M2 AMIS

Rapport de Stage : Apprentissage pour un réseau LoRa

Pierre DEMONTI

Encadrants :

David AUGER
Pierre COUCHENEY
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).

SEPTEMBRE 2021

Sommaire

1 Introduction **4**

2 Objectif du stage et contexte **5**

3 Technologie LoRa **5**
3.1 Présentation générale 5
3.2 Noeud et Station de base 5
3.3 LoraWAN 6
3.3.1 Topologie 6
3.3.2 Étalement de spectre 6
3.3.3 ADR 7
3.3.4 Paramètres 8
3.3.4.1 Facteur d'étalement 8
3.3.4.2 Puissance 9
3.3.4.3 Autres paramètres 9
3.3.4.4 Coût énergétique 9
3.3.5 Collisions 9
3.3.6 Classes 10
3.3.7 Trame LoRa 11

4 Analyse de la technologie LoRa **11**
4.1 Une technologie récente 11
4.2 Les limites et LoRaWAN 12
4.3 Impact des paramètres 12

5 Simulateur **13**
5.1 LoRaSim 13
5.2 Autres simulateurs 13
5.3 Simulateur personnalisé 13

6 Etat de l'art **16**
6.1 Approche décentralisée 16
6.2 Approche centralisée 16
6.3 Approche hybride 16

7 Approche décentralisée et apprentissage par renforcement **17**
7.1 Apprentissage par renforcement 17
7.2 Bandit 17
7.2.1 Thompson sampling 18
7.2.2 UCB1 18
7.2.3 Noeud et Bandit 19
7.3 Fonction de coût 19
7.4 Q-Learning 19
7.5 Amélioration envisageable 20
7.6 Comparaison des différents algorithmes 21

8 Résultats **21**
8.1 Gestion de collisions . . . 22
8.2 Déplacement d'un noeud . . . 22
8.3 Résultats d'une approche centralisée par DQN . . . 23
8.4 Mesures pour une simulation normale . . . 23
8.5 Analyse et interprétation . . . 25
8.5.1 Approche décentralisée . . . 25
8.5.2 Approche centralisée . . . 25

9 Conclusion et Perspectives **26**

10 Annexes **27**
10.1 Approche centralis ée . . . 27
10.1.1 Deep reinforcement learning . . . 27
10.1.2 Dueling Deep Q network . . . 27
10.1.3 Méthodologie . . . 28
10.1.3.1 Utilisation de l'antenne . . . 28
10.1.3.2 Structure du réseau de neurones . . . 29
10.1.3.3 Fonctionnement . . . 29

Remerciements

Je remercie mes encadrants David AUGER, Pierre COUCHENEY, Jean-Michel FOURNEAU, pour m'avoir accompagné, guidé et conseillé pendant ce stage.

Je tiens aussi à remercier ma famille et mes amis pour leur patience et leur bienveillance durant cette période.

