# AMS Sonore

Axel Ava

Sergio Guarino

Marc Savioz

*ISC - 6ème Semestre (Soir) - HEPIA*

[**AMS Sonore**](#_hj2w1yfl91nn) **1**

[But du projet](#_qnuy9kvspuie) 3

[Matériel](#_ra5e6vxo41vb) 4

[Solution développée](#_k65h7wj5fwyl) 4

[Sécurité](#_58k205n98d64) 4

[Web](#_d9leyqjjqzv2) 4

[Identification par nom d’utilisateur/mot de passe](#_gfc7ogzco2et) 4

[Authentificateur: google authenticator](#_o7yukq7ng05d) 5

[Telecom](#_ie7mczpb8zpk) 6

[Communication](#_ni9kj2kev3a5) 7

[LoRa](#_datrwzglyit) 7

[Format des paquet](#_t5inzy24n5mi) 8

[Calcul “airtime”](#_9q6z9poxbydt) 9

[Formatage des paquets](#_2skse0blq0zx) 10

[Encodage “string”/json](#_owk09oh21823) 10

[Encodage binaire](#_ezopqhwkqvg3) 11

[Limitations](#_x7a981m3q4ui) 12

[GPS](#_rew5hxepe6qa) 12

[Microphone](#_bzuslajeux6g) 13

[Problèmes rencontrés](#_eszb03iafxps) 15

[Conclusions](#_z7zqdlvakz42) 16

# 

## But du projet

Dans le cadre du projet de semestre de la formation ISC de l’HEPIA, une proposition de solution pour la détection de la pollution sonore a été effectuée par les élèves Axel Ava, Sergio Guarino, et Marc Savioz.

Le but étant de pouvoir afficher sur une carte les lieux où le niveau de bruit serait supérieur à un seuil fixé. Ceci s’est fait en mesurant le niveau de bruit tout en conservant les avantages de la mobilité, permettant ainsi de pouvoir éventuellement placer les capteurs de manière dynamique à différents endroits en différents lieux.

Afin d’y parvenir, des capteurs (microphones) équipés de modules GPS et LoRa sont utilisés. Ces modules présentent les avantages suivants:

* Consommation énergétique faible.
* Grande portée
* Bonne couverture dans la région du canton de Genève

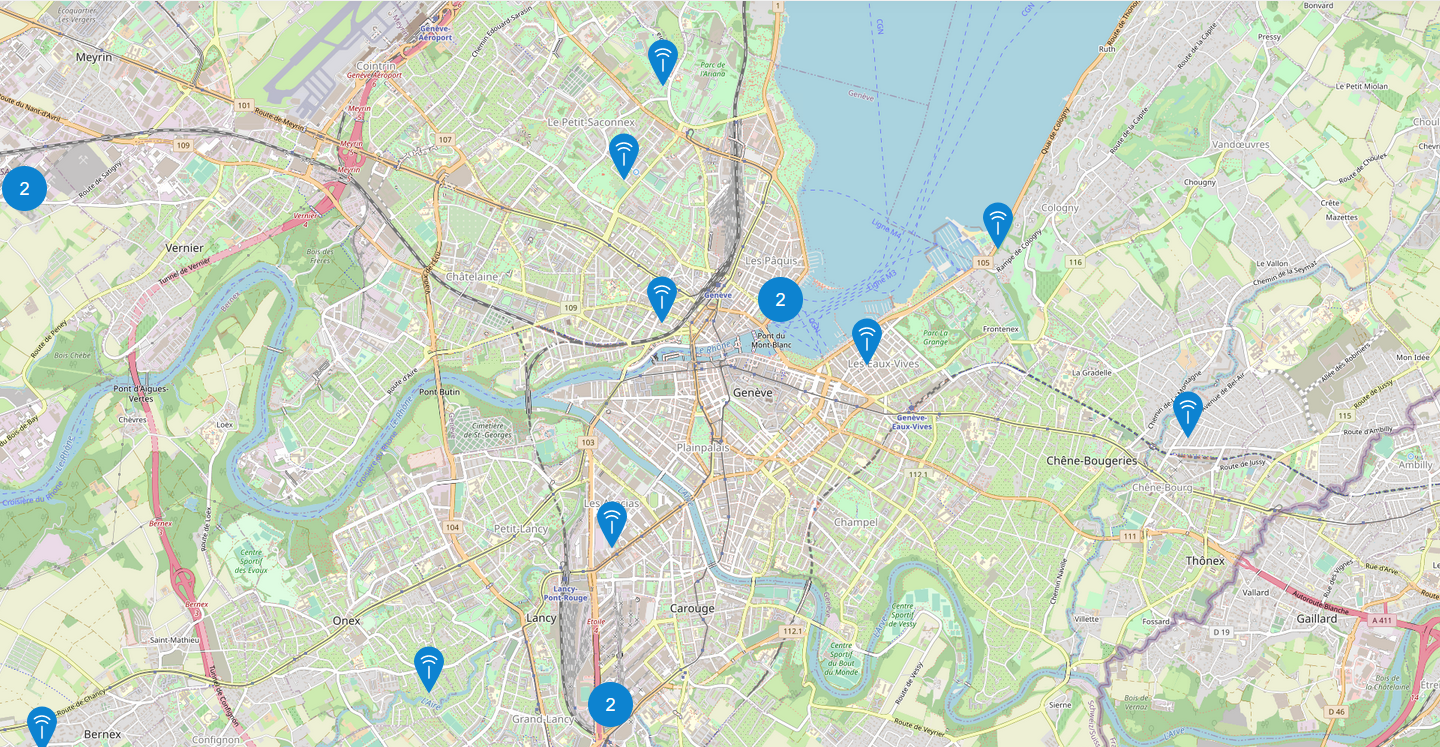


Figure 1: Carte des gateways LoRa en ligne dans la région genevoise.

L’idée est de pouvoir déployer ces capteurs (aux abords d’une route par exemple) pendant une durée déterminée, puis de pouvoir les déployer facilement sur un autre site de mesure.

## Matériel

Pour développer cette solution, les modules électroniques suivants ont été utilisés pour le prototypage de cette solution.

