Simulateur LoRa
Rapport de stage

Tanguy Culerier

Septembre 2021

Sous la direction de :
Pierre Coucheney
David Auger
Jean-Michel Fourneau

Ce projet a été partiellement financé par la région
Paris Ile-de-France Region via une subvention du DIM RFSI (projet EPINE).

Table des matières

1 Introduction

Ce stage a été réalisé au laboratoire David de l'université de Versailles Saint-Quentin-en-Yvelines et encadré par Pierre Coucheney, David Auger et Jean-Michel Fourneau.
Le stage a été réalisé en collaboration avec un autre stagiaire Pierre Demonti. Je me suis occupé de la conception du simulateur et La conception des algorithmes d'apprentissage a été faite par Pierre.
Au vu de la situation sanitaire, le stage s'est déroulé en partie en distanciel (quatre jours par semaine).

1.1 LoRa et LoRaWan

LoRa est une technologie qui permet la communication à faible coût énergétique et à bas débit entre un réseau de capteur et une antenne. Ces communications peuvent se faire sur des distances qui peuvent atteindre une dizaine de kilomètres. Ces capteurs, consomment très peu d'énergie et il est donc possible de les alimenter avec une batterie pendant un temps assez long de l'ordre de plusieurs années. Ces capteurs vont envoyer leurs paquets à une antenne, qui elle, est connectée directement à internet et au réseau électrique. Cette antenne va envoyer via le réseau internet les données collectées. Le réseau formé par l'ensemble des capteurs et de l'antenne forme un réseau en étoile.
Un réseau de capteur LoRa peut permettre par exemple de mettre en place un ensemble de capteur d'hygrométrie dans un champ, sans avoir besoin de les relier au réseau électrique.

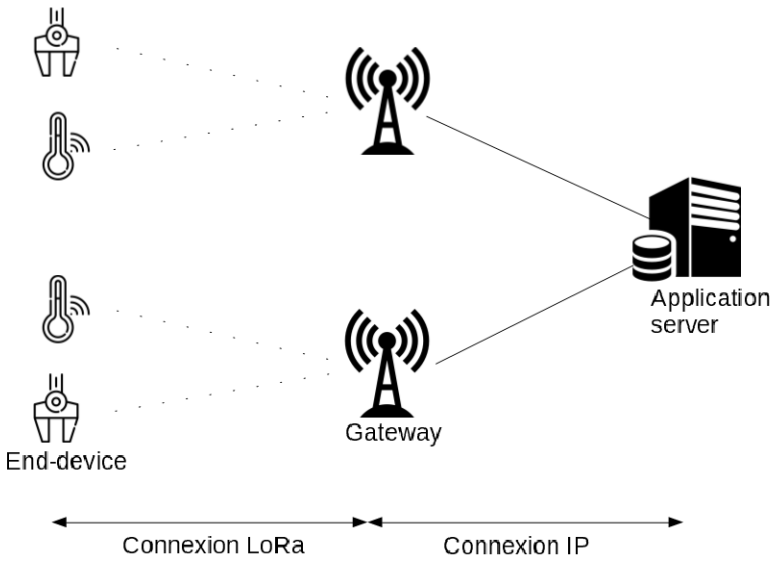


FIGURE 1 – Schéma d'un réseau LoRa

LoRaWan est quant à lui le protocole qui va permettre de gérer la communication dans ce réseau. Il va par exemple définir les fréquences utilisées et les différents paramètres de communication. Les fréquences utilisées pour l'envoi des paquets sont des fréquences libres, elles sont donc soumises aux différentes législations en fonction des pays. Il est nécessaire de respecter ces différentes règles pour pouvoir émettre sur celle ci.
LoRaWan est basé sur le protocole aloha, ce qui implique que chaque capteur va envoyer des paquets de manière indépendante les uns des autres. Cela implique que si deux capteurs émettent en même temps, alors dans certains cas il existe un risque que les deux paquets rentrent en collisions et que ceux-ci soient perdus. Cela obligera le capteur à réémettre le paquet une seconde fois. Il existe dans la communication LoRa un ensemble de paramètres, qui permet d'empêcher les collisions. Il est donc nécessaire de bien choisir ces paramètres.

1.2 Objectif

Comme les capteurs sont alimentés avec des batteries, il est nécessaire de préserver l'espérance de vie de celles-ci, pour optimiser le temps où le réseau sera utilisable. Les capteurs LoRa possèdent un ensemble de paramètres qui vont influencer de diverse manière, la consommation énergétique des capteurs. Certains de ces paramètres vont diminuer (ou augmenter) le risque de collision (et donc économiser l'énergie nécessaire pour renvoyer le paquet). Et d'autres impactent directement la consommation énergétique des capteurs. Ces paramètres peuvent également modifier la distance maximum où un paquet émis par un capteur peut être capté par l'antenne.
Le but de ce stage est de concevoir un simulateur capable de modifier ces différents paramètres de manière dynamique, en utilisant des algorithmes d'apprentissages. Cela permet de tester leur efficacité pour optimiser l'espérance de vie des batteries.
L'article utilisé pour la base de ce stage [1] décrit l'application de différent algorithme de bandit à plusieurs bras. Cet article décrit un modèle de simulation pour LoRa ainsi que des données sur l'utilisation d'algorithmes d'apprentissages. Cet article a donc servi de base pour la conception du simulateur.
Ce stage s'est décomposé en deux grandes parties. La recherche et le croisement des informations sur la technologie LoRa et la communication dans un réseau LoRa. Et la conception du simulateur à partir des informations collectées.

2 État de l'art

2.1 Fonctionnement LoRa

Les capteurs LoRa envoient de manière périodique des paquets à l'antenne (figure 2). De manière simple, ces paquets sont composés de trois parties. Le préambule qui permet d'initialiser la communication entre le capteur et l'antenne. L'entête qui contient les informations concernant le capteur et le paquet en lui même. Et pour finir le payload qui contient la charge utile (les données à transmettre).

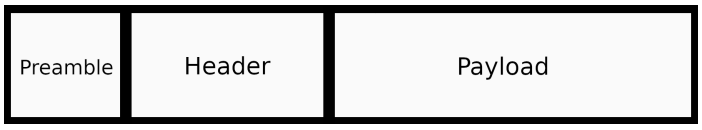


Figure 2 – Paquet LoRa

Un capteur LoRa communique avec son antenne à l'aide de la modulation *chirp*. Les informations sont encodées en faisant varier la fréquence du signal. Le paquet est découpé en symbole (plusieurs bits par symbole) et chaque symbole sera transmit par une variation spécifique de la fréquence. Par exemple le symbole de préambule est encodé par une variation partant de la fréquence la plus petite pour atteindre en fin de la transmission du symbole la fréquence la plus grande.

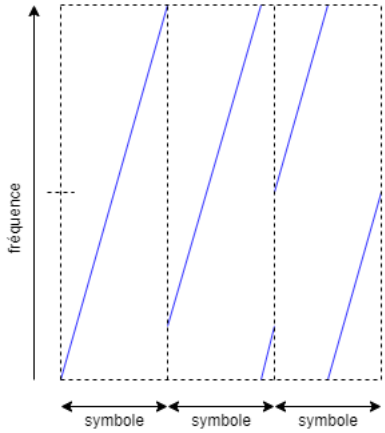


FIGURE 3 – Modulation chirp

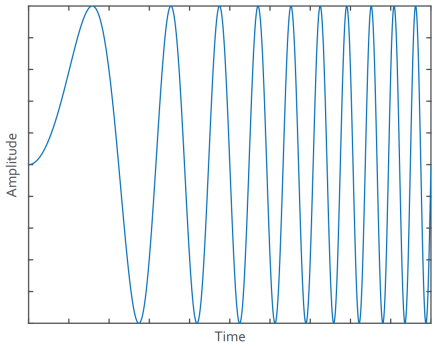


FIGURE 4 – Symbole de préambule

La figure 3 représente l'émission de plusieurs symboles. Le premier correspond au symbole de préambule. Le signal correspondant à ce symbole est représenté sur la figure 4.
Sur la figure 3 on peut voir deux autres symboles qui encodent deux séquences binaires différentes.

Les capteurs LoRa peuvent être paramétrés selon trois classes (mode de fonctionnements) différentes. La classe A consiste au fait que le capteur est allumé uniquement quand celui-ci émet ou qu'il attend l'acquittement de l'arrivée de son paquet. L'antenne peut donc communiquer avec un capteur seulement si il vient d'émettre un paquet. La classe B (balise) et C (capteurs allumés en permanence) consomment trop d'énergie et donc empêchent d'utiliser des batteries pour alimenter les capteurs. J'ai donc fait le choix pour le simulateur de n'utiliser que la classe A.

2.2 Paramètres

Dans la communication LoRa, différents paramètres doivent être réglés. Certains de ces paramètres peuvent être impactés par la législation du pays où le réseau est installé (par exemple les fréquences utilisées ou encore la puissance d'émission des paquets). De plus, ces paramètres vont avoir un impact sur la distance de réception (puissance et spreading factor) ou encore sur la gestion d'erreur lors de la transmission (coding rate).