1 Introduction

De nos jours, l'utilisation de l'IOT et des capteurs est en pleine expansion. Que ce soit dans l'agriculture, pour le suivi de l'humidité des cultures, dans l'industrie, pour contrôler l'intégrité des installations ou éteindre les dispositifs, ou plus couramment dans les villes, dans le cadre des smart cities.
Toute cette diversité de systèmes correspond à des topologies multiples à densités variables, passant de de quelques capteurs très éparpillés sur de longues distances, à des contextes plus concentrés comme c'est le cas, par exemple, dans les smart cities. Tous ces capteurs fonctionnent généralement sur batterie du fait de leur positionnement, batterie qui nécessite donc d'être changé régulièrement. Il convient alors, dans un contexte de transition écologique, mais aussi de praticité, de mettre en place un fonctionnement équilibré sur le plan énergétique.
C'est dans ce contexte que LoRa ou Long Range, souhaite pallier ces problématiques en mettant en place un réseau économique sur le plan énergétique aussi bien de par la technologie que par les protocoles.
Ce stage s'est déroulé au sein du laboratoire DAVID pour Données et Algorithmes pour une Vie Intelligente et Durable situé sur le campus de l'UFR des sciences de l'Université de Versailles-Saint-Quentin-En-Yvelines. J'ai été intégré au sein de l'équipe ALMOST (Algorithms and Stochastic Models) et encadré par Auger David, Coucheney Pierre et Fourneau Jean-Michel. Ce stage fait suite à un autre portant sur la technologie LoRa et son fonctionnement de manière plus générale. Ce stage demande donc de poursuivre ces recherches mais aussi de les exploiter afin d'appliquer un choix dynamique de paramètres de transmission. Par ailleurs, il s'effectue en concert d'un autre stage au sein du laboratoire, qui lui est concentré sur le développement du simulateur permettant d'effectuer les tests. L'encadrement s'est effectué en deux parties. Une partie hebdomadaire, avec des réunions en présentiel pour faire le point de l'avancée et fixer des objectifs à atteindre. Une partie plus libre via une messagerie en ligne pour exposer des résultats intermédiaires ou discuter d'idées plus expérimentales. J'ai également eu l'opportunité de présenter une partie de mon travail lors d'une présentation de l'ensemble des travaux de stages et de doctorants du laboratoire. L'objectif de ce stage sera d'améliorer la performance énergétique côté protocole, à l'aide de techniques d'apprentissage par renforcement. On s'intéressera aux différents types de paramètres sur lesquels on peut jouer pour à la fois avoir un réseau permettant un échange de données solide tout en considérant l'aspect énergétique.
On présentera une approche de problème de bandit et comment l'adapter à notre problème, ainsi que l'analyse de performances de différents algorithmes.
Le stage a lieu en collaboration avec Tanguy Culerier, autre stagiaire, chargé de développer un simulateur adapté aux modifications que nous aurons besoin d'apporter.

2 Objectif du stage et contexte

LoRa est une technologie récente permettant à des capteurs de communiquer. L'objectif de ce stage est d'abord de mettre au point un simulateur en collaboration avec un autre stage. Ce simulateur devra permettre d'analyser le réseau et son comportement. Dans un second temps, on utilisera ce simulateur pour étudier des méthodes de choix de paramètres dynamiques.

3 Technologie LoRa

Pour pouvoir mettre en place un simulateur et comprendre comment gérer la sélection de paramètres lors d'une transmission, il nous faut d'abord comprendre comment LoRa fonctionne. Dans un premier temps, je vais présenter les principaux aspects de LoRa qui auront de l'importance pour la suite.

3.1 Présentation générale

LoRa, ou Long Range, est une technologie créée en 2010 par la start-up Cycleo, et ensuite rachetée en 2012 par l'entreprise américaine Semtech[10], permettant la transmission de données à longue portée. Elle est basée sur un principe de modulation de fréquence par étalement de spectre, permettant de transmettre des données sur de longues distances, le message transmis le plus loin ayant atteint les 12 km dans un espace ouvert. Il existe plusieurs types de réseaux utilisant la technologie Lora, notamment Zigbee et LoraWan. Dans ce rapport nous nous concentrerons sur LoRaWAN, qui, de par sa nature open source, et sa documentation plus fournie permet une approche plus libre.

3.2 Noeud et Station de base

Les deux composants principaux de LoRa sont les nœuds (nodes) et la station de base (base station). Un nœud est composé d'un capteur, peu importe sa nature, et d'un composant informatique permettant de gérer les protocoles de transmissions tel qu'un raspberry pi par exemple. La station de base est elle constituée d'une antenne et d'un terminal ayant accès à un réseau soit local, soit à internet directement. Dans le cas où plusieurs antennes sont installées, c'est le serveur auquel elles sont connectées qui décide quelle antenne répondra au capteur.

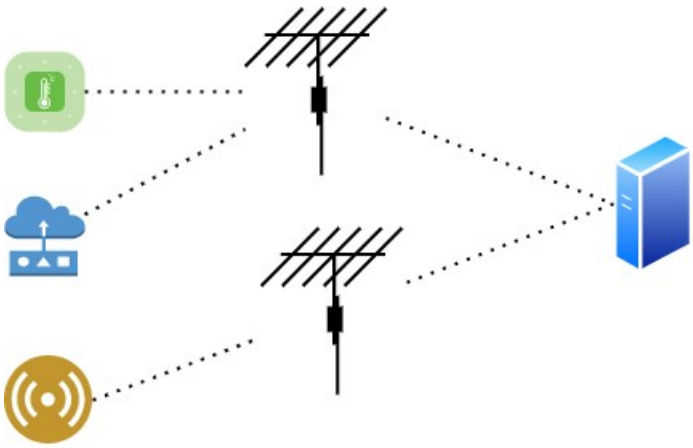

FIG. 1 – *Réseau LoRa*

3.3 LoraWAN

LoRaWan est le protocole associé à la technologie LoRa. Avec lui viennent plusieurs restrictions et normes permettant d'homogénéiser le fonctionnement du réseau.