* Raspberry Pi 4 avec OS Raspbian (Raspberry)
* Waspmote avec LoRa et GPS (Libelium)
* LoRa / GPS HAT pour Raspberry (SeeedStudio)
* Microphone MAX9814 (Adafruit)
* Convertisseur ADC ADS1015 (Sparkfun)
* carte d’adaptation pour interfaçage composants (Sparkfun)

## Solution développée

### Sécurité

Nous avons exploré principalement deux domaines de sécurité, notamment la sécurité web et la sécurité en télécommunication (dans le protocole LoRa).

#### Web

Nous avons sécurisé l’accès à la carte en mettant en place un système d'authentification à deux facteurs(A2F).

Authentification à double facteur est une méthode d'authentification forte qui permet à un utilisateur d’accéder à une ressource informatique uniquement après avoir présenté deux preuves d'identité distinctes. Les deux preuves sont en général une chose possédée et une chose sue.

Dans le cadre de ce projet, nous avons implémenté un système dans lequel l’utilisateur doit premièrement insérer son nom d’utilisateur et son mot de passe, une fois cette information vérifiée celui-ci doit insérer un code qui est affiché sur l’application google authenticator.

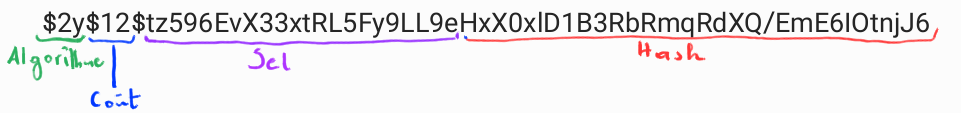
##### Identification par nom d’utilisateur/mot de passe

Les informations de connexion sont stockées dans une base de données normalement inaccessible à l’utilisateur. Ici la base de données a été créée via l’application web phpmyadmin, application disponible sur le serveur web Wamp. Le nom d’utilisateur y est stocké en clair tandis que le mot de passe a été chiffré.



Le mot de passe a été hashé avec la fonction password\_hash de PHP. Cette fonction de hachage utilise l’algorithme de hachage bcrypt qui est fort et irréversible.

Bcrypt a été créée en 1999, c’est un algorithme jugé robuste encore aujourd’hui, il se veut résilient aux attaques par CPU & GPU en intégrant un sel ainsi qu’un coût (nombre d’itérations pour créer le hash) nativement. Une chaîne de caractères hashée en bcrypt se présente comme suit:



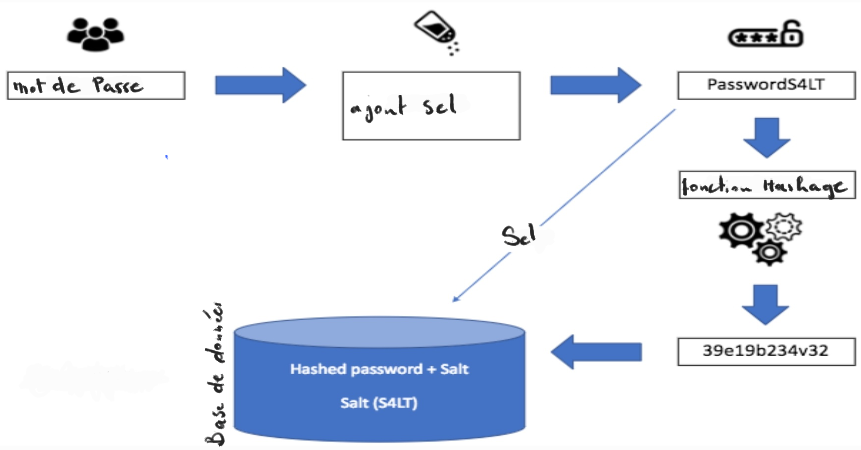
$2y: L'identifiant de l'algorithme de hachage (bcrypt)

$12: Coût ( ==> 496 tours)

tz596EvX33xtRL5Fy9LL9e: 16-bytes (128-bit) de sel, codé en base64 à 22 caractères

HxX0xlD1B3RbRmqRdXQ/EmE6IOtnjJ6: 24-bytes (192-bit) hash, codé en base64 à 31 characters.

L’idée du salage est d’ajouter au mot de passe de l’utilisateur une chaîne aléatoire (le sel) qui va compliquer l’utilisation des rainbow tables sur le hash en multipliant le nombre de hash possibles pour un même mot de passe.



L’algorithme Bcrypt se différencie des autres algorithmes tels que MD5, SHA-1, SHA-256, RIPEMD-160 par le fait que c’est une fonction de hachage qui se repose sur l’algorithme Blowfish. Elle est conçue de façon à pouvoir être ralentie lorsqu’on augmente le nombre de passages(coût) dans la fonction.

##### Authentificateur: google authenticator

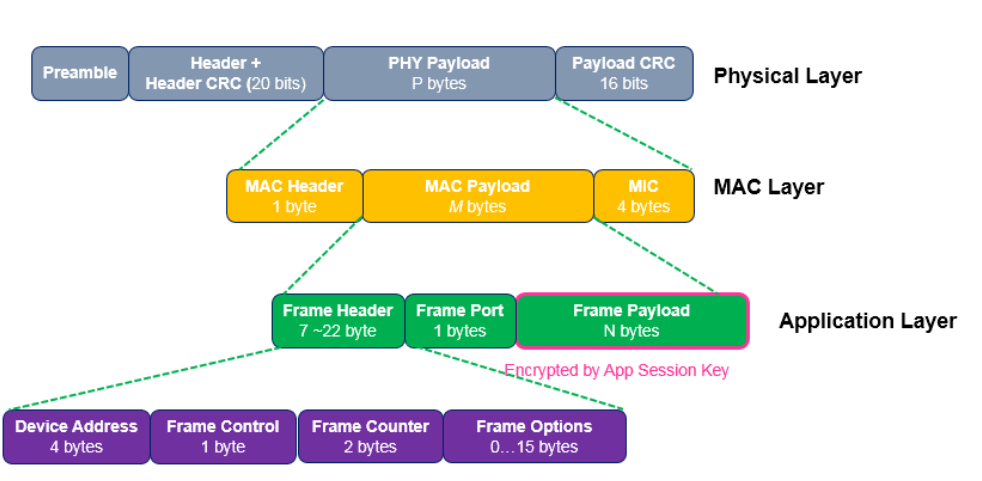
L’authentification à double facteurs requiert un deuxième élément pour prouver son identité. Nous avons donc choisi d’implémenter le deuxième facteur avec l’application google authentification.

