sample_index  
            self.avg_index += max_index  
            self.tot_loop += 1
```

and then, we just take the maximum from the FFT and check whether the maximum is in the channel you want to transmit on.

If we detect that the channel is busy, we just wait for X ms and repeat this step.

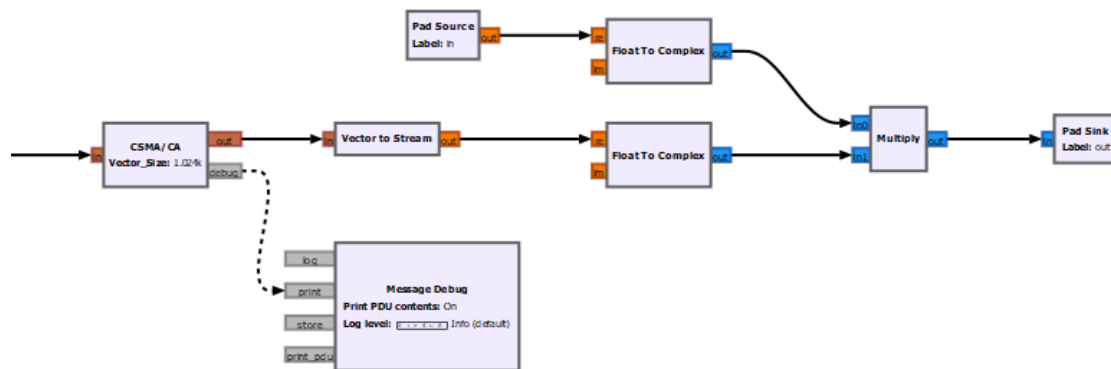
The output of this block is just a constant : 1 or 0 (1 if the channel is free 0 otherwise).