3.3.1 Topologie

LoRaWAN est basée sur une topologie en étoile [3], c'est-à-dire qu'un capteur est connecté à une et une seule base, tandis qu'une base est associée à plusieurs capteurs.

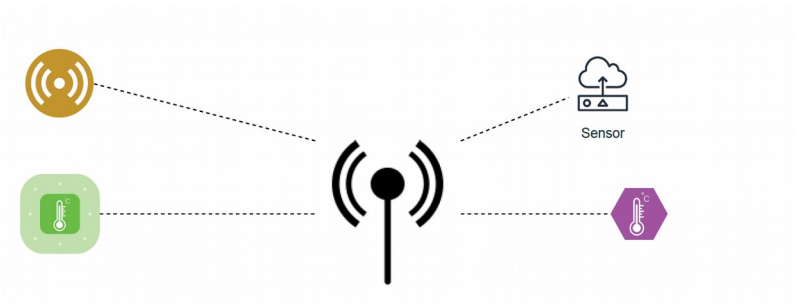

FIG. 2 – *Topologie en étoile LoRaWAN*

Cette topologie permet de faire fonctionner un grand nombre de capteurs en même temps, puisque l'on peut toujours rajouter des antennes en plus si celles en services ne sont plus suffisantes pour couvrir le réseau. En cas d'échec de transmission, un message peut être réémis jusqu'à sept fois.

3.3.2 Étalement de spectre

L'étalement de spectre est l'une des caractéristiques clés de la technologie LoRa. Il permet d'émettre des données sur de longues distances sans pour autant perdre en information et limiter le risque de collisions avec d'autres émissions. L'étalement de spectre consiste à générer un signal dont la fréquence varie de façon continue. Cette variation est définie par une valeur appelée facteur d'étalement ou spreading factor. Ces variations sont orthogonales, c'est-à-dire que pour deux valeurs de spreading factor différentes, il ne peut pas y avoir de collisions.

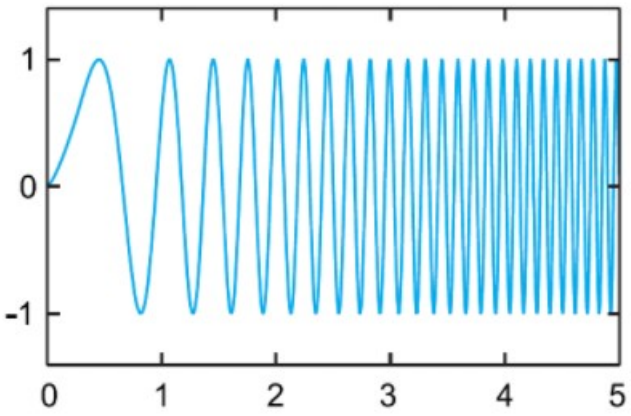

FIG. 3 – *Étalement de spectre*

3.3.3 ADR

Pour permettre aux capteurs de ne pas utiliser en permanence une combinaison de paramètres inutilement puissants et donc énergivores, Semtech a mis en place un algorithme baptisé Adaptative Data Rate ou ADR, qui a pour but de minimiser l'impact énergétique tout en contrôlant les collisions. Il est divisé en deux parties : une sur le capteur qui est chargé d'augmenter d'abord la puissance puis le Spreading Factor tant que l'antenne ne reçoit pas le message, l'autre sur l'antenne qui doit diminuer le Spreading Factor ou la puissance pour économiser de l'énergie. Il est important de noter qu'ADR est un algorithme générique. Il n'est donc pas précisé comment ces choix sont faits même si plusieurs heuristiques semblent faire consensus dans la documentation.

Algorithme 1 : Node part

```
Résultat : SF, Pw
tant que Message non reçu par la base faire
    si Pw == MaxPw alors
        SF +=1;
    sinon
        Pw +=1;
    fin
fin
```

Ici on choisit d'augmenter la puissance jusqu'au maximum. Si cela ne suffit pas alors on peut augmenter le facteur d'étalement.