Le principe de fonctionnement de l’application est le suivant. L’application crée un code éphémère calculé depuis une clef numérique propre à l'utilisateur. Lors de la première utilisation, une clef numérique secrète de 80 bits unique pour chaque utilisateur est générée. Cette clef est transmise sous forme de QR code. L'application calcule à chaque connexion une signature numérique HMAC-SHA1 à partir de cette clef, en codant le nombre de périodes de 30 secondes écoulées depuis l'« epoch » Unix. Une partie de cette signature est prélevée et convertie en un nombre à 6 chiffres affiché par l'application et l'utilisateur doit recopier sur le site.

#### Telecom

Tout d’abord, il faut savoir que le protocole LoRa est organisé en couches.

* La couche physique, qui s’occupe simplement de la transmission des bits
* La couche MAC
* La couche Application

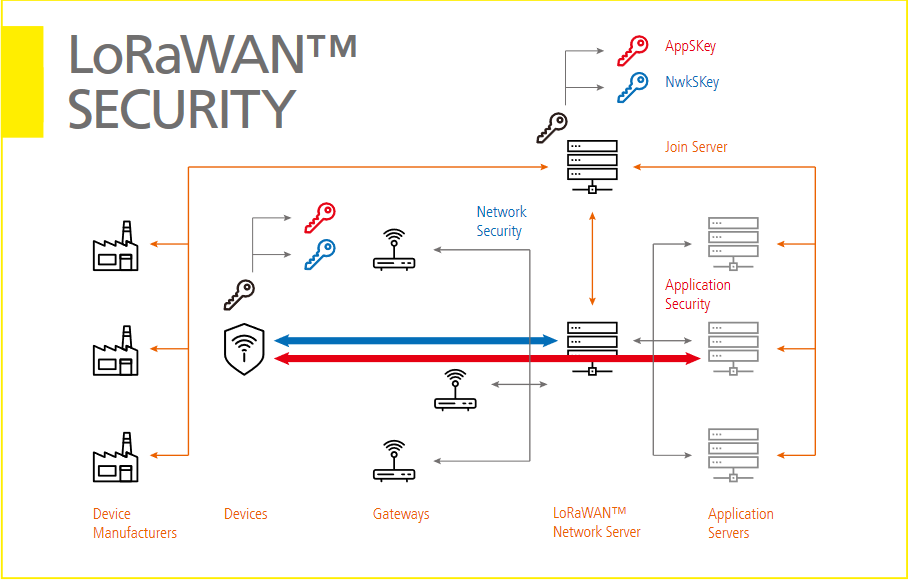


La sécurité implémentée dans le protocole LoRa est double:

* Le payload de la couche MAC est chiffré par une clé nommée NwkSKey.
* Le payload de la couche Application est chiffré par une clé nommée AppSKey.

Lors du désempilement des couches par le serveur de réseau, le payload MAC est déchiffré. Révélant ainsi le contenu de la couche Application. Or, le payload de cette couche est toujours chiffré à ce moment, donc une personne attaquant ce serveur n’aura pas accès aux données transmises.

Ensuite, le paquet de la couche application est envoyé à un serveur d’application, ou il est déchiffré avec la clé AppSKey, permettant au développeur d’accéder à ses données.



### Communication

Le schéma suivant présente, de manière globale, le système développé pour ce projet.

Tout d’abord, un microphone connecté via un ADC (convertisseur analogique numérique) à la carte de prototypage (Waspmote ou Raspberry Pi) envoie des données audio.

Ces données sont quantifiées et classées, pour des raisons de simplification, en trois catégories: HIGH, MED, LOW.

Une mesure des valeurs sur 10 minutes[[1]](#footnote-0) est effectuée, avec les valeurs maximales, minimales et moyennes sur ces 10 minutes. De plus, la position GPS du module est obtenue.

Une fois les 10 minutes écoulées, un paquet contenant ces informations est envoyé sur le réseau LoRaWAN en utilisant la plateforme The Things Network.

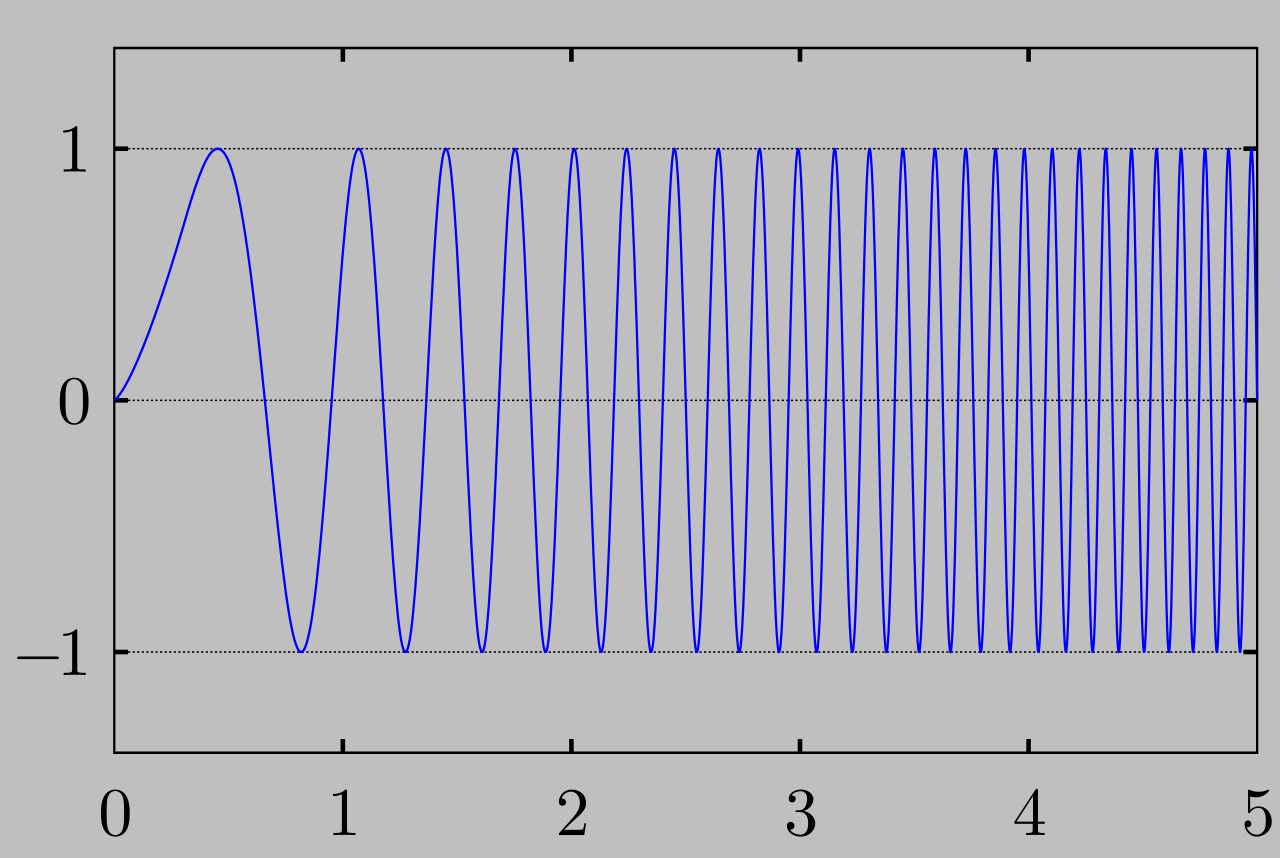
Ces données sont ensuite récupérées par un serveur web afin de les afficher sur une carte permettant ainsi une visualisation concrète des données récoltées.



#### LoRa

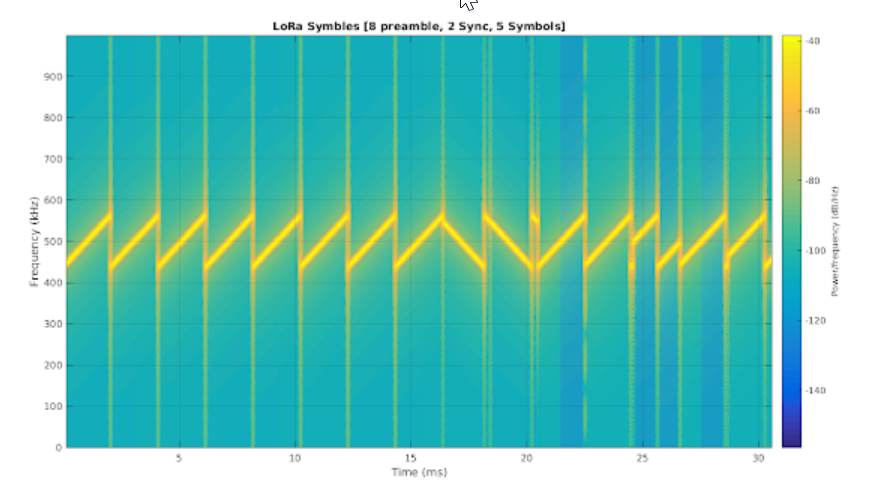
Le LoRa (Long Range) est un protocole de communication radio. La couche physique de ce protocole utilise une variante propriétaire de modulation Chirp Spread Spectrum.

Dans ce type de modulation, le signal est étalé dans le spectre, la bande passante devenant plus large tout en effectuant un balayage des fréquences.



Exemple d’une fréquence “chirpée” vers le haut de manière linéaire avec le temps.

Image [Creative Commons](https://en.wikipedia.org/wiki/en:Creative_Commons) [Attribution-Share Alike 3.0 Unported](https://creativecommons.org/licenses/by-sa/3.0/deed.en) by [Georg-Johann](https://commons.wikimedia.org/wiki/User:Georg-Johann)



Analyse spectrale d’un signal LoRa - <https://www.sghoslya.com/p/lora-is-chirp-spread-spectrum.html>

Ce type de modulation présente les avantages suivants[[2]](#footnote-1):

* Résistant à l’effet Doppler (utile p.ex. sur les satellites LEO)
* Robuste face aux perturbations
* Consommation faible d'énergie
* Résistant au multipath

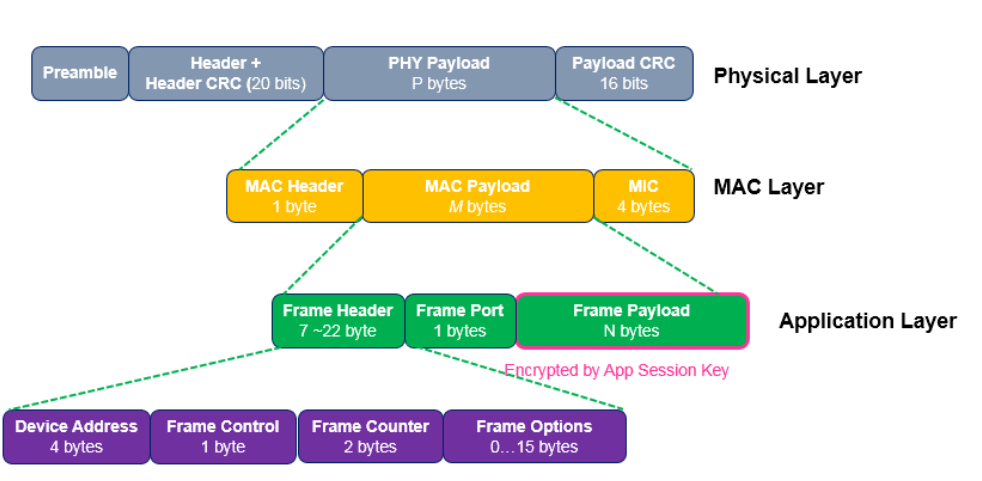
##### Format des paquet

Afin de ne pas consommer trop de bande passante, une réflexion est nécessaire sur le formatage des paquets afin d’être le plus efficace possible.

En effet, les valeurs suivantes doivent être transmises:

* La position GPS (latitude, longitude)
* Le niveau sonore (min, max, moyenne) - sur 5 minutes

Une trame LoRa est constituée de la manière suivante, ce qui veut dire qu’en plus du payload, de l’overhead à cause des différents headers (min. 12 bytes, 13 si le paquet contient des données dans le payload) est présent et doit être pris en compte pour le calcul de l’airtime:



##### Calcul “airtime”

On peut déterminer le temps consommé par notre trame avec l’outil suivant:

[https://avbentem.github.io/airtime-calculator/ttn/](https://avbentem.github.io/airtime-calculator/ttn)

La plateforme waspmote et sa librairie LoRa permettent de choisir le data rate en utilisant la fonction setDataRate(x)

Pour notre prototype, nous avons d’abord utilisé le datarate index 0, car il permet d’avoir la plus longue portée. En contrepartie, le débit est faible et l’airtime est long.

Après vérification, un datarate de 3 est utilisé car, comme décrit plus bas, le datarate 0 n’est pas suffisant pour envoyer autant de messages que souhaité tout en suivant les réglementations en vigueur.

Comme nous tentons d’avoir un système “edge”, les calculs de niveau sonore, de position etc sont effectués directement dans le end-device, permettant ainsi de ne pas utiliser trop de bande passante.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data Rate Index** | **Spreading factor** | **Bande passante** | **Portée (indicatif)** | **Débit (indicatif) [bps]** |
| 0 | 12 | 125 kHz | +++ | 250 |
| 1 | 11 | 125 kHz | ++ | 440 |
| 2 | 10 | 125 kHz | + | 980 |
| 3 | 9 | 125 kHz | - | 1760 |
| 4 | 8 | 125 kHz | -- | 3125 |
| 5 | 7 | 125 kHz | --- | 5470 |

Ainsi, le “airtime” peut être simplement calculé grâce à la formule

t[s] = data[[3]](#footnote-2) [bits] / data rate [bits/s]

#### Formatage des paquets

##### Encodage “string”/json

Donc, si ces informations sont transmises sous forme d’une chaîne de caractères, beaucoup de temps serait perdu. Par exemple, un json:

{

"GPS": [46.20944, 6.13564],

"Noise": [10, 20, 15]

}

Ce type d’encodage utiliserait donc 66 bytes + overhead = 79 bytes.

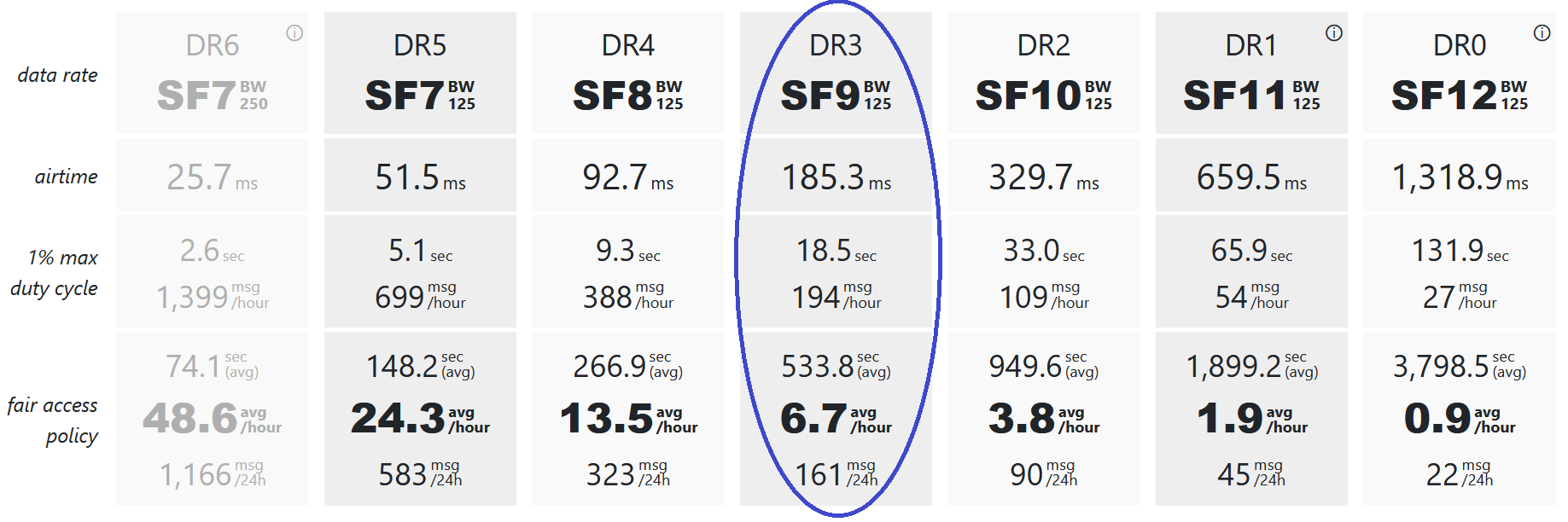
**Duty cycle**

Un tel paquet correspond à un airtime de 3,285.0​ms, ce qui signifie que, pour un duty cycle de 1%, une pause entre les messages de 328.5​ s. minimum doit être effectuée. Cela correspond à 10 messages / h maximum.

**Fair access TTN**

De plus, la politique de *fair access* du réseau The Things Network permet uniquement un airtime total par appareil, par jour, de 30 secondes. Cela veut dire que l’on serait limité à 9 messages/jour tout au plus.

De plus, les restrictions ne permettent pas d’envoyer plus de 51 bytes dans la bande européenne lors de l’utilisation d’un SF de 11 ou 12 comme nous avons choisi.



**Il faut donc une autre solution !**

##### Encodage binaire

Afin d’être plus efficace, une méthode d’encodage doit être mise en place (i.e. envoyer les valeurs binaires directement)

Le payload binaire est formaté selon le schéma suivant:

Le but de la première itération était d’utiliser un seul byte afin d’économiser un maximum de temps dans les airs. Cette version ne permet pas une assez grande précision dans la position GPS (2^6 = 64 valeurs possibles seulement - 32 pour chaque valeur - respectivement la latitude et la longitude). Une autre solution est nécessaire.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **AMS Sonore Packet V0.1** | | | | | | | |
| **Bits** | **Bit 0** | **Bit 1** | **Bit 2** | **Bit 3** | **Bit 4** | **Bit 5** | **Bit 6** | **Bit 7** |
|  | **Type** | | **Payload** | | | | | |
| **Sound** | 0 | 0 | Min | | Max | | Avg | |
| **GPS** | 1 | 1 | Lat | | | Lng | | |
| **N/A** | 0 | 1 | x | x | x | x | x | x |
| **N/A** | 1 | 0 | x | x | x | x | x | x |

Afin d’apporter plus de précision, imaginons que nous souhaitons en plus des degrés, plusieurs décimales après la virgule. Un float32 peut suffire à cette tâche (4 bytes).