```

if (self.state != State.EMITTING):
    output_items[0][:] = 0

if (self.state == State.EMITTING):
    output_items[0][:] = 1

```



We just have to multiply the output of the CSMA/CA block with the signal to transmit or not.

For this MAC section, we have chosen to use a serial link context, i.e. encapsulating data packets, and the payload will define our application protocol.

We will therefore be doing a light implementation of a TDM protocol, without being as complex.

There are no strong real-time constraints, so a client message can reach the server after several tens of seconds.

Encoding the sensor address:

Pre required:

The sensors will have to be synchronized at each request with a frame at least 1 time per day.

The server connected to the internet will sync via NTP and send a sync every 24 hours to the client.

The maximum number of clients per channel will be 16 addresses.

Details of a typical minute:

The server will have the default address 1 and will communicate with its sensors from second -1 to 19 of each minute.

-5 seconds of network rest seconds from 20 to 24

-Seconds 25 to 55: Sensor addresses from 25 to 55 in steps of 2. 25, 27, 29 etc ...

So the sensor (client) will have as parameter the channel frequency + its Id (Src Client see below).

Example:

Sensor 1: Channel 1, address 25 will communicate with these characteristics from the 25th second of each minute.

Encoding sensor/central server frame

We are using a client-server architecture with a single server and several clients or sensors.

Coded on 5 bytes:

-Byte 1: Name: Frame size

Role: Defines the size of the protocol frame.

-Byte 2: Name: Srv Addr

Role: Destination address 1 server per installation

-Byte 3: Name: Src Client

Role: Client address: each client has a unique address aka client source

-Byte 4: Name: Val

Role: Value

-Byte 5: Name: Unit

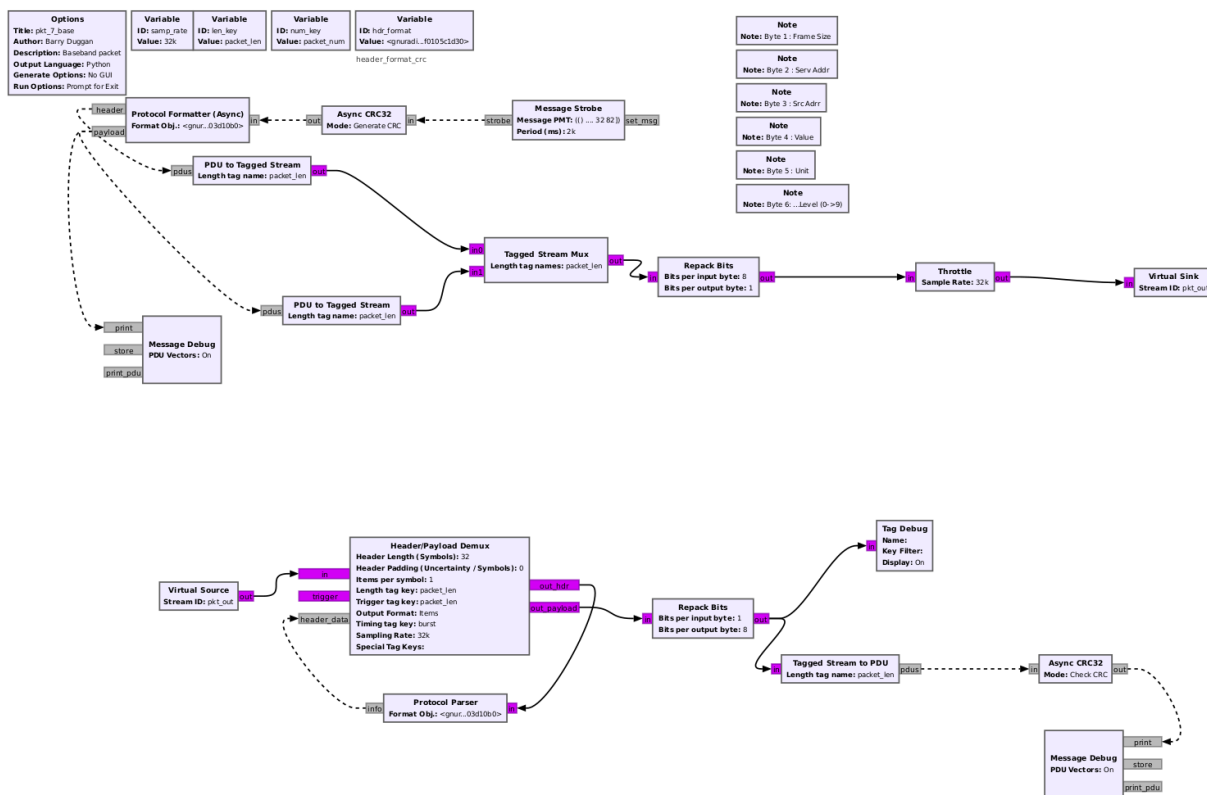
Role: Coding on one byte Unit e.g. Celcius degree

-Byte 6: Name: Level

Role: The urgency level of the message. (0 to 9) e.g. syslog.

GnuRadio Simulation

Note that this implementation only works with one sender and one receiver. You will need to duplicate and invert the simulation to have bidirectional flows.



Conclusion

In our report, we dove into setting up Wireless Sensor Networks for a smart medical home, focusing on the Physical, MAC, and Application layers. Our goal was to create a network that handles sensor nodes and actuators smoothly, while keeping security and responsiveness in mind for a smart home environment.

We tackled requirements like mobility, security, quality of service, and power consumption, making sure each layer met these needs. Opting for BPSK modulation at the Physical layer proved reliable, and our custom CSMA/CA algorithm at the MAC layer made our network more dependable.

Our application layer featured a simplified Time Division Multiplexing (TDM) protocol, allowing the central server to communicate with sensors effectively. However, it's worth noting that our team makeup had limitations. With only one specialist in the SC domain among six team members, mostly from AE or GP specialties, our approach had its challenges.

Although we ensured our implementations' integrity in GNU Radio, there's room for improvement. Despite the hurdles, our work lays the groundwork for better healthcare monitoring and advancements in smart home medical care. Ongoing research and collaboration across specialties will be key for refining smart medical home setups.