Algorithme 2 : Base part

```
Résultat : SF, Pw
si RSSI < Seuil alors
    si Pw ≤0 alors
        SF -= 1;
    sinon
        Pw -= 1;
    fin
fin
```

Pour déterminer si la puissance du signal est trop forte, on utilise des heuristiques qui se servent généralement du RSSI ou Received Signal Strength Indication. En ce qui concerne le niveau à partir duquel on considère un signal comme étant trop fort, cela reste à l'appréciation de l'administrateur réseau. Comme on peut le constater, la particularité d'ADR est d'être un algorithme ouvert aux modifications et aux adaptations de chacun. Cependant, il est majoritairement basé sur des heuristiques, ce qui ne permet pas d'approcher efficacement les valeurs optimales.

FIG. 4 – *ADR*

3.3.4 Paramètres

LoRa permet de faire varier différents paramètres. Nous allons, dans un premier temps, présenter les deux paramètres LoRa que l'on peut modifier dynamiquement : le facteur d'étalement et la puissance.

3.3.4.1 Facteur d'étalement

Comme expliqué précédemment, la technologie LoRa est basée sur des méthodes d'étalement de spectre, permettant d'associer un message à un chirp dont la fréquence varie. Pour permettre l'optimisation de ces transmissions, plusieurs façons de faire varier le signal ont été mises en place : ce sont les facteurs d'étalement[5]. Ces facteurs d'étalement ont été prédéfinis et sont nommés par un nombre : SF1, SF2, SF3, etc. Lorawan utilise les SF de 7 à 12. Ainsi, un message émis à SF12 ira plus loin qu'un message émis à SF7. Cependant, cela implique plusieurs inconvénients. D'abord, les messages avec un spreading factor plus élevé, seront plus longs. En effet, avec l'étalement de spectre, on étale le message comme s'il était plus gros, rendant le temps de transmission plus long, un temps dans l'air plus long et donc un risque de collision plus élevé. De plus, générer un message plus long, signifie qu'il va falloir consommer plus d'énergie pour émettre le message. Pour comprendre l'impact du spreading factor, on peut noter que si un noeud émet à SF7 pendant 1 an toutes les dix minutes, sa batterie pourra durer un an, alors qu'un noeud émettant à SF12 ne dépassera pas 0.8 ans soit 292 jours.

Data Rate (DR)	Configuration			Physical Bit Rate (bit/s)
	Modulation	Spreading Factor (SF)	Bandwidth	
0	LoRa	SF12	125 kHz	250
1	LoRa	SF11	125 kHz	440
2	LoRa	SF10	125 kHz	980
3	LoRa	SF9	125 kHz	1760
4	LoRa	SF8	125 kHz	3125
5	LoRa	SF7	125 kHz	5470
6	LoRa	SF7	250 kHz	11,000
7	FSK	50 kbit/s		50,000
8–15	Reserved for Future Use			

FIG. 5 – *Relation entre SF et DR*

Le facteur d'étalement est le premier paramètre important que nous considèrerons.

3.3.4.2 Puissance

La puissance du signal se mesure en dBm et permet d'augmenter la portée de transmission. Dans le cas de LoRaWan, elle s'étale de -2dBm à 20dBm. Tout comme le facteur d'étalement, augmenter la puissance revient également à une dépense énergétique plus élevée. La puissance doit donc également être prise en compte.

3.3.4.3 Autres paramètres

Le Spreading Factor et la puissance sont les deux paramètres principaux qui peuvent être modifiés dynamiquement et qui impactent significativement aussi bien la distance d'émission, que la consommation énergétique. Cependant, il existe d'autres paramètres :

- Largeur de bande : Détermine la largeur de la bande passante disponible pour une émission. LoRaWan propose trois largeurs de bandes 125kHZ, 250kHz, 500kHz. Dans les faits, seuls les 125khZ sont utilisés puisqu'avec 500kHz les performances ne sont plus intéressantes et les législations n'autorisent en général pas les 250kHZ.
- Fréquence de la porteuse : Correspond à des canaux de transmission différents.
- Coding Rate : Proportion de bits transmis qui transportent réellement de l'information et est généralement de 4/5 [4]. L'impact de ce paramètre est peu documenté.

3.3.4.4 Coût énergétique

Pour déterminer, si notre réseau est économe, on va considérer la quantité d'énergie dépensée par les capteurs. Celle-ci est proportionnelle au temps du message dans l'air. De ce fait, un message avec un facteur d'étalement plus grand, qui reste plus longtemps en l'air, aura une consommation supérieure. La puissance, qui correspond à la force avec laquelle on transmet le message, impacte aussi la consommation d'énergie.

3.3.5 Collisions

Si on veut un système qui consomme le moins d'énergie possible, on peut se contenter de fixer le facteur d'étalement et la puissance aux moins énergivores possibles. Cependant, lors d'une transmission de message, il est possible que ce dernier ne parvienne jamais à destination et ce pour plusieurs raisons :

- La portée de l'onde n'était pas suffisante pour atteindre la station. Il faut donc, soit ajuster les paramètres si cela est possible, soit rapprocher le capteur.
- Il y a eu une collision due aux paramètres. Cela signifie que deux messages avaient les mêmes paramètres à l'émission et qu'ils se sont percutés, provoquant la perte de ces messages.
- Il y a une capture de message. Il est possible que deux messages avec le même facteur d'étalement et des puissances proches se collisionnent. Dans ce cas, c'est le message avec la puissance la plus haute qui écrase l'autre.

Ce problème de collision implique la nécessité de choisir avec soin les paramètres d'émission.

3.3.6 Classes

Le protocole LoRaWan prévoit 3 classes, A, B et C [3], de communication. Seule la classe A est obligatoire à l'implémentation, tandis que les deux autres sont optionnelles. Il faut étudier ces classes pour définir celle qui est la plus adaptée à notre cas.

La classe A propose une communication bidirectionnelle:

– Le nœud émet une transmission montante vers la passerelle.

– Deux fenêtres RX1 et RX2 sont ouvertes par le nœud pour recevoir la réponse de la passerelle.

– La première fenêtre RX1 utilisera les mêmes paramètres que ceux utilisés par le nœud pour transmettre le message, tandis que la deuxième fenêtre RX2 utilise des paramètres fixes prédéfinis en cas d'échec.

– En dehors des fenêtres, le nœud est comme en veille et ne consomme que l'énergie nécessaire au capteur lui-même.

La classe A permet donc un échange de type ALOHA, c'est-à-dire que les capteurs peuvent émettre un message à n'importe quel moment, ce à quoi l'antenne doit être préparée pour pouvoir répondre dans les temps. Cette classe est certes la plus restrictive, mais consomme beaucoup moins d'énergie que les autres classes.

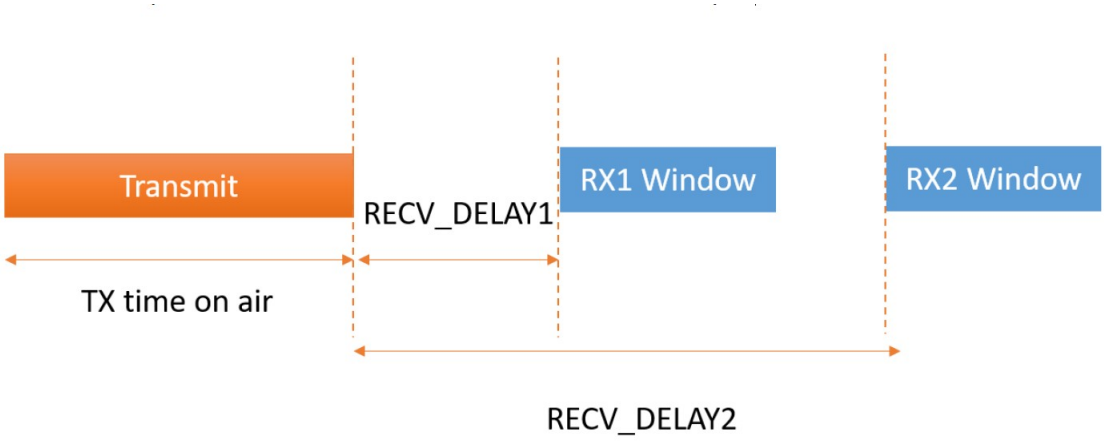

FIG. 6 – *Classe A*

La classe B utilise le principe de balise de synchronisation. Chaque fois qu'un nœud reçoit cette balise, il peut synchroniser son horloge sur la passerelle à laquelle il est connecté. Il faut donc des fenêtres de réception supplémentaires et donc une consommation en hausse.