2.2.1 Spreading Factor

Le Spreading Factor (ou facteur d'étalement en français) et abrégé en SF, est un paramètre qui peut être dynamique dans un capteur LoRa. Ce paramètre va impacter la distance à laquelle un paquet peut être entendu ainsi que la durée de transmission d'un paquet.
Le SF est noté comme un entier allant de 7 à 12 (il existe un SF6 mais celui-ci n'est jamais utilisé). Le fait de paramétrer correctement le SF va avoir plusieurs impacts sur la communication dans le réseau LoRa.
Le premier impact qu'aura le choix du SF, est que les différents SF sont quasiment orthogonaux. Ce qui veut dire que si deux capteurs sont réglés sur deux SF différents (par exemple le SF 7 et le SF 12), alors, si les paquets sont émis en mêmes temps, il n'y aura pas de collisions et donc les deux paquets seront captés par l'antenne.
Le second effet qu'aura le choix du SF est que celui ci impacte la distance à laquelle le paquet pourra être capté. Plus le SF sera grand, plus le paquet pourra être démodulé de loin. En contrepartie, le temps d'émission d'un paquet sera plus long si on augmente le SF.

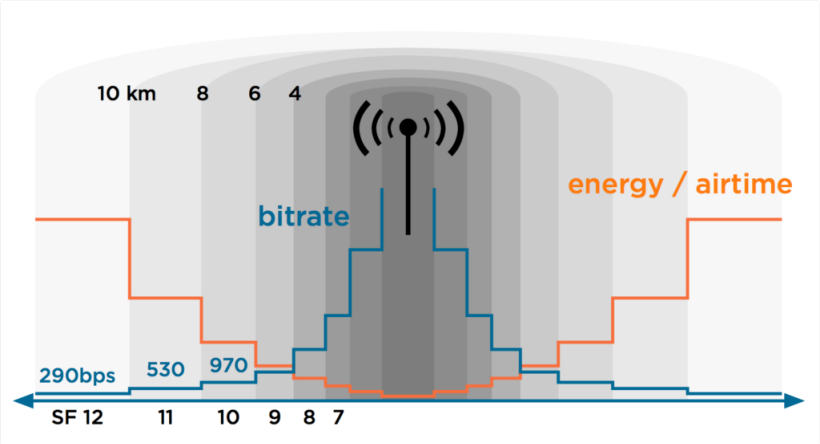


FIGURE 5 – Distance en fonction du SF

Comme on peut le voir sur le graphique figure 5, plus le SF est grand, plus il sera possible que le paquet soit envoyé loin. Par exemple, on peut voir qu'avec un SF 8, il est possible d'envoyer un paquet à 4 kilomètres. Alors qu'avec un SF 12, il est possible d'envoyer un paquet à 10 kilomètres (les distances peuvent changer en fonction du modèle de la puce LoRa, de l'antenne et de la densité du milieu).
On peut également voir sur ce graphique que plus le SF est grand, plus on consomme d'énergie pour envoyer un paquet. Cette augmentation est causée par le fait que plus le SF est grand, plus le débit binaire est petit.

facteur d'étalement	débit (bit/s)
SF12	250
SF11	440
SF10	980
SF9	1760
SF8	3125
SF7	5470

TABLE 1 – Tableau du débit binaire en fonction des SF

On peut voir sur ce tableau que le débit binaire diminue fortement quand le SF augmente. Ce qui va impliquer que plus le SF est grand plus l'envoie d'un paquet sera long. Et donc le capteur consommera plus d'énergie pour envoyer un paquet.

Ce paramètre à donc beaucoup d'importance dans la consommation énergétique, un SF trop petit empêchera les paquets d'arriver à destination. Alors qu'un SF trop grand va fortement augmenter la consommation énergétique (car le temps d'émission augmente grandement avec les SF). Cette augmentation du temps d'émission augmente également, le risque de collision.
De plus le choix du SF est important car d'autre capteur peuvent emmètre en même temps et donc générer une collision. Le capteur devra donc réémettre le paquet perdu. Choisir un SF qui n'est pas trop utilisé par d'autre capteur va permettre d'éviter des collisions et donc d'économiser de l'énergie.

2.2.2 Power

La puissance d'émission d'un paquet (en dBm) correspond à la puissance à laquelle le paquet sera émis. Cette puissance varie entre -2 et 20 dBm. Plus la puissance est grande, plus il sera possible d'envoyer un paquet à une grande distance. Mais en contrepartie, la consommation énergétique va augmenter en fonction de la puissance. Les valeurs de l'énergie consommée en fonction de la puissance dans le simulateur sont tirées d'un calculateur LoRa [2] et sont données en milliampère.
L'algorithme permettant de gérer le dynamisme du facteur d'étalement et de la puissance, est appelé ADR (Adaptive Data Rate) et est donné dans le protocole LoRaWan [3]. Cet algorithme n'est pas définit strictement et il en existe donc plusieurs versions, dont certaines utilisent des mesures réelles. Il est donc difficile d'implémenter cet algorithme dans un simulateur.

2.2.3 Bandwidth

Le bandwidth (ou largeur de bande en français), peut prendre trois valeurs : 125, 250 ou 500 kHZ, mais il semble que seul le bandwidth 125 soit utilisé pour les SF de 7 à 12.
L'utilisation du bandwidth 500 n'est jamais mentionné dans le protocole LoRaWan. Le bandwidth 250 semble permettre l'utilisation d'un SF noté 6. Mais LoRaWan autorise uniquement l'utilisation des SF compris entre 7 et 12, ce bandwidth n'est pas utile. La modification de ce paramètre a donc été bloquée dans le simulateur développé.

2.2.4 Coding Rate

Le coding Rate, est un paramètre qui va impacter la gestion d'erreur lors de la transmission des paquets. Avant l'envoi d'un paquet, un certain nombre de bits de gestions d'erreur sont ajoutés. Ces bits permettent de détecter si un paquet a été altéré lors de son envoi.
Le Coding Rate, noté CR, peut valoir 1, 2, 3 ou 4. Ce qui correspond que pour quatre bits, de 1 à 4 bits de gestion d'erreur seront ajoutés. Plus le CR est grand moins le paquet risque d'être perdu à cause d'altération par des interférences.
Comme le CR augmente la taille des paquets, le temps d'émission pour un paquet sera augmenté, (avec un CR de 4, la taille du paquet de base est multiplié par deux). Cette augmentation du temps reste faible par rapport à celle causée par l'augmentation du SF.

2.2.5 Fréquences

Les fréquences utilisées par LoRaWan sont des fréquences libres d'utilisation (ISM). LoRaWan définit trois fréquences à utiliser pour ce protocole (860, 864 ou 868 MHz). Il est possible d'utiliser des fréquences en plus de celle ci. Mais cela peut engendrer des collisions si deux fréquences se retrouvent trop proches. Les fréquences définit dans LoRaWan sont suffisamment éloignées pour empêcher ce genre de collisions.
Il est également important de noter que les fréquences libres sont soumises à différentes réglementations en fonction du pays où l'on se trouve. Par exemple en Europe, les fréquences utilisées dans LoRaWan sont soumises à un duty cycle de 1%. Ce qui veut dire qu'un capteur ne peut émettre qu'un pourcent du temps (il doit donc attendre 99 fois le temps d'émission de son paquet avant d'en envoyer un autre). Le temps d'émission est donc restreint (environ 36 secondes d'émission par heure) et donc le nombre de paquets maximal est donc faible en utilisant des grands SF. Par exemple, avec un SF 7, un paquet est envoyé en 56 ms et on peut donc envoyer 1318 paquets en une journée. Alors qu'avec un SF 12, un paquet est envoyé en 1318 ms et donc on pourra uniquement envoyer 65 paquets par jour.

2.3 Collisions

Lors de l'envoi de paquet dans un réseau LoRa, il faut que plusieurs conditions soient réunies pour que deux paquets rentrent en collision. Tout d'abord, les deux paquets doivent arriver en même temps. Les deux paquets doivent également être sur le même SF et sur une fréquence suffisamment proche.

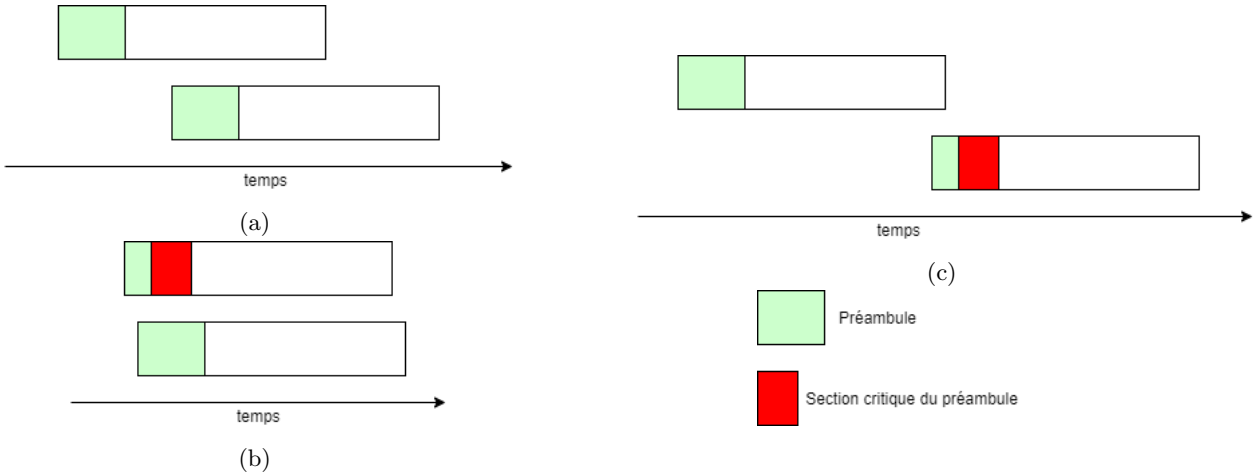


FIGURE 6 – Collisions entre paquets

Lorsque deux paquets rentrent en collision, alors il n'est plus possible de différencier le contenu de ces paquets. Les deux paquets sont donc perdus (Figure 6(a)).
Dans certaines conditions il est également possible qu'un paquet survive à une collision :
Il faut se placer dans le cas où un paquet de faible puissance arrive en premier et que la section critique du préambule (les cinq derniers symboles) ne soit pas encore arrivée à l'antenne. Le paquet qui arrive en second doit avoir une puissance suffisamment grande pour écraser le premier paquet et prendre sa place. Le premier paquet sera donc considéré comme du bruit. On appelle cela l'effet de capture. Et un seul sera perdu lors de la collision (figure 6(b)).
Un paquet peut également réussir à passer juste après un autre si la section critique du préambule de ce paquet arrive à l'antenne après la fin du paquet précédent (il est possible de perdre une partie du préambule, figure 6(c)).
Minimiser le nombre de collisions permet d'éviter que les capteurs soient en obligation de réémettre les paquets pour que ceux ci soit au final captés par l'antenne. Cela permet donc d'économiser la batterie des capteurs.

2.4 LoRaSim

Le simulateur LoRaSim [4] est un simulateur de réseau LoRa très utilisé. Mais ce simulateur possède des défauts de conception et permet uniquement de simuler des réseaux avec des paramètres statiques. Or l'objectif de ce stage est de pouvoir avoir un simulateur qui permet d'utiliser des algorithmes d'apprentissage afin de définir les paramètres optimaux pour maximiser l'espérance de vie de la batterie. Il faut donc que les paramètres soient dynamiques au cours de la simulation.
D'autre défaut du simulateur sont que certaines parties du code ne sont pas fonctionnelles. Certaines parties comme la réémission sont également gérées de manière approximative ce qui rend complexe l'utilisation d'algorithme d'apprentissage.
Au lieu d'utiliser directement le code de LoRaSim, j'ai choisi de réimplanter la base du simulateur, en utilisant un échéancier et des évènements, pour remplacer Simpy une bibliothèque Python qui permet de faire des simulations basées sur des processus.

3 Simulateur

Le but de ce stage était de concevoir un simulateur capable dans un premier temps de simuler un réseau LoRa, de manière à se rapprocher le plus possible de la réalité. Le second objectif de ce simulateur est d'être suffisamment modulaire pour permettre l'utilisation de différents algorithmes d'apprentissage afin de gérer les paramètres des capteurs, dans l'objectif d'optimiser la durée de vie de la batterie. Contrairement à LoRaSim, pour permettre l'utilisation de ces algorithmes, le simulateur est capable de modifier les paramètres des capteurs pendant la simulation.
Ce simulateur permet donc de simuler ce qui se passe dans un réseau LoRa lorsque des algorithmes d'apprentissage sont utilisés. Cela permet donc de comparer leurs efficacités dans différentes conditions.

3.1 Fonctionnement de la simulation

Comme dit précédemment, le simulateur qui a été réalisé, est un simulateur en temps discret, qui utilise un échéancier et un ensemble d'évènements. Comme les capteurs envoient leurs paquets de manières indépendantes, la gestion des évènements d'un capteur sera assez régulière. L'ensemble de ces évènements sont représentés sur le schéma ci dessous.

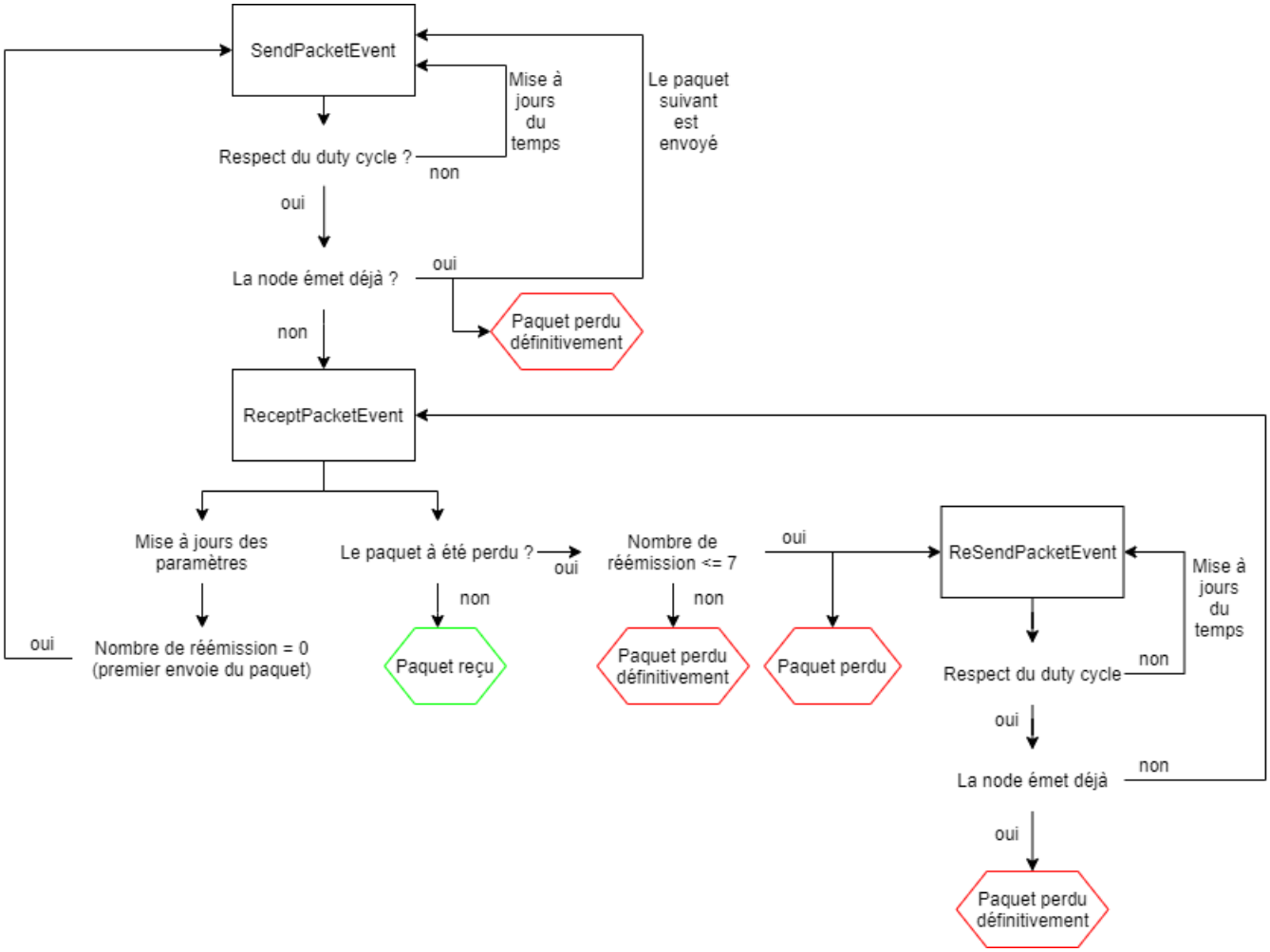


FIGURE 7 – Schéma de la gestion des évènements

Le premier évènement SendPacketEvent va être créé. Cet évènement contient le paquet qui sera envoyé et la date d'envoi du paquet, lorsque cet évènement sera exécuté par l'échéancier. Il faut tout d'abord vérifier que le Duty Cycle est bien respecté. Pour cela une variable dans l'objet correspondant au capteur, contient le temps à partir du quel, il peut émettre. Si le Duty Cycle n'est pas respecté, alors un nouvel évènement avec une date mise à jour est créé.
Par la suite on vérifie que le capteur n'est pas en train d'émettre, si c'est le cas, le paquet est perdu et le paquet suivant est préparé (création d'un nouvel évènement SendPacketEvent)
Si toutes les conditions sont validées, le paquet peut être émis et un évènement ReceptPacketEvent est créé. La date de cet évènement correspond à la date où le paquet aura été totalement envoyé.

Quand un évènement ReceptPacketEvent est exécuté par l'échéancier, deux choses se passent :
Dans un premier temps, on vérifie si le paquet a subi une collision (le paquet a bien été réceptionné par l'antenne). Si le paquet a subi une collision on va vérifier si le nombre de réémission est inférieur ou égal à 7 (le nombre de réémission maximale est défini par LoRaWan). Si le paquet peut être encore réémit, un évènement ReSendPacket est créé. Cet évènement va permettre de renvoyer le paquet ayant subi une collision.
Dans un second temps, les paramètres du capteur vont être mis à jour à l'aide de l'algorithme définit dans le capteur. Puis si le paquet qui vient de finir d'être reçu en est à sa première émission, on prépare l'envoi du suivant en créant un nouvel évènement SendPacketEvent.

L'évènement ResendPacketEvent fonctionne quasiment de la même manière que SendPacketEvent à la différence que lorsqu'un capteur essaye d'envoyer un nouveau paquet tout en étant déjà en train d'en émettre un autre, celui ci ne créera pas de nouveau paquet. La création du paquet suivant se fait lorsqu'un paquet n'ayant jamais subi de réémission est capté par l'antenne ou perdu.

Cette boucle d'événement s'exécutera pour tous les capteurs jusqu'à ce que l'échéancier atteigne la date maximum de simulation ou qu'un capteur ai dépensé la totalité de l'énergie contenue dans sa batterie.

3.2 Node et Packet

Les classes Node et Packet sont très importantes dans le simulateur car elles vont conserver toutes les données de paramétrage. Elles représentent respectivement les capteurs et les paquets dans un réseau LoRa réel. Ces objets vont représenter l'état du système. La classe Node contient les paramètres des capteurs LoRa ainsi que différents compteurs pour conserver les données de la simulation. La classe Packet va quant à elle contenir les paramètres qui ont été utilisés pour envoyer le paquet et certaines variables qui serviront lors de la simulation.

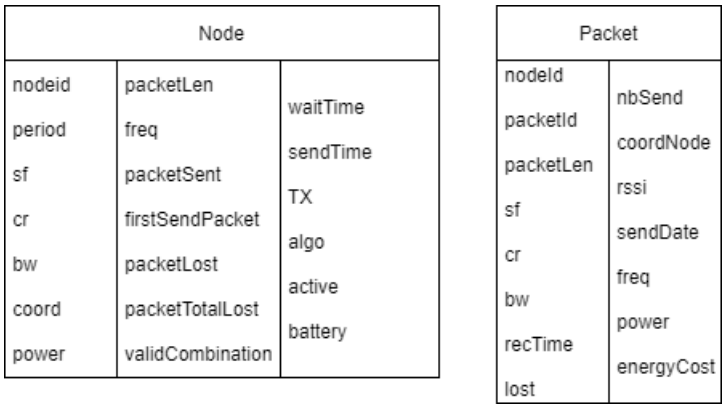


Figure 8 – Classe Node et Packet

3.2.1 Node

Cette classe représente un capteur LoRa du réseau. Elle contient beaucoup d'arguments qui permettent de conserver l'état des paramètres du capteur, conserver des données propres à la puce LoRa ainsi que des compteurs qui permettent de collecter des données au cours de la simulation.

Pour permettre les différents calculs lors de la simulation, il est nécessaire de conserver les différents paramètres liés à LoRa. La variable *SF* contient le Spreading Factor qui sera utilisé par le capteur. Cet argument est un entier compris entre 7 et 12.
La puissance d'émission est contenue dans la variable *power*, qui est défini par un entier compris entre -2 et 20. L'unité de la puissance est le dBm.
En plus des deux paramètres les plus importants pour l'énergie et les collisions, les trois autres sont également essentiels pour les calculs. Le coding Rate est contenue dans la variable *CR*. Cette variable est un entier compris entre 1 et 4. Le bandwidth est stocké dans la variable BW et peut en théorie prendre 3 valeurs (125, 250 ou 50). Comme très souvent seule la largeur de bande 125 est utilisée, dans le simulateur cette variable est fixée à 125. L'unité du bandwidth est le kilohertz. Le dernier paramètre, la fréquence de la porteuse peut prendre les valeurs : 860000000, 864000000, 868000000. Cette fréquence est stockée dans la variable *freq*. L'unité de la fréquence de la porteuse est le Hertz.

La classe Node contient également des attributs qui définissent l'état du capteur. *NodeId* et *coord* définissent l'identifiant du capteur et sa position. Cette donnée est connue dans le simulateur mais pas forcément dans un réseau LoRa. Un objet *battery* permet de conserver la consommation énergétique du capteur. Deux attributs concernent les paquets qui sont envoyés, *packetLen* qui correspond à la taille des paquets qui seront émis. Et *period* qui correspond, au temps moyen (en milliseconde) entre deux envoie de paquet. Le temps est tiré à l'aide d'une loi exponentielle de paramètre λ (tel que $\lambda = 1/period$).
L'algorithme d'apprentissage qui permet de choisir les paramètres est stocké dans un objet *algo*. Cet objet permet de rendre modulaire l'ajout de nouveaux algorithmes.

Les variables *waitTime* et *sendTime* correspondent respectivement au temps que doit attendre le capteur après sa dernière émission et au temps où le dernier paquet a été envoyé. Ces deux variables permettent de gérer le duty cycle.
ValidCombination est une liste qui contient toutes les paires SF / puissance qui sont utilisables en fonction de la distance. L'attribut *TX* contient le tableau qui permet de convertir la puissance utilisée en milliampère.
Différents compteurs sont conservés dans l'objet Node pour pouvoir, pendant la simulation conserver différentes informations, qui seront par la suite écrite dans des fichiers CSV ou sur des graphiques. Les différents compteurs sont les suivants :

- *packetSent*, est un compteur du nombre de paquets total qu'a envoyé le capteur.
- *firstSendPacket*, est un compteur du nombre de paquets qui ont été émis pour la première fois (les réémissions ne sont pas comptabilisées).
- *packetLost*, est un compteur du nombre de paquets qui ont été perdu (collisions, paquet de trop faible puissance).
- *packetTotalLost* est un compteur du nombre de paquet qui n'ont pas été transmis (7 réémissions échouées).

Cette classe va principalement servir pour stocker les paramètres et les compteurs au cours de la simulation. Mais elle possède également la capacité de créer les paquets qui seront utilisés dans les évènements.

3.2.2 Packet

Les instances de la classe Packet sont crées par le capteur qui les émet. Des arguments tels que packetLen, sf, cr, bw, coordNode, freq et power proviennent du capteur (Node) . Ils sont donc égaux aux paramètres du capteur lors de l'envoi du paquet. Ces objets paquets sont chacun stocké dans des évènements SendPacketEvent ou reSendPacketEvent si le paquet n'a pas encore été envoyé. Et dans des évènements receptPacket si le paquet est en cours d'envoi.

L'attribut *recTime* contient le temps que va mettre le paquet à être envoyé. Il est calculé à partir de la taille des paquets, du spreading factor, du bandwidth et du Coding Rate. Ce calcul provient de la documentation technique de seemtech [5] et est également utilisé dans LoRaSim [4]. Les équations utilisées pour calculer le temps d'émission d'un paquet sont les suivantes :

$$Tsym = \frac{2^{SF}}{BW} \tag{1}$$

Cette équation permet d'obtenir le temps nécessaire pour émettre un symbole, SF étant le spreading factor utilisé pour envoyer le paquet et BW la largeur de bande utilisée.

$$Tpaquet = Tpreambule + Tpayload \tag{2}$$

Un paquet LoRa est composé d'un préambule et de la charge utile (entête et données à transmettre). Il faut donc sommer le temps pris pour transmettre le préambule ($Tpreambule$) et celui pour transmettre le reste des données ($Tpayload$).

$$Tpreambule = (nbPreambule + 4.25) * Tsym \tag{3}$$

Cette équation permet de calculer la durée d'émission du préambule du paquet LoRa. Par défaut *nbPreambule*, le nombre de symbole composant le préambule, est fixé à 8.

$$Tpayload = payloadSymNb * Tsym \tag{4}$$

$$payloadSymNb = 8 + max(\left\lceil \frac{8 * PacketLen - 4 * SF + 28 + 16 - 20 * H}{4 * (SF - 2 * DE)} \right\rceil * (CR + 4), 0) \tag{5}$$

Ces deux équations permettent de calculer le temps nécessaire pour émettre la charge utile du paquet. H vaut 0 si l'entête du paquet est présente (et 1 sinon). Dans le simulateur tous les paquets possèdent une entête et H est donc fixé à 0.
DE correspond à l'utilisation d'une optimisation (low data rate optimization) et vaut 1 si elle est activée (0 sinon). Dans le simulateur cette option est désactivée (et donc DE est fixé à 0).
A l'aide de l'équation (5) on peut donc obtenir la taille en symbole du paquet. Il reste donc à multiplier cette valeur par le temps d'envoi d'un symbole pour obtenir le temps d'envoi du payload en millisecondes.

L'attribut *lost* est un booléen qui marque si le paquet a été perdu (puissance trop faible ou collision). Cet attribut sert lors de la gestion des collisions.
nbSend est le compteur du nombre de fois où le paquet a été émis. Lors de la première émission, cette variable est fixée à 0. Si cette variable est inférieure à 7, il est possible de réémettre le paquet en cas de collision.

Le *rssi* correspond à la puissance du signal lorsque celui ci arrive à l'antenne. En effet la puissance du signal est impactée par une perte de puissance en fonction de la distance. Pour calculer cette perte de puissance, j'ai repris le même modèle de propagation que dans LoRaSim. Il s'agit du *Log-distance path loss model* [6]

$$lpl = lpld0 + 10\gamma \log_{10}(\frac{d}{d0}) + \chi \tag{6}$$

Le résultat, noté lpl, correspond à l'atténuation du signal (en dB).
lpld0 est la valeur de référence de l'atténuation du signal qui a été prise dans LoRaSim. Cette constante est fixée dans le simulateur à 127.41. d0 la distance de référence, est elle, fixé à 40.
La variable γ correspond au path loss exponent, qui est le coefficient de l'affaiblissement du signal. Cette valeur est également celle de LoRaSim et est fixée à 2.08.
La variable d quant à elle, correspond à la distance entre le capteur et l'antenne, cette valeur est exprimée en mètres. La variable χ quant à elle correspond à une variation basée sur une fonction gaussienne centrée en 0. Cette variation n'est pas utilisée dans le simulateur (et donc χ est fixé à 0).

$$rssi = Power - Gl - lpl \tag{7}$$

Pour calculer le rssi, qui correspond à la puissance du signal lors de son arrivée à l'antenne. Le rssi se calcule en soustrayant l'atténuation du signal à la puissance d'émission du signal.[7]
Gl représente la somme des gains et des affaiblissements du signal qui sont liés à l'environnement. Comme il ne nous est pas possible de les connaître et que les différents simulateurs ne l'utilise pas. Il a été choisi de fixer cette variable à 0.
Le rssi permettra de déterminer si le message arrive avec une puissance suffisante pour l'antenne et permettra également de comparer la puissance entre deux paquets lors des collisions pour déterminer si l'effet de capture intervient.

L'attribut *energyCost* contient l'énergie qui sera consommée par l'envoie du paquet par le capteur.

$$energyCost = \frac{rectime}{3600000} * TX \tag{8}$$

Le coût énergétique (en milliampère-heure) est calculé avec l'équation ci dessus. La variable *rectime* correspond au temps d'envoi (en milliseconde) du paquet. Et TX, correspond à la consommation énergétique du capteur (en milliampère) en fonction de la puissance sur laquelle il est réglé.

3.3 Combinaison SF / puissance valide

L'antenne, en fonction des paramètres des paquets, possède ce qui s'appelle une sensibilité. Cette sensibilité correspond au rssi minimum que peut avoir un paquet en fonction du SF choisi. Cette sensibilité varie en fonction du bandwidth. Plus celui ci est grand moins la sensibilité permet d'obtenir des paquets de faible rssi, c'est pour cette raison que le bandwidth 125 est en général le seul à être utilisé.

SF	sensibilité (BW125)
7	-126.50
8	-127.25
9	-131.25
10	-132.75
11	-134.50
12	-133.25

TABLE 2 – Tableau de la sensibilité en fonction du SF

Cette table des sensibilités a été récupérer dans l'article. *Do LoRa Low-Power Wide-Area Networks Scale ?* [7]. Cette table des sensibilités peut varier en fonction de modèle de l'antenne et des puces LoRa.
Grâce à cette table des sensibilités, il est possible de calculer la distance maximale où on peut placer les capteurs en fonction de la puissance d'émission et du SF. Dans LoRaSim, une modification de l'atténuation a été faite et a rendu faux ce calcul. Mais il est possible de retrouver l'équation à l'aide du calcul du rssi. Il a donc été possible avec l'équation ci-dessous, de calculer cette distance maximum.

$$maxDist = d0 * 10^{\frac{sensi-power+lpld0}{10*\gamma}} \tag{9}$$

A partir de cette équation et des paramètres utilisés dans le simulateur (en utilisant les données de la table 2), il est possible de déterminer que le la distance maximum où il est possible de placer un capteur avec les paramètres SF12 et une puissance de 20 dBm est de 896 mètres (et 369 mètres avec un SF7).

3.4 Collisions

Les collisions sont très importantes dans la communication LoRa. Comme le réseau est composé d'un ensemble de capteurs qui vont émettre de manière indépendante, le risque d'une collision entre deux paquets devient très important. Le duty cycle va permettre d'éviter une surcharge du réseau, mais les collisions sont toujours possibles. L'intérêt de minimiser les collisions, est dans un premier temps d'éviter de perdre des informations. Comme il est impossible d'éviter totalement les collisions, il est prévu dans LoRaWan que les capteurs peuvent réémettre leurs paquets (jusqu'à 7 fois) s'ils ne reçoivent pas d'acquittement de l'antenne. Mais ces remisions coûtent de l'énergie et donc réduisent l'espérance de vie de la batterie. Il est donc important de bien configurer les paramètres pour minimiser le nombre de collisions.
Il est possible de perdre de manière différentes les informations contenues dans un paquet : les collisions, un paquet avec une puissance trop faible ou encore les erreurs de transmissions. Ce dernier point est lié au Coding Rate (qui permet de corriger les erreurs). La gestion des erreurs de communication et leur correction n'a pas été implémentée dans le simulateur.

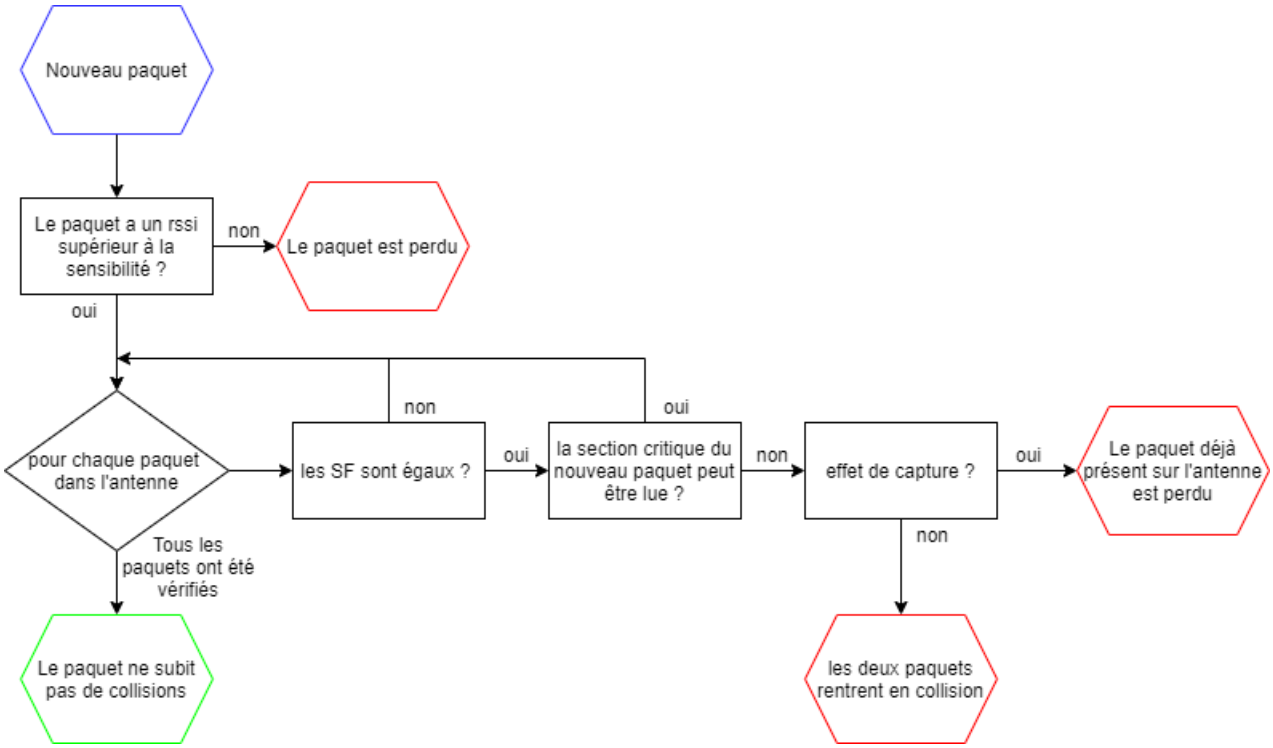


FIGURE 9 – Schéma de la gestion des collisions

Le schéma ci dessus représente la gestion des collisions dans le simulateur. Tout d'abord, on vérifie que le rssi du paquet est supérieur à la sensibilité correspondant au SF de l'antenne. Si le rssi est trop bas l'antenne ne sera pas capable d'entendre et donc de démoduler le paquet. Celui ci sera donc perdu. Par la suite le simulateur va tester si le paquet rentre en collisions avec l'un des paquets qui sont déjà en train d'être transmis.
Dans un premier temps, on vérifie si les deux paquets utilisent le même SF. Si le SF n'est pas le même, on considérera qu'il ne peut pas y avoir de collision. Dans la réalité, les SF sont quasi-orthogonaux et donc il peut y avoir une collision entre deux paquets ayant un SF proche. Pour simplifier la gestion des collisions, les collisions inter-SF ne sont pas prises en compte dans le simulateur.
Si le SF des deux paquets est le même, alors on va vérifier si il est possible que le paquet qui arrive, passe juste après le paquet déjà en train d'arriver. En effet si la section critique de l'entête du paquet peut être lu alors celui ci peut être démodulé. Dans le simulateur le préambule est composé de 8 symboles et la section critique de 5. Il est donc possible de capter le paquet si le paquet dans l'antenne se termine avant que le capteur n'envoie 3 symboles. Si cette condition n'est pas remplie, une collision a lieu.
Il y a deux types de collisions pour les paquets LoRa, la collision ou les deux paquets sont perdus et la collision où le paquet qui arrive va capturer le paquet étant en train d'arriver. L'effet de capture arrive lorsque le paquet qui arrive est beaucoup plus puissant que le paquet était dans en train d'arriver. Le nouveau paquet doit avoir un rssi étant au moins 6dBm plus grand que l'autre paquet.
Lors d'une collision simple, les deux paquets verront leurs variables *lost* être affectées à True. Dans le cas d'une capture, seul le paquet qui était déjà en cours d'envoi, va avoir une mise à jour de sa variable *lost*.
Cette variable va signaler au simulateur lors de l'exécution de l'évènement receptPacketEvent que le paquet n'a pas été acquitté et donc qu'il est nécessaire de réémettre le paquet. Le capteur ne peut pas connaître la raison de la perte de son paquet. Dans un réseau réel il est donc quasiment impossible de déterminer d'où proviennent les pertes de paquets.

3.5 Apprentissage, paramètres, et énergie

Le but de ce simulateur est de pouvoir gérer de manière dynamique les paramètres afin d'optimiser l'espérance de vie des batteries. Rendre dynamique ces paramètres, va permettre d'utiliser des algorithmes d'apprentissage.
Pour cette étape deux grandes parties du simulateur ont dû être conçues :

— Concevoir la base du simulateur pour pourvoir modifier facilement les paramètres au cours de la simulation.
— Rendre simple le fait d'utiliser des algorithmes d'apprentissage et également pouvoir en rajouter de nouveaux sans devoir reconcevoir le simulateur à chaque ajout d'un nouvel algorithme.

Pour rendre les paramètres dynamiques, à chaque fois qu'un paquet a été émis (réceptionné ou perdu), le simulateur fait appel à la méthode *chooseParameter* de son objet *algo*. Cet objet contient l'algorithme d'apprentissage qui va définir le comportement du capteur. Ne m'occupant pas de la partie conception des algorithmes d'apprentissage, j'ai mis en place deux comportements. Le premier étant le comportement statique où le capteur ne change jamais de configuration (celui ci me permet de faire des comparaisons avec LoRaSim). Le second comportement implémenté est le fait qu'à chaque fois que le paquet d'un capteur est perdu, les paramètres du capteur sont choisis de manière aléatoire dans les paires (SF, Power) qui sont valides pour le capteur. Avec ce second comportement, il est déjà possible d'observer une organisation des capteurs sur les différents SF. Par contre la consommation d'énergie n'est pas optimale. L'utilisation de véritable algorithme d'apprentissage, permet de diminuer cette consommation d'énergie.

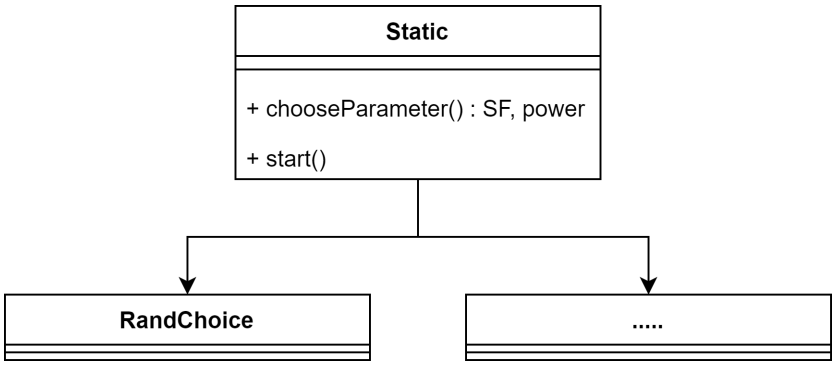


FIGURE 10 – Hiérarchie des classes d'apprentissage

Pour permettre l'ajout simple de nouveaux algorithmes, ceux ci sont encapsulés dans une classe qui hérite de la super classe *Static*. Dans cette super classe, deux méthodes sont présentes et ne font rien (car ce comportement ne change pas les paramètres) :

— *start* est la méthode qui sera appelée lors de l'initialisation de l'algorithme par le capteur. La méthode permet de mettre en place l'algorithme lors de la création du capteur.
— *chooseParameter*, comme dit précédemment, est appelée à chaque arrivée de paquet (qu'il soit perdu ou non). Cette méthode permet de mettre à jour les paramètres SF et Power du capteur.

Ces méthodes doivent être implémentées dans les sous classes pour pouvoir modifier le comportement du capteur. Pour ajouter simplement un nouvel algorithme d'apprentissage, il suffit donc d'ajouter une nouvelle classe étendant la classe *Static* dans le fichier des algorithmes et d'ajouter cette classe dans un fichier de configuration qui permet de signaler au programme son existence.

3.6 Utilisation

J'ai testé pendant ce stage plusieurs simulateurs LoRa et je me suis rendu compte que ceux ci rentre les paramètres de simulation directement dans la ligne de commande utilisée pour les exécuter. Or cela devient très complexe car le nombre d'arguments devient très grand. J'ai donc décidé d'utiliser plusieurs fichiers de configuration lors de la conception du simulateur. Certain de ces fichiers servent à paramétrer les particularités des capteurs (sensibilité, TX) ou à ajouter de nouveaux algorithmes d'apprentissage. Un autre fichier de configuration permet quant à lui de placer et déterminer les différents paramètres des capteurs. Son utilisation va permettre de mettre en place plusieurs groupes de capteurs avec des paramètres différents (ce qui serait très compliqué à faire uniquement avec une ligne de commande) ou encore réutiliser une configuration d'une simulation précédente. Un autre fichier de configuration permet d'ajouter des évènements qui serviront à déplacer les capteurs au cours de la simulation
Ces différents fichiers de configuration simplifient l'utilisation du simulateur et permettent également de le rendre moins rigide que les simulateurs qui ont servi de base.

3.7 Données de simulation

L'intérêt de concevoir ce simulateur est de collecter des données pour pouvoir par la suite les utiliser pour mettre en place les paramètres des algorithmes d'apprentissage ou encore prévoir le comportement d'un réseau dans diverses situations. De ce fait le nombre de données générées est très grand.
Dans un premier temps des graphiques sont générés, ceux ci permet de voir rapidement comment a évolué le système au cours de la simulation.
Les graphiques suivant ont été générés après une simulation de 100 capteurs LoRa, placés à une distance suffisamment proche pour pouvoir utiliser l'intégralité des SF.

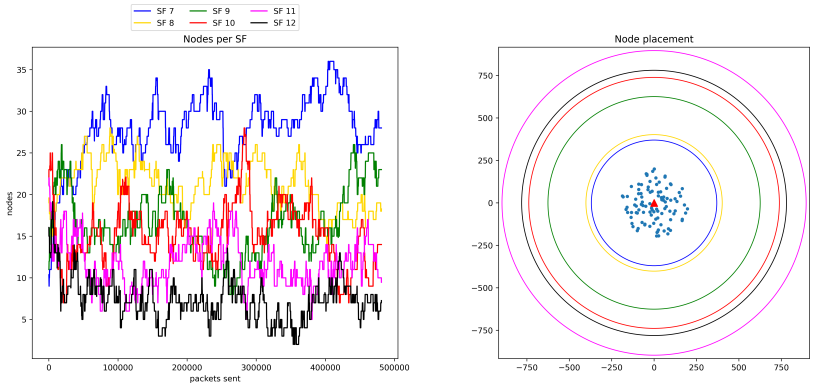


FIGURE 11 – Graphique du nombre de capteurs par SF et du placement des capteurs

Le premier graphique figure 11 permet de voir le nombre de capteurs pour chaque SF pendant la simulation, celui ci permet de voir si les capteurs favorisent un SF en particulier. Ce fichier contient également un graphique qui montre le positionnement des capteurs par rapport à l'antenne. Dans cette simulation, le SF de chaque capteur a été tiré de manière aléatoire. On peut voir sur le graphique "node per SF", qu'une organisation de l'utilisation des SF semble se mettre en place. Cela est du au fait que si une collision se produit, le capteur utilise de manière aléatoire l'une des paires utilisable (SF, puissance), comme nouveau paramètre. Le risque de collision étant plus faible sur les petits SF (car le temps d'émission d'un paquet est moins long), les capteurs ont tendance à les utiliser.

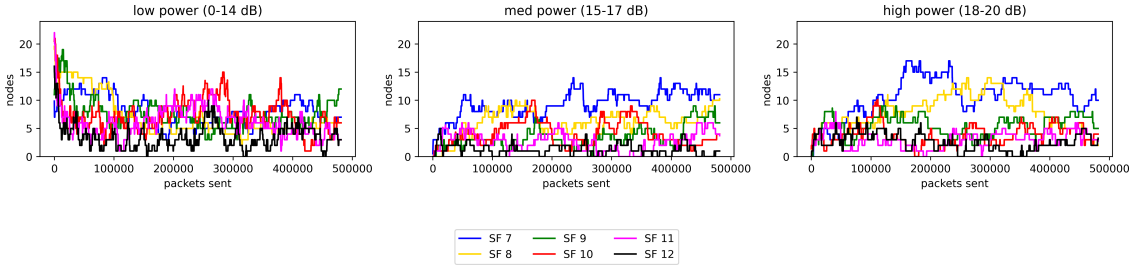


FIGURE 12 – Graphiques du nombre de capteurs en fonction du SF et de la puissance

Les graphiques figure 12 permettent de représenter la répartition des capteurs en fonction du SF et de la puissance. Les trois graphiques correspondent à 3 niveaux de puissance (0-14, 15-17, 18-20 dBm). Comme les capteurs sont proche de l'antenne, beaucoup de combinaisons SF / puissance sont utilisables dans cette simulation. Les capteurs utilisent donc toutes les puissances à leurs dispositions.

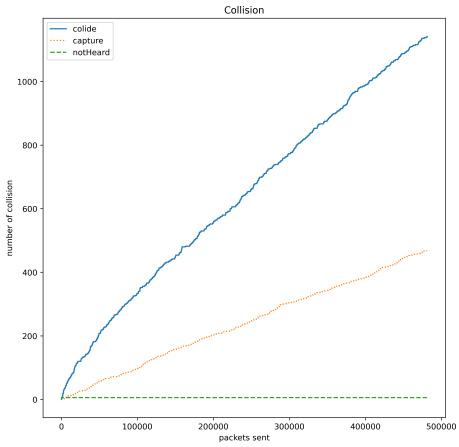


FIGURE 13 – Graphique des collisions

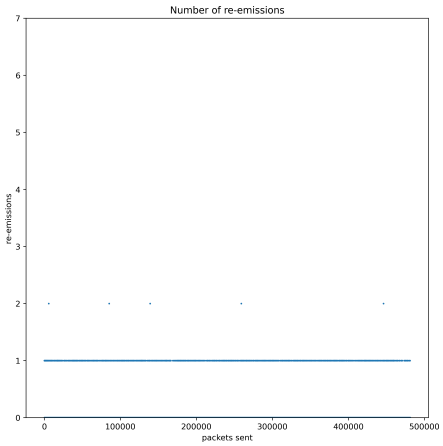


FIGURE 14 – Graphique des réémission

Le graphique figure 13 représente le nombre de paquets perdus au cours du temps. On peut voir que dans cette simulation, la majorité des pertes de paquets sont causés par des collisions simples. Le restant des paquets perdus est causé par l'effet de capture. Il n'y a pas de perte de paquet dû à un rssi trop faible car les paramètres utilisés appartiennent à la liste des combinaisons SF / puissance utilisables.
Le graphique figure 14 représente le nombre d'émission de chaque paquet. Ce graphique permet d'estimer le nombre de réémissions et donc si les paquets arrivent à destination. On peut voir ici que le nombre de fois où les paquets sont émis n'évolue pas pendant la simulation car le changement aléatoire des paramètres n'améliore pas les collisions avec 100 capteurs pendant la simulation. On peut également observer que le nombre maximum de réémissions pour un paquet est deux et que c'est arrivé pour seulement 5 paquets. Le reste des paquets a été transmis lors de la première émission ou avec une retransmission.

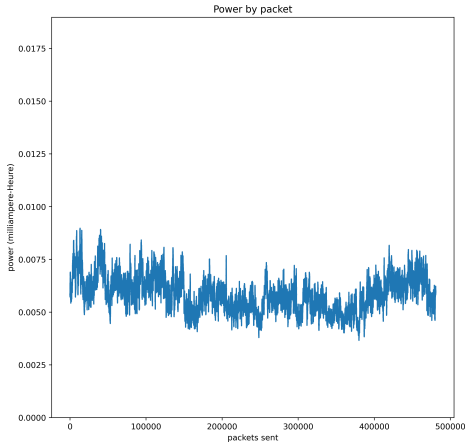


FIGURE 15 – Graphique de la consommation énergétique par paquet

Le graphique de la figure 15 représentation de la consommation énergétique moyenne du réseau (en milliampère-heure). Si l'algorithme d'apprentissage est efficace, cette courbe doit avoir une tendance descendante.
On peut voir sur ce graphique qu'avec une méthode aléatoire pour changer les paramètres, il n'y a pas de diminution de la consommation énergétique.

Le simulateur génère également des journaux au format CSV, qui vont par exemple conserver des informations sur l'état du système (état des batteries, nombre de collisions) ou l'état des capteurs de manière individuel au cours de la simulation. Un fichier (*saveENV.txt*) contient quant à lui la sauvegarde de l'état des capteurs à la fin de la simulation. Le contenu de ce fichier peut être utilisé pour relancer une nouvelle simulation avec les mêmes capteurs.

```
# time,day,sf,power,energy,firstSentPacket,packetColid,packetTotalLost
979701,0,12,14,0.0161,1,0,0
2211606,0,12,14,0.0322,2,0,0
4226910,0,12,14,0.0484,3,0,0
7367410,0,12,14,0.0645,4,0,0
8265456,0,12,14,0.0806,5,0,0
9317053,0,12,14,0.0967,6,0,0
10548879,0,12,14,0.1128,7,0,0
10862825,0,12,14,0.1290,8,0,0
11017959,0,12,14,0.1451,9,0,0
```

FIGURE 16 – Fichier journal du premier capteur

La figure 16 correspond à une partie du fichier qui collecte les informations du premier capteur. Chaque ligne de ce fichier correspond à l'état du capteur après avoir fini d'envoyer un paquet. Chaque ligne va contenir la date de l'envoi du paquet, les paramètres SF et power, l'énergie qui a été consommée par le capteur, le nombre de nouveaux paquets envoyés, le nombre de paquets perdus et pour finir le nombre de paquets définitivement perdus (dont le nombre réémissions est supérieur à 7). Toutes ces données permettent de retracer l'historique du capteur pour comprendre son comportement (et celui de l'algorithme d'apprentissage utilisé) dans le réseau.

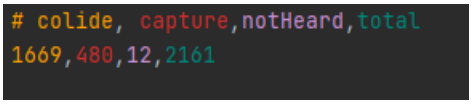


FIGURE 17 – Fichier des collisions

Le fichier de la figure 17 contient les compteurs des différents motifs de perte de paquets. Cela permet d'identifier les causes majeures de perte. Ce fichier contient les mêmes valeurs que le graphique figure 13 et permet de se faire une idée sur la cause majeure des pertes de paquets.

3.8 Scénario

Il est possible lors d'une simulation d'intervenir pour altérer la configuration du début. Il a été rendu possible à l'aide d'un fichier de configuration d'ajouter des évènements pour déplacer un capteur. Ces évènements ont pour but de déplacer un capteur pour observer comment l'algorithme d'apprentissage va réagir à cette modification. Cela peut être utile car un réseau LoRa n'est pas forcément statique (par exemple des capteurs fixés sur du bétail).
Les capteurs peuvent en début de simulation, être placés de plusieurs manières différentes à l'aide du fichier de configuration, de manière aléatoire, en grille ou aligné. Il est également possible de placer un capteur aux coordonnées que l'on souhaite.

3.9 Problèmes rencontrés

Plusieurs problèmes ont été rencontrés lors de la conception de ce simulateur. Dans un premier temps, les calculs utilisés dans le simulateur utilisent des données physiques. Il est complexe de trouver des valeurs liées à des conditions spécifiques. Par exemple beaucoup paramètres du calcul de la perte de signal (lpld0 et γ) sont des mesures réels.
De plus, certaines données des capteurs LoRa comme le TX ou la sensibilité de l'antenne peuvent énormément changer entre deux modèles de puce (ou d'antenne). Les fichiers de configuration ont permis de contourner cette problématique.

Un autre problème dans la simulation d'un réseau LoRa est la surcharge du nombre de paquets à envoyer. En effet, comme les capteurs peuvent envoyer un nombre maximum de paquets par heure, dans certains cas, avec les réémissions, ce maximum est grandement dépassé. Dans la littérature cet aspect de la simulation n'est jamais pris en compte car il ne touche pas directement le fonctionnement des puces LoRa, mais l'implémentation logicielle du capteur.

L'un des grands problèmes lors de la conception du simulateur est que nous n'avons pas trouvés de données sur ce qui se passe réellement dans un réseau LoRa. Il a donc été difficile de déterminer si le simulateur est fonctionnel où non. Pour essayer de vérifier la véracité de nos simulations, j'ai donc essayé de comparer mes résultats avec d'autres simulateurs.

4 Résultat

4.1 Simulation simple

Comme première expérimentation j'ai comparé une simulation avec une configuration simple (10 capteurs et le SF fixé à 12. Cette simulation a une durée de 100 jours et chaque capteur envoie un paquet environ toutes les demi-heures.
Dans un premier temps on peut observer que les deux résultats sont proches. la simulation a été faite dix fois sur chaque simulateur.
LoRaSim trouve en moyenne 527 collisions pour 48085 paquets envoyés. Mon simulateur quant à lui, trouve en moyenne 631 collisions pour 48334. Les résultats sont très proches, mais mon simulateur trouve un peu plus de collisions. Cela est causé par le fait que LoRaSim ne gère pas les réémissions.

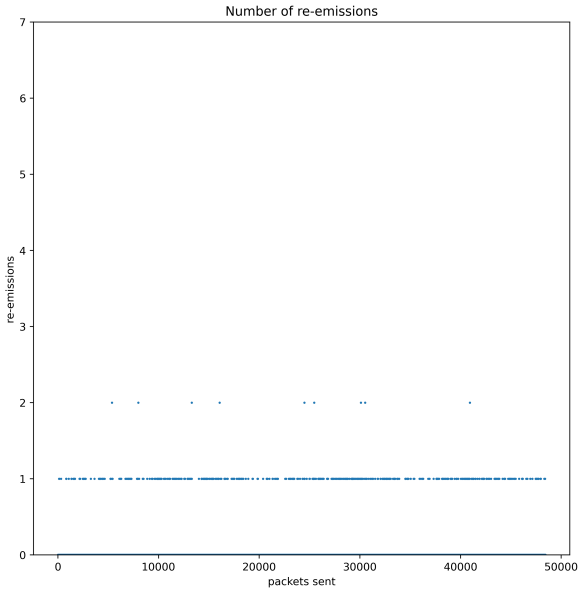


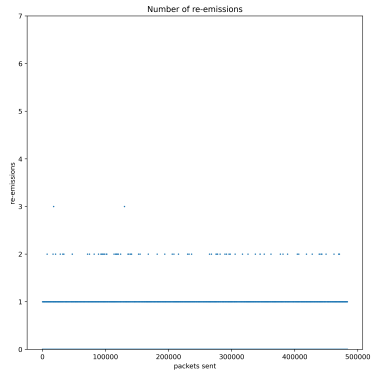
FIGURE 18 – Graphique des réémitions (10 capteurs, comportement statique)

Sur le graphique figure 18 on peut voir que certains paquets ont été réémis deux fois, cela augmente donc le nombre de paquets en circulation dans le réseau.

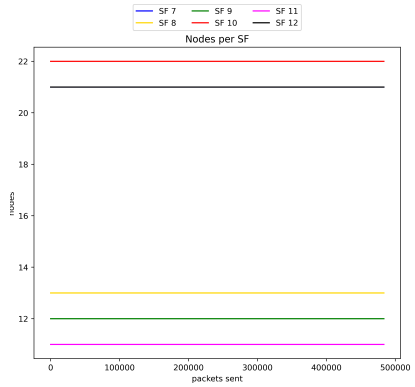
Cette première expérimentation permet de vérifier que le simulateur est capable dans des cas simples de trouver des résultats qui se rapprochent du simulateur existant.

4.2 Stratégie aléatoire VS stratégie statique

Dans cette partie, deux simulations sont comparées. Les deux simulations comportent 100 capteurs qui ont la même fréquence d'envoi des paquets. Les capteurs sont placés suffisamment proche pour pouvoir utiliser tous les SF.
Dans la première simulation, les capteurs ne changent pas de paramètres. La seconde utilise une "stratégie aléatoire", où à chaque fois qu'un paquet subi une collision, le capteur associé à ce paquet tire de manière aléatoire une paire SF / puissance dans la liste des combinaisons valides.

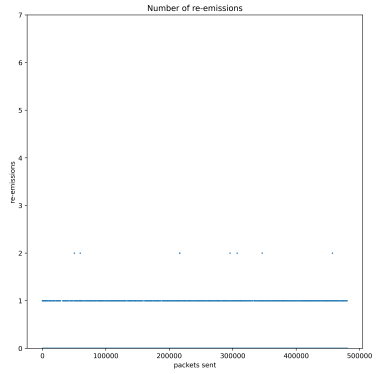


(a) Graphique des réémissions

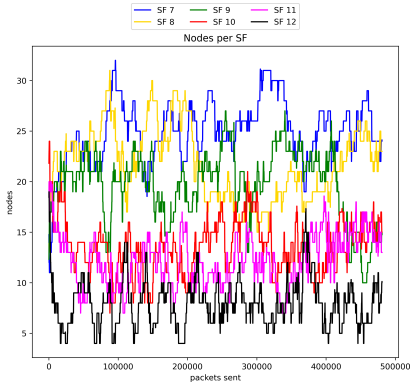


(b) Graphique d'utilisation des SF

FIGURE 19 – Simulation statique



(a) Graphique des réémissions



(b) Graphique d'utilisation des SF

FIGURE 20 – Simulation choix aléatoire

On peut observer sur le graphique 20(a), il y a moins de paquet réémit plus d'une fois que sur le graphique 19(a). Cela est dû au fait que dans la seconde simulation, on peut observer que les capteurs semblent s'être répartis sur les SF où les paquets sont émis le plus rapidement (SF 7, 8 et parfois 9). Le simple fait de modifier les paramètres de manière dynamique au cours de la simulation permet donc de diminuer le nombre total de collisions et donc la consommation énergétique. Le choix des paramètres SF / puissance impacte également beaucoup la consommation énergétique. La stratégie aléatoire ne fait pas cette optimisation.

5 Conclusion

Au cours de ce stage, un simulateur de réseau LoRa a été conçu. Ce simulateur est capable de simuler le fonctionnement des capteurs LoRa avec une gestion dynamique des paramètres. Ces paramètres dynamiques permettent l'utilisation de différents algorithmes d'apprentissage pour tenter d'optimiser la consommation énergétique des capteurs. Il est suffisamment modulaire pour pouvoir ajouter de nouveaux algorithmes simplement. L'utilisation de différents fichiers de configuration permet de faire simulation avec différents modèles de réseau (placements et réglages des capteurs, utilisation de scénario de déplacement). Ces fichiers de configuration permettent également de simuler différents modèles de puce LoRa. Le simulateur est suffisamment modulaire pour pouvoir ajouter et utiliser différents algorithmes d'apprentissage de manière simple. Ce simulateur est capable de simuler des réseaux de bonne taille (plus de 100 capteurs) et sur des durées longues (plus de 1000 jours). Le simulateur obtenu correspond aux objectifs fixés au début de ce stage.

Une partie du travail de ce stage, a été de recueillir des informations sur la technologie LoRa et le protocole LoRaWan. Cette recherche d'information a permis de collecter et de recroiser les différentes informations, pour pouvoir les utiliser dans le simulateur. La conception du simulateur a été faite en parallèle de cette recherche, pour permettre les tests sur des modèles simples qui ont peu à peu été complexifiés. Les résultats obtenus après différentes simulations, sont correctes et en les comparant avec les données déjà existantes (simulateur déjà existant), on peut supposer que le simulateur fonctionne bien.

Ce simulateur peut être amélioré car certains phénomènes physiques ont été volontairement écartés pour simplifier certains aspects du simulateur. Dans un premier temps, pour s'approcher davantage de la réalité, les collisions inter-SF pourraient être implémentées. De plus dans un réseau réel, il peut y avoir plusieurs antennes. Un capteur peut donc envoyer ses paquets à plusieurs antennes différentes. Le réseau n'est donc plus uniquement en étoile.

Références

[1] Node-based optimization of LoRa transmissions with Multi-Armed Bandit algorithms, *Raouf Kerkouche, Reda Alami, Raphaël Féraud, Nadège Varsier, Patrick Maillé, 2018*

[2] LoRa Modem Calculator Tool ; SeemTech

[3] Rule and regulation, In Europe,
https ://lora.readthedocs.io/en/latest/#general-considerations

[4] LoRaSim,
https ://www.lancaster.ac.uk/scc/sites/lora/lorasim.html

[5] LoRa Modem Design Guide,
SeemTech,
https ://www.rs-online.com/designspark/rel-assets/ds-assets/uploads
/knowledge-items/application-notes-for-the-internet-of-things/LoRa%20Design%20Guide.pdf

[6] Log distance path loss or log normal shadowing model,
http ://www.idc-online.com/technical_references/pdfs/electronic_engineering/Log_Distance_Path_Loss_or_Log_Normal_Shadowing_Model.pdf

[7] Do LoRa Low-Power Wide-Area Networks Scale ?,
Martin Bor, Utz Roedig, Thiemo Voigt, Juan M. Alonso, 2016

[8] Réseaux très basse consommation, longue portée, bas débit,
Anthony Juton, 2019

[9] Cours - Formation LoRa / LoRaWAN,
Sylvain Montagny, Université Savoie Mont Blanc, 2019

[10] Optimal SF Allocation in LoRaWAN Considering Physical Capture and Imperfect Orthogonality,
Christelle Caillouet, Martin Heusse, Franck Rousseau, 2019

[11] Energy Consumption Model for Sensor Nodes Based on LoRa and LoRaWAN,
Taoufik Bouguera, Jean-François Diouris, Jean-Jacques Chaillout, Randa Jaouadi, Guillaume Andrieux, 2019

[12] Simulating LoRaWAN : on Importance of Inter Spreading Factor Interference and Collision Effect,
Juho Markkula, Konstantin Mikhaylov, Jussi Haapola, 2019