Le format suivant est donc développé:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **AMS Sonore Packet V0.2** | | | | |
| **Bytes** | **0** | **1** | **2** | **3** | **4** |
|  | **Type** | **Payload** | | | |
| **Sound** | 0x50 | Max noise level  0x00 - 0xFF | Min noise level  0x00 - 0xFF | Avg noise level  0x00 - 0xFF | Timespan in minutes  0x00 - 0xFF |
| **Latitude** | 0x7A | float32 latitude | | | |
| **Longitude** | 0x70 | float32 longitude | | | |
| **N/A** | \* | \* | \* | \* | \* |

Cela permet également de laisser place à d'éventuels autres types de paquets, pour d’autres capteurs par exemple. (255 types de paquets possibles).

***Note****: Pour des raisons de temps ce format n’a pas été implémenté dans le prototype mais pourrait l’être relativement facilement.*

**Duty cycle**

Un tel paquet (5 byte de payload + 13 d’overhead = 18 bytes) équivaut à un airtime de 1318.9 ms avec un datarate index de 0.

Avec un duty cycle horaire de 1%, cela correspond à une attente nécessaire de 131.9 secondes entre chaque message.

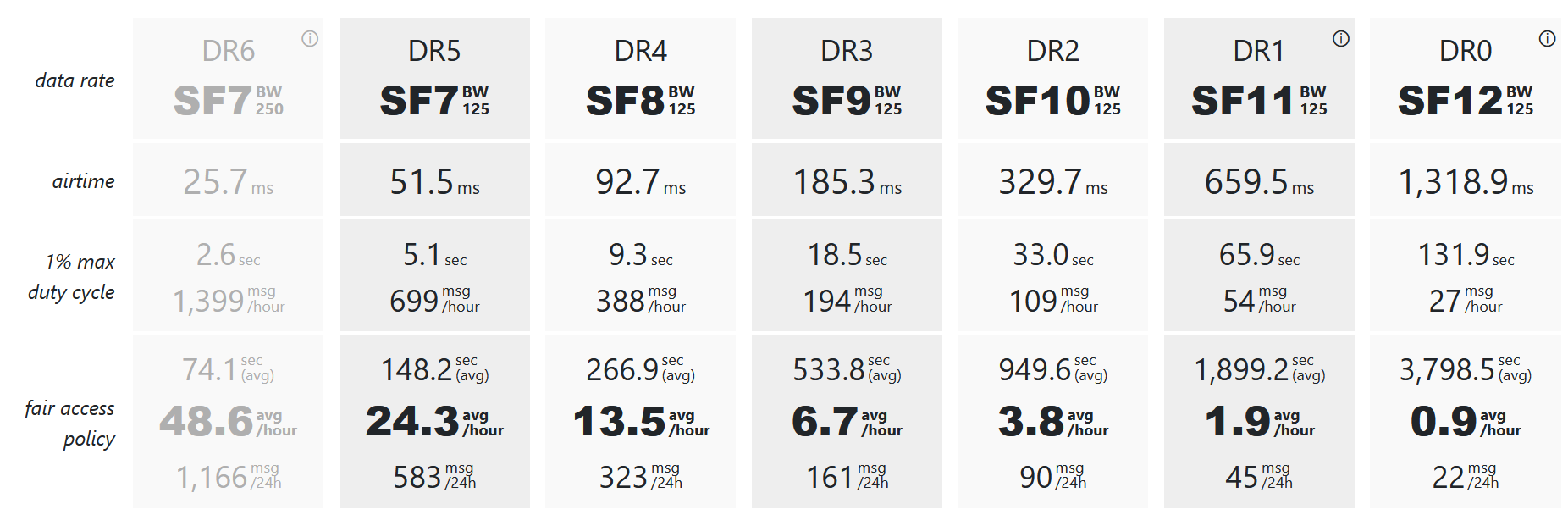
Cela veut dire que 27 messages peuvent être envoyés par heure, à concurrence de la politique fair access ci-dessous.

**Fair access TTN (30 sec airtime / device / 24h)**

La politique fair access TTN limite la capacité de transmission à 0.9 messages par heure et 22 messages par jour si on utilise le datarate 0. Cela ne suffit clairement pas pour notre application.

Une décision est donc prise de régler le datarate index à 3 afin de pouvoir envoyer jusqu’à 161 messages par jour, (6.7 messages par heure en moyenne sur 24 heures).

**Conclusion**: Utiliser un data rate index de 3, avec des paquets de 18 bytes, permet d’avoir une cadence d'échantillonnage assez précise d’environ 10 minutes.



##### Limitations

Les régulations nationales, internationales ainsi que celles du réseau utilisé (The Things Network) définissent un *duty cycle maximal.* Cela signifie qu’il n’est autorisé de transmettre qu’avec un *taux maximal* temporel afin de ne pas encombrer le spectre et bloquer d’autres utilisateurs.

Les détails sont disponibles sur <https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/>

#### GPS

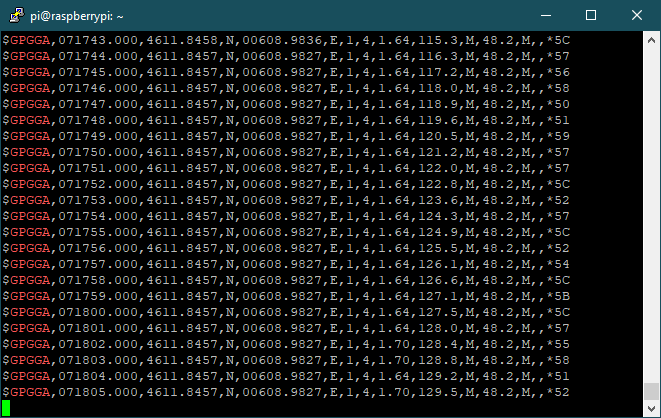
Les cartes LoRa utilisées intègrent également un module GPS (Global Positioning System), qui peut être utilisé pour déterminer la position du système.

Le format des données reçues du GPS se base sur la norme établie par la NMEA (National Marine Electronics Association), qui définit une série de messages différents que les satellites transmettent.

La description du principe de fonctionnement du GPS est hors du cadre de ce rapport, mais il suffit de dire que ce système est composé de 2 parties : un récepteur de signal (la carte utilisée dans le projet) et des satellites (31 au moment de l’écriture de ce rapport). Pour connaître la position sur Terre du récepteur, il faut qu’au moins 3 satellites soient à la portée du récepteur.

Les messages envoyés par les satellites peuvent contenir des informations concernant la position du récepteur (latitude, longitude, altitude, vitesse), mais également celles concernant les satellites eux-mêmes (la quantité de satellites en vue, leur altitude, l’azimut). Une liste complète de différents types de messages peut être trouvée ici: <http://freenmea.net/docs>.

Les données utilisées pour ce projet sont la latitude et la longitude. Ces données peuvent être extraites à partir du message qui commence par GPGGA. Ci dessous un exemple du format de ces données :



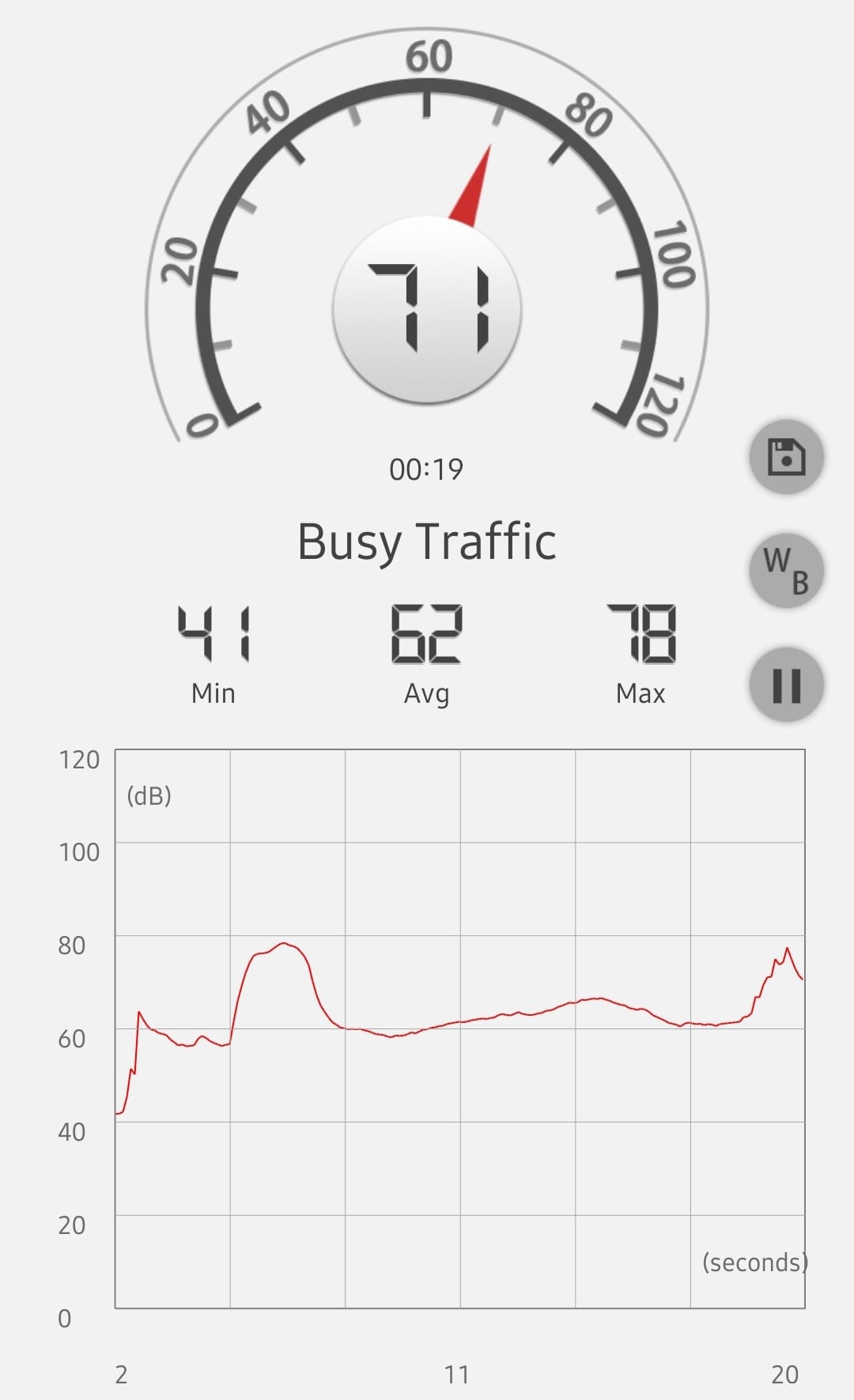
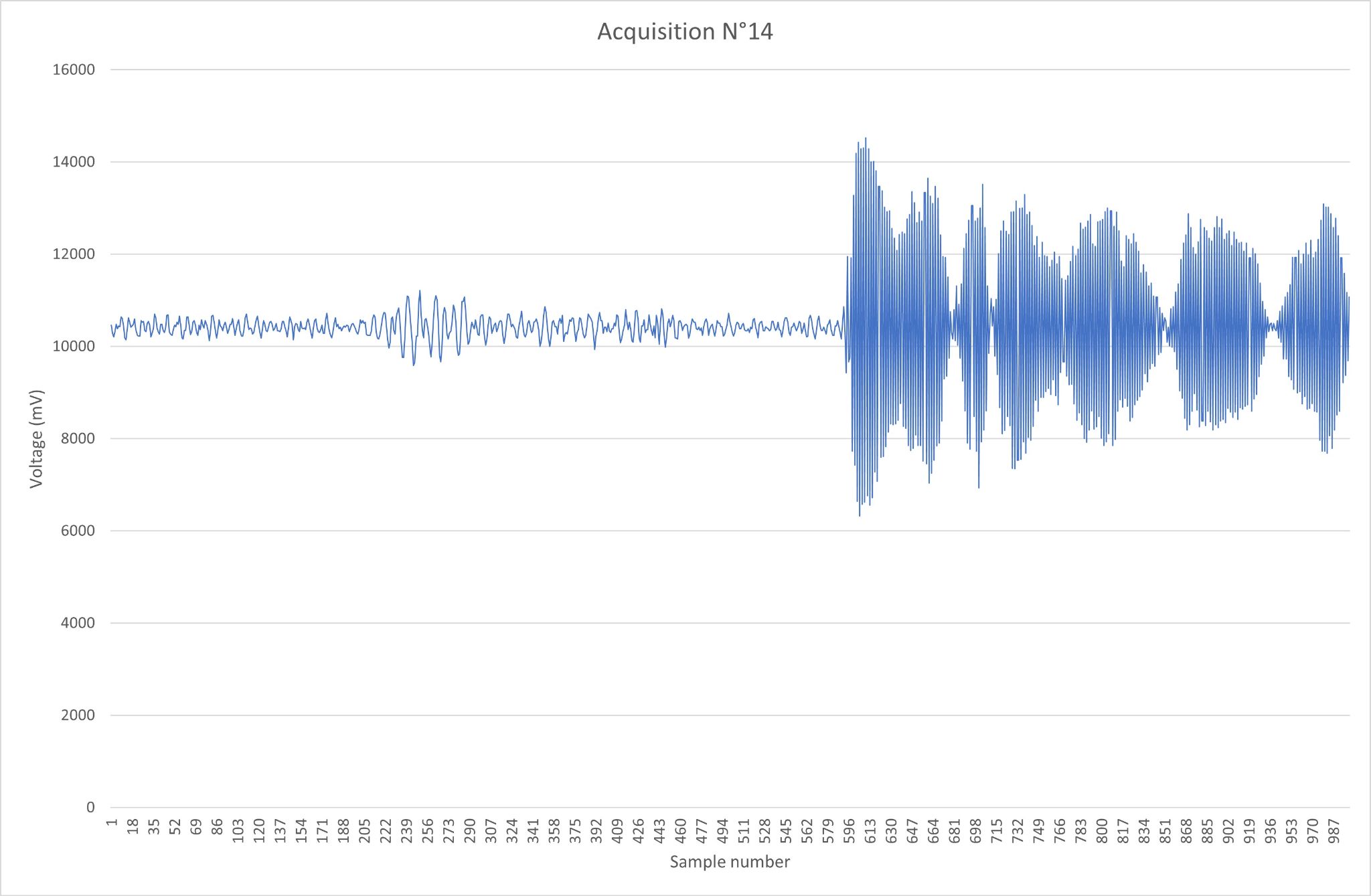
Quant à la sécurité du GPS, les données sont en clair et il n’y a pas de chiffrement qui est effectué. Ceci rend sensible le récepteur à des attaques: par exemple une personne malveillante pourrait générer un signal GPS (avec l’équipement approprié) et fausser la mesure de la position (et donc déplacer le projet sans qu’il y ait possibilité de le savoir).

#### Microphone

Un microphone à électret permet la détection du son. Le microphone est monté sur une carte comprenant un amplificateur (MAX9814) qui fournit en sortie un signal analogique amplifié. Après plusieurs tests, le gain de cet amplificateur a été choisi de 50dB (pin GAIN connecté à GND). Une autre caractéristique du circuit de gestion du microphone, est qu’il y a une option par défaut pour avoir un gain automatique : cela veut dire que l’amplificateur va adapter automatiquement le signal de sortie pour avoir toujours le même niveau sonore. Pour remédier à cela, nous avons dessoudé un résistor et court-circuité les pins.

Comme le Raspberry Pi n’a pas une entrée audio analogique, un convertisseur 12bit analogique-numérique a dû être utilisé pour pouvoir traiter le son. Ce convertisseur communique ensuite en I2C avec le Raspberry Pi pour transmettre les données. Une carte d’adaptation a également été utilisée pour pouvoir brancher le convertisseur sans devoir souder des composants.

Ci-dessous on peut voir un exemple du signal que l’on reçoit quand on lit les données du microphone :



Nous avons extrait l’amplitude du signal pour pouvoir calculer la valeur en décibel du niveau du son. Pour le calibrage, on s’est servi d’une application pour smartphone. Plusieurs applications ont été vérifiées pour s’assurer que les valeurs étaient correctes.

Après avoir récupéré les données du microphone, plusieurs filtres ont été testés pour avoir une meilleure qualité du signal et réduire le bruit impulsionnel. Notamment, un filtre moyenneur et un filtre médian ont été testés.

## Problèmes rencontrés

* Les modules micro avaient un gain automatique, ce qui empêche une mesure réelle du niveau sonore. Une résistance a dû être enlevée afin de désactiver ce mode.
* Du matériel semblait DOA (un micro) ce qui empêchait l’avancement de certains tests. Il se trouve que c’était une mauvaise interprétation des données.
* Le PiHat ne permettait pas de connecter d’autres périphériques. Un header spécial a été commandé.
* L’utilisation du module LoRa du PiHat s’est révélée assez difficile, puisque il manquait des exemples et des sources en ligne.
* Pour pouvoir utiliser le module Waspmote avec le Raspberry Pi, il faut pouvoir lire et écrire des données en même temps. Il n’a pas été trouvé une solution qui permet de faire cela.

## Conclusions

Ce projet a permis au groupe de développer un prototype sur un semestre et avait les avantages d’une approche pratique à un problème précis.

Ce travail nous a permis d’être interdisciplinaires car il était au croisement des systèmes embarqués, des télécoms et du développement.

Nous avons appris à travailler en groupe et à nous organiser, au moyen de rendez-vous internes hebdomadaires afin d’assurer le suivi de la réalisation du projet.

Des points à améliorer seraient de prendre plus de temps au début du projet afin d’assurer la compatibilité du matériel à acheter avant le lancement de la phase de réalisation.

Un suivi obligatoire avec les professeurs aurait été bienvenu car il aurait permis de plus régulièrement “rendre des comptes” et ainsi assuré une progression plus fluide.

Malgré cela, ce projet a été intéressant à implémenter et a permis l'acquisition de compétences autrement jamais utilisées.

1. Cette valeur découle des calculs obtenus dans la section "[formatage des paquets](#_2skse0blq0zx)”, voir la partie “[conclusions](#cs6fj1bltpww)”. [↑](#footnote-ref-0)
2. https://www.frugalprototype.com/wp-content/uploads/2016/08/an1200.22.pdf [↑](#footnote-ref-1)
3. data contient le payload effectif, c'est-à-dire les données envoyées ainsi que l’overhead. Cet overhead vaut 13 bytes dans le cas de notre application. [↑](#footnote-ref-2)