Les nœuds réglés sur la classe C restent en permanence en écoute, prêts à recevoir les informations de l'antenne, ce qui signifie une consommation en hausse.

La classe A est donc la classe la plus intéressante pour notre problème qui allie efficacité de transmission et consommation.

3.3.7 Trame LoRa

LoRa a été créée dans l'optique de faire fonctionner des appareils avec un débit relativement faible. Il est donc intéressant de voir quelle quantité de données il est possible de transmettre puisque cela nous servira pendant l'apprentissage. La trame se compose de :

– Un préambule utilisé par l'antenne pour verrouiller le signal LoRa.
– Un en-tête avec la taille de la charge utile (le message en lui-même) en octets, un coding rate (CR) et l'activation d'une vérification appelée CRC. La charge utile est comprise entre 1 et 255 octets.
– Le CRC s'il est activé.

FIG. 7 – *Trame LoRa*

Comparé à la tendance moderne d'avoir des quantités énormes de données en transit, ici il faut faire attention à la limitation de la taille des messages. La communication entre l'antenne et les nœuds est en effet limitée et cela d'autant plus avec la classe A qui ne permet pas à la passerelle de communiquer quand elle veut.

4 Analyse de la technologie LoRa

Avant de commencer à faire des expérimentations et à étudier le comportement d'un réseau LoRa, il faut noter plusieurs spécificités.

4.1 Une technologie récente

Une des informations principales à noter sur LoRa est la relative jeunesse de cette technologie. En effet, même si elle date de 2010, LoRa a été très peu étudiée. Hormis une étude au Danemark[12] peu détaillée, l'étude du comportement d'un capteur lors d'un déplacement ou des réseaux à taille humaine[7], il n'y a pas eu de réelles études grandeur nature. Les paramètres permettant de faire fonctionner le réseau sur simulateur sont ceux utilisés sur les précédents simulateurs et peuvent difficilement être vérifiables. Par ailleurs, comme expliqué en 3.3.3, ADR repose sur des heuristiques, mais aucun code ou pseudo-code reportant précisément son comportement n'est trouvable. Les versions trouvées sont en effet toutes différentes et ne collent jamais strictement aux indications vagues données par Semtech. Du fait du peu d'installations LoRa mises en place, les résultats obtenus dans ce rapport n'ont pas été testés en conditions réelles et restent donc à vérifier. Enfin, même si l'on étudie des méthodes de sélection dynamiques de paramètres, cela implique que le capteur soit connecté à une autre plateforme plus puissante, permettant des calculs, et donc plus énergivore, mais cela est aussi peu documenté.

4.2 Les limites et LoRaWAN

Sans analyses approfondies, on peut déjà établir de nombreuses restrictions. LoRaWAN implique un réseau à basse énergie et à taille de message limité. Cela restreint certes les possibilités de l'utilisateur, mais aussi notre marge de manœuvre pour l'apprentissage. D'abord, le capteur n'est pas toujours à l'écoute, il ne peut donc pas dépendre trop de l'antenne. Si on veut utiliser l'antenne pour faire un choix dynamique de paramètres, il faut rester dans les limites d'un protocole Aloha fonctionnant sur le principe de message/réponse. Ensuite, même si l'on parvient à communiquer, la taille des messages est problématique. La quantité d'information à transmettre est fortement restreinte par la limitation de LoRaWAN, mais aussi par le fait qu'une place non négligeable est occupée par le message du capteur(transmission de données de température, de mouvement...). Dans le même temps, on ne peut pas se permettre de réserver un temps pour l'apprentissage puis d'émettre puisque cela résulterait en un réseau inutilisable avant x temps.

4.3 Impact des paramètres

L'algorithme ADR laisse à penser que l'augmentation du facteur d'étalement et de la puissance font grimper proportionnellement la consommation d'énergie et la portée. La documentation laisse à penser que, par exemple, un spreading factor de 8 associé à une puissance de 0 dBm portera plus loin et consommera plus d'énergie que le facteur d'étalement 7 avec 20dBm. Cependant, avec les calculs suggérés et les mesures effectuées en condition réelle, on se rend rapidement compte que cela n'est pas vraiment le cas. La puissance, de par son échelle logarithmique, a peu d'impact sur la distance pour de petites valeurs, mais la fixer à 20dBm peut multiplier jusqu'à 3 la portée pour un facteur d'étalement donné.

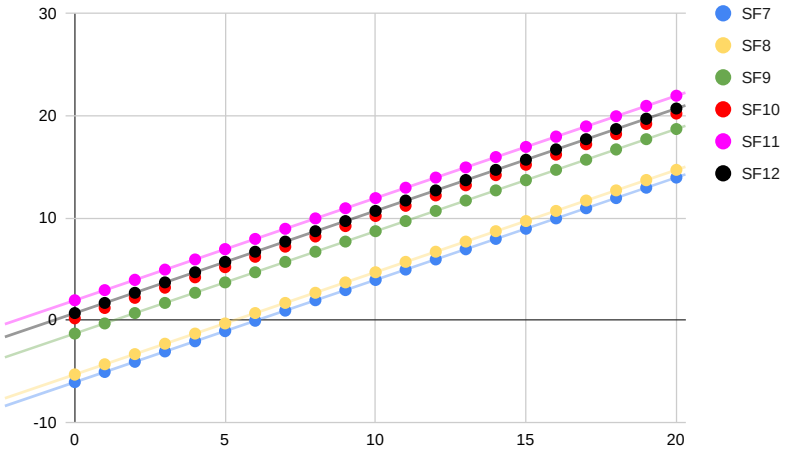

FIG. 8 – *Portée du signal en fonction de la puissance d'envoie par facteurs d'étalement*

Ce qu'on observe sur le graphique c'est l'importance majeure de la puissance, puisque les paramètres ne peuvent pas juste être rangés dans l'ordre croissant de facteur d'étalement mais doivent prendre en compte le couple complet. Ainsi le couple (7, 0) aura approximativement la même portée que le couple (7,4). On note aussi que, dans notre cas, le spreading factor 11 porte plus loin que le facteur 12, et semble donc être plus avantageux puisque moins coûteux en énergie.

Toutes ces informations seront ainsi prises en compte pour la suite.

5 Simulateur

Afin de pouvoir effectuer nos expérimentations sur la technologie LoRa, il nous faut utiliser un simulateur. Ce simulateur nous permettra de mettre en place un réseau de capteurs, une antenne, puis de mesurer la consommation énergétique d'un capteur en particulier ou alors du système entier. Il existe déjà des simulateurs pour les réseaux LoRa, le principal étant LoRaSim.

5.1 LoRaSim

LoRaSim est développé en Python et utilise le framework Scipy. Ce framework est responsable de l'exécution de la simulation, remplaçant, par exemple, un échéancier. Il se décompose en quatre scripts, gérant des cas plus ou moins particuliers comme la possibilité d'avoir plusieurs antennes. Cependant, ce simulateur présente plusieurs limites. D'une part, le programme est très peu modulable, c'est-à-dire qu'il est difficile d'effectuer des modifications dessus sans modifier le comportement global. D'un point de vue pratique, il est développé en Python 2 version antérieure de Python 3. Ensuite, LoRaSim se base sur des calculs physiques qui ne correspondent pas à d'autres standards, notamment la calculatrice LoRa officielle de Semtech [1]. Il est également difficile de rajouter nos propres événements comme le déplacement d'un nœud pendant la simulation. De plus, il manque une gestion de réémission. En effet, LoRaWAN prévoit une réémission jusqu'à sept fois d'un paquet n'ayant pu être transmis et cela est complètement absent. Mais le souci majeur, est l'absence de gestion dynamique des paramètres des nœuds, c'est-à-dire qu'il n'est pas possible de modifier la puissance ou le facteur d'étalement pendant la simulation ce qui est nécessaire pour notre sujet de stage.

5.2 Autres simulateurs

D'autres simulateurs existent et sont disponibles librement comme LoRaFree et LoRaMab. LoRaFree est une version améliorée de LoRaSim, mais comporte les mêmes problèmes que ce dernier. Quant à LoRaMab, il propose une modification dynamique des paramètres, mais son code source est peu pratique à l'utilisation puisqu'il est difficile de rajouter nos propres algorithmes autour de ceux déjà implémentés.

5.3 Simulateur personnalisé

Devant l'absence de simulateur convenant aux besoins du stage, le choix s'est porté sur la réalisation de notre propre simulateur. Il a été réalisé en Python 3 sans utilisation de bibliothèques particulières autres que mathématiques pour les émissