INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

WSN REPORT - SMART SUBWAY

5A ISS - 2024/2025

Par Brian Biendou
Samia Boukouiss
Cedric Chanfreau
Baptiste Henriet
Yann Jobard
Marie Brunetto
Timothé Bigot

Introduction

In urban environments, public transportation systems play a pivotal role in ensuring mobility and sustainability. Among these, the subway stands out as a vital lifeline for millions of commuters daily. However, as urban populations continue to grow, the challenges associated with overcrowding, air quality, and passenger well-being have intensified. Addressing these challenges requires innovative solutions that leverage modern technology to enhance both the safety and comfort of passengers.

This report introduces the concept of a "Smart Subway" system, focusing on the integration of advanced sensors to monitor and improve environmental conditions within subway cars.

SOMMAIRE

I - The Smart Subway concept

This report introduces the concept of a "Smart Subway" system, focusing on the integration of advanced sensors to monitor and improve environmental conditions within subway cars. The primary objectives of this initiative are:

1. Prevention of Passenger Discomfort: By detecting early signs of potential passenger distress, such as excessive heat or elevated carbon dioxide levels, proactive measures can be implemented to ensure passenger safety.
2. Real-Time Communication of Passenger Density: Providing passengers with accurate and timely information about crowd levels within subway cars to enhance travel planning and reduce congestion.

To achieve these goals, the proposed system incorporates sensors capable of monitoring:

- Temperature and Humidity: Key indicators of passenger comfort and environmental quality.
- Carbon Dioxide (CO2) Levels: A marker of air quality and occupancy density.
- Passenger Count: Essential for assessing crowd levels and managing load distribution.

By analyzing these parameters, the Smart Subway system aims to transform the commuting experience, fostering a healthier, safer, and more efficient urban transit environment.

Indeed, we have, in each wagon, three sensors. One for the temperature level, another one for the C02 level, and finally, about three cameras, to count the number of users in the wagon. On this last one, we are only considering having an entity camera that gives us the number of users.

We also consider that the environment of our application is based upon the wagon itself. In this way, we do not have mobility aspects and challenges to face off.

In this project, we are considering two different communication parts. First we have the messages that are sent from the sensors via an antenna to the subway station, and then this subway station sends those frames to a base station. We are focusing only on the first one, with a star network topology for the sensors in the subway wagons, and a point to point antenna network for the subway’s antennas, located at every station of the subway.

There are various elements to take into consideration for our project, and one important aspect among disponibility, security, quality of service, and also energy consumption, that will be reviewed all throughout this report.

II - Physical layer

1. Frequency

At the physical layer level, it is first necessary for us to define a frequency at which our system will operate. To do this, we chose a frequency within the ISM band, allowing for relatively low-cost operation while still meeting our needs. Among the various possible frequency bands, we selected 868 MHz, ranging from 863 MHz to 870 MHz [1]. This band is indeed used for several wireless sensor network applications. Considering that the size of a car is 26 meters, we therefore have a frame size of 104 meters. The 400 MHz frequency and band related does not represent a good option, as there would be too many losses. Conversely, a higher frequency would allow a lower power consumption, but would result in higher costs due to the more advanced technology required, which involves greater miniaturization [2]. We arbitrarily chose 868 MHz within the selected frequency band, with horizontal polarization, as this is the most suitable in our case.

The first limitation regarding the frequency band used for our project was that it needed to be license-free, orienting us towards the ISM bands. Then, we had to take into account the physical context of the subway, mainly its size, to know how far our signals should reach, as well as it being underground, which affects the way waves propagate.

2. Modulation

The transmission of information from the wagons to the central base is carried out in two stages. First, the wagon sends sensor data to an antenna located at each metro station. Then, this antenna forwards the information to a central base that processes the data. In this project, we are focusing only on the first step. In fact, we do not have the time to define and implement the two parts, which requires different frequencies and modulation models due to several and important differences. Now that we define in the previous part the frequency , we had to choose a modulation for the signals.

We chose BPSK modulation (Binary Phase Shift Keying), which is quite robust against interferences that we may encounter in our application. With all the users in the subway and the devices emitting at different frequency levels, robust communication is critical.The sensors and the antenna on each of the wagons are connected together via strong, shielded wires. However, incorporating security within the protocol is also essential, given the critical nature of this application. Additionally, using a protocol with a low data rate helps reduce power consumption. Even though our system is directly linked to the subway's power supply, achieving efficient energy consumption remains important.

III - Mac layer :

1. Use Case, Constraints and Addresses

As explained in the introduction, our protocol will be constrained by several factors that differentiate it from a classic static sensor network.

- Firstly, mobility wise, the gateway placed on a subway will be moving on the subway line, through the city. Considering that on his way, it will have to send data to every station it encounters and will be out of range of previous stations.
- Secondly, it will have to share its media channel with other subways and each subway train must be identifiable to enable Tisséo's monitoring system to pinpoint the origin of alarming data and apply a targeted solution without disrupting the entire network.
- Thirdly, our system is meant to be settled in Toulouse, that means we have to take in consideration the current state of the subway network, anticipate potential evolution, and learn about the practices within Tisséo's organisation.

Considering those constraints, and the fact that we decided to focus on the subway to station communication only, we considered the station being in charge of sending the data up to Tisséo's infrastructure. That meant that there would be no need for addresses that can be understood by regular network infrastructure, no MAC address, IP address needed (as a side note, there would be no need for such a large pool of big format addresses).
To begin with, we had a quick look at how many subways are used on this network. We found the number of 130 subways for both the line A and B.

First Issue : Wagon management

After a quick math, 130 subway meant we had to use 8 bits, allowing us to write 256 unique addresses with one byte. Now, for redundancy purposes, we thought that each sensor could send his data uniquely. With up to 4 wagons per subway, that would mean 4 x 3 = 12 sensor ID per subway : 4 bits. As a bonus with the last 4 bits of the second byte, we thought it would be an interesting idea to add a unique random key for the communication. Leaving us with the following address format :

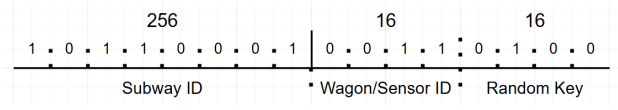


Figure 1 - Our first address format for our devices

However, that idea was cancelled thanks to M.Loubet, who underlined the fact that wagons are not exclusive to the subway they're in. Each one can be separated from the three others and be placed either in a warehouse for maintenance, or on another subway. This meant it would cause major reconfiguration to address each subway properly ! So, we gave up on the **Subway ID**.

Second Issue : Centralizing sensor or Unique sensor

Our second problem was the following question : "*Is it better to have each one of the 3 sensors of a wagon to communicate with the gateway, or to have a gateway gathering the information and sending one big data package ?* ". To answer that question, we came up with the following table to describe the pros and cons of each solution :

	1 Node	3 Sensors
Pros	- less addresses - small overhead - less channel ocupation	- cheap frame loss - redundancy of sensors
Cons	- costly frame loss - no information if node is down	- more addresses - bigger overhead

Figure 2 - Advantages and Inconvenients or single or multiple gateways for a single car

After debating on how dangerous it would be to have a gateway down and potentially no kind of information about the safety of the wagon. We concluded that even without sensors, critical levels of liveability were sensationally rare and would only occur if the reasons that caused this situation were already way more threatening than this. Ex : *Deadly oxygen level in the subway means deadly level of oxygen outside. Same goes for temperature or crowdedness*. So we settled on the fact that our product was made mostly for comfort use and alerting for abnormal situations.

Final Addresses Format

Finally, here are the choices we came up with and the other minor factors we had to take in consideration. Starting with the address, we chose to use a unique identifier for each gateway. This choice was made by taking in consideration the sensors are "black box" that gives us the information reliably. As such, the sensors are physically linked to the gateway using wire, rather than wireless communication.

As we've seen that Tisséo's network is composed of 130 subways of 2 to 4 wagons, we calculated that :

130 x 4 = 520; Taking the maximums, there are 130 subways of 4 wagons
520 + 40 = 560: We add the 40 stations of line A and B
512 < 560 < 1024;

We would need 10 bits to create all of our addresses. (this format also allows for the potential arrival of the line C, which would add 20 stations, and around 250 wagons.)
The standard address of 10 bits was, as you may have noticed, conceived with the thought that both the stations and wagons would share the same address pool. This choice was made so we could have a single MAC Frame Format, allowing for back and forth exchanges.

2. Sequence Diagram

Data transmission

With the addresses issue out of the way, we could focus on designing the typical data transmission process by answering “What if” and “How do” questions :

- **How does the gateway knows when to send data :**

To avoid having the gateway spamming data emission while being between two stations, we chose that the gateway would send data when he would be invited to. As so, the Station will be the one sending requests of data every 10 seconds.

- **How does the gateway and the station know who to communicate with :**

The station can’t guess which wagon is currently stopped inside of it. To initiate the communication with the wagons, the station is going to send some discovery request in broadcast. Those discovery broadcast frame will contain the address of the station as a data and a null destination address. The gateway will send data to the last discovery address it received.

- **What if the Data has not been received :**

When the gateway sends the data, it starts a timer to wait for an Acknowledgement. If no ACK arrives before the timer ends, it sends another Data packet to the same Station address until he receives an ACK.

Typical data transmission scenario

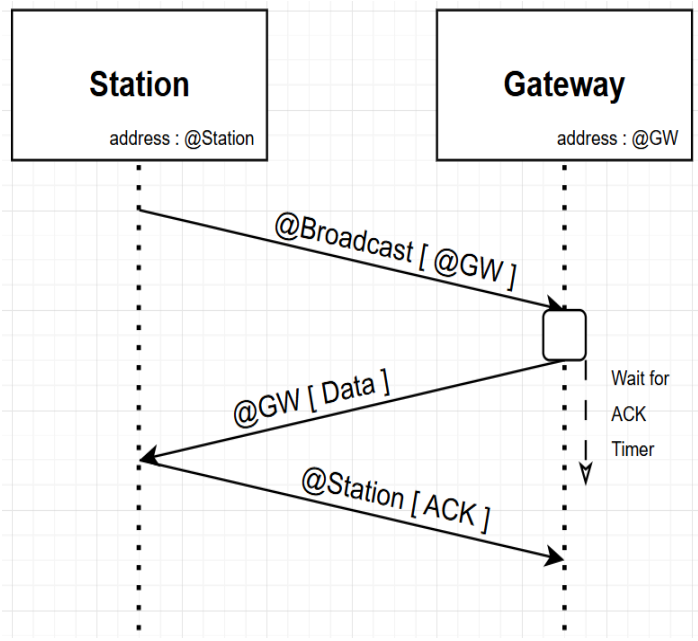
The typical data transmission scenario consists of the following steps :

1) The Station sends a discovery every 10 seconds

2) The Gateway sends the data to the Station and starts a timer for ACK

3) The Station receives the Data and send an ACK

4) The Gateway receives the ACK and stops its timer



3. MAC Frame Format

Now that we have explained our use cases, let's have a look at the format of the message between the Gateway and the Station's receiver. We tried to make it as compact as possible as it will be sent several times while the subway is waiting at the station, and as there can be up to 7 other gateway that want to send their data too.

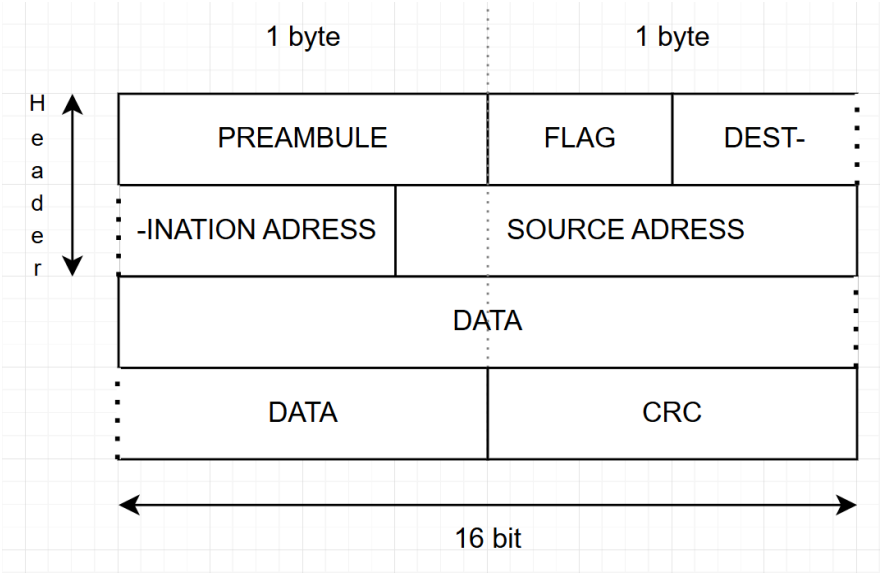


Figure 3 - Final header for our frames

Here is an explanation of our fields :

- **Preambule**: A 8-bit long field containing alternating ones and zeros. It allows the receivers to synchronize their clock at the bit-level with the transmitter.
- **Flag**: A 4-bit long field indicating what kind of message is contained. The Gateways only reads Discovery and ACK messages and ignores Payload ones.
    - 1111 : **DISCOVERY** frame, only sent in broadcast by the station. It contains the address of the stations in the field data.
    - 0011: **PAYLOAD** frame, only sent by the gateway, to the address received in the discovery.
    - 1100: **ACK** frame, only sent by the station, to acknowledge the reception of a payload frame
- **Destination Address**: A 8-bit long field for the destination device address. It is filled with ones if the frame is a Discovery.
- **Source Address**: A 8-bit long field for the source device address.
- **Data**: A 4-byte long field for storing the sensor's data. If the packet's flag is ACK, the data field is ignored and typically filled with zeros.
- **CRC**: An 8-bit long field containing the Cyclic Redundancy Check checksum result. This value is verified at the end of packet reception, and if the check fails, no ACK packet is sent in response.

4. Medium Access Control

To manage the access to the communication medium in our Smart Subway project, we chose to implement the CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) protocol including RTS/CTS (Request to Send/Clear to Send) mechanism. Indeed this MAC protocol is well suited for our project due to some reasons:

Collision Avoidance
CSMA/CA reduces the likelihood of collisions by requiring devices to listen to the channel before transmitting. The RTS/CTS mechanism minimizes collisions by reserving the channel for the sender, ensuring that other devices are aware of the transmission.

Dynamic Access
Unlike TDMA, which requires precise time synchronization, CSMA/CA allows devices to access the channel dynamically. This is a good advantage for our environment, where the number of sensors vary and don’t need to wait for a specific time slot.

Efficient Use of Bandwidth
CSMA/CA allows for a proper use of the available bandwidth by reducing the number of collisions and retransmissions.

Energy Efficiency
Devices can enter a low-power sleep mode when they are not actively transmitting or receiving data. It will help to conserve energy and extend the battery life of the devices.

Scalability
CSMA/CA is scalable and can accommodate a varying number of devices. This is important in our scenario, where the number of sensors and subways may change over time.

5. Clock synchronization

We chose CSMA/CA and so clock synchronization is not essential, at least not as much as with TDMA. In our case, the preamble can fulfill this function and allows the receivers to synchronize their clocks at the bit level with the transmitter. Additionally, the preamble can serve as a frame start signal and help differentiate it from noise, for instance.

IV - Implementation

Once our system was well thought out, we were able to model it, as it would help us to notice potential flaws in our system and make some more visual explanations of our ideas.

1. GNU Radio Companion

The first tool that we wanted to use was GNU Radio Companion, which is a software designed for software-defined radios and signal processing systems. In our case, it would mainly be helpful for the physical part of our system, with the MAC part being implemented in python. GNU Radio Companion uses visual programming, where classes, variables and functions are seen as blocks, the latest having input and output of defined type. This model gives us an easy way to implement our system. Using the BPSK modulation, and simulating the reception of the frame by a written file we got the code as shown in figure 4:

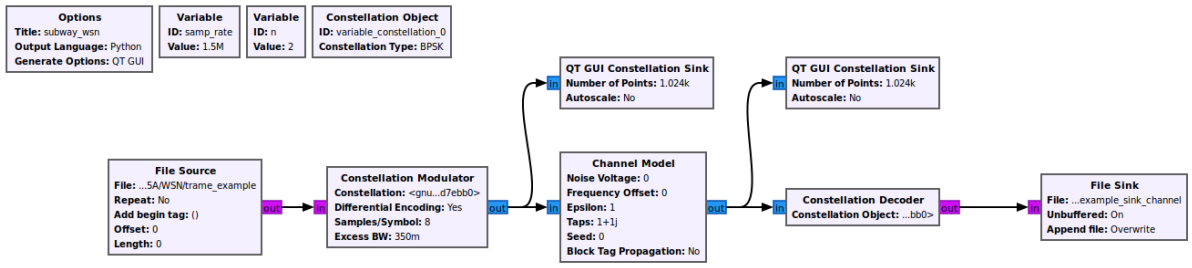


Figure 4 - Simulation in GNU radio of the physical layer

However, we had a lot of limitations using GNU Radio, between the deprecated or non-functional blocks or the lack of updated documentation. We learned through tests that the File sink block was not functional, which was a problem as we did not find another way to display or save a received frame that has been demodulated. After some trial and error, we decided that GNU Radio was not the best tool for our project.

2. Python

As the MAC layer was written in Python, we decided to implement the physical layer and all implementations using this language, as it offers a lot of documentation and libraries.

First, the MAC layer was developed using three classes: Frame, Sensor and TDMAMAC.

- **Frame** would represent the frame with all the corresponding information: start frame delimiter, flag, addresses, data and crc. An heredated class has also been created to split the data into temperature, air quality and person count.
- The second class is the **Sensor**, which will randomly pick a realistic value within a predefined range.

- Finally, **TDMAMAC** is a class written to simulate a TDMA behaviour.

Then, we created the physical layer. For this one, we used one class: **Signal**, as well as functions:

- **modulate_bpsk** and **demodulate_bpsk** that turns frame into signal or signal into frame. For the second one, it takes the closest value in case of noise.
- **show_constellation_diagram**, that can show the constellation diagram of a given signal.
- **simulate_canal**, that will add random noise to a signal, using a given strength. This can be useful to challenge the crc of our frame

Once both were implemented, we were able to run a simulation using a python script called simulation_send_receive.py. If you want to run this script, it is available on this git repository.

3. Results

Finally, we were able to test out our system by simulating the layers altogether. We decided to run multiple tests with multiple values for the data to ensure that every case would be working. We will show two runs of our program in this section: one normal run, and one with extra noise.

Control simulation

For the first one, we runned the program with the base noise. As you can see in the constellation diagrams, we can clearly distingate the two states of a BPSK modulation, with a phase of 0° and 180°. As expected, the program was able to properly demodulate and read the received frame.

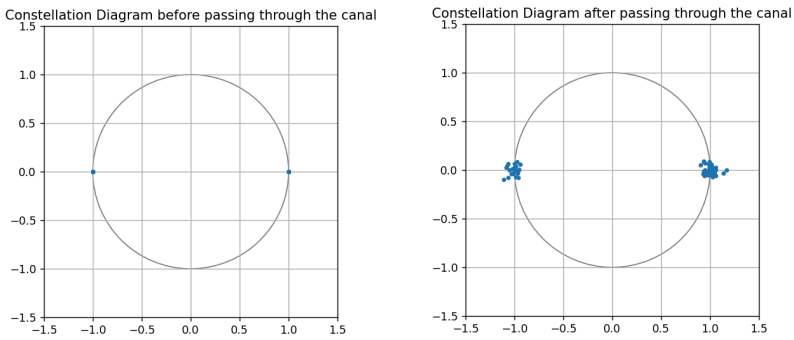


Figure 5 - Constellation diagrams in a control simulation

```
New frame:
---frame---
sfd: 165,
```

```
flag: 1,
destination address: 12,
source address: 15,
crc: 15,
number of persons: 45,
temperature: -8,
air quality: 1,
binary:
000011110011000000010000101001011111101100000001111110000010110 1
-----------
Modulation of the signal

Passing through the canal, getting noised

Demodulation of the signal

Received frame:
---frame---
sfd: 165,
flag: 1,
destination address: 12,
source address: 15,
crc: 15,
number of persons: 45,
temperature: -8,
air quality: 1,
binary:
000011110011000000010000101001011111101100000001111110000010110 1
-----------
```

Figure 6 - Output of our control simulation

Noisy simulation

Our second run was done using the simulation of a noisy medium. The noise is randomly added, but this allows us to show the value of the crc field. You can test this configuration by running the command:

```
python simulation_send_receive.py 0.5
```

Figure 7 - Command to run a noise value of 0.5, control value being 0.05

You can see in the following output that the crc remained the same, as the source address changed, showing the corruption of the data. We can also see that BPSK is powerful, as it is the only error in our code, while the constellation diagram clearly shows chaos in the value of the signal.

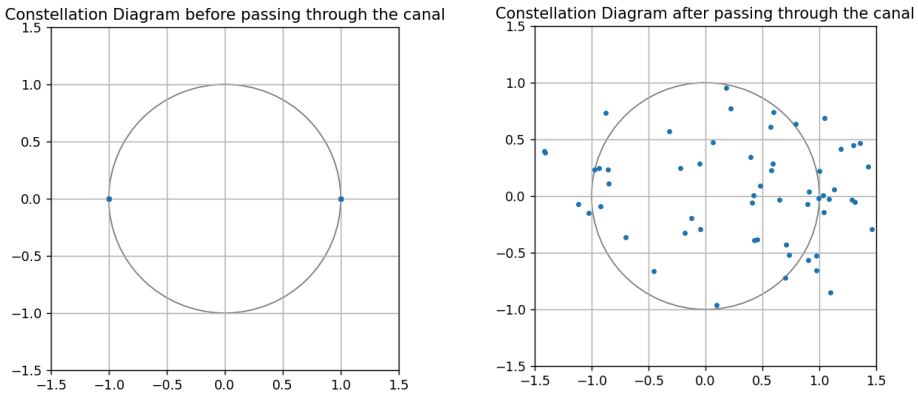


Figure 8 - Constellation diagrams in a noisy simulation

```
New frame:
---frame---
sfd: 165,
flag: 1,
destination address: 12,
source address: 15,
crc: 6,
number of persons: 40,
temperature: 7,
air quality: 6,
binary:
0000111100110000000100001010010100000110000001100000011100101000
-----------
Modulation of the signal

Passing through the canal, getting noised

Demodulation of the signal

Received frame:
---frame---
sfd: 165,
flag: 1,
destination address: 12,
source address: 11,
crc: 6,
number of persons: 40,
temperature: 7,
air quality: 6,
binary:
0000101100110000000100001010010100000110000001100000011100101000
-----------
Frame corrupted. Current crc: 6, expected: 2
```

Figure 9 - Output of our noisy simulation

Conclusion

Through this lab, we were able to brainstorm and conceive the physical and mac layer for an hypothetical application. This exercise was very interesting to apply our knowledge about telecommunication and sensor networks. It also showed us the complexity of conceiving a system both efficient and scalable.

Bibliographie

[1] Zhou et al, “Measurement of RF Propagation in Tunnels”, 2013 IEEE Antennas and Propagation Society International Symposium
https://www.researchgate.net/publication/262009244_Measurement_of_RF_propagation_in_tunnels

[2] A. Maassen, T. Sirvent, and R. G. Petroccia, "Impact of Clock Frequency on the Performance and Consumption of a Secure IoT Node," ResearchGate, Sep. 2018. [Online].
https://www.researchgate.net/publication/327546447_Impact_of_Clock_Frequency_on_the_Performance_and_Consumption_of_a_Secure_IoT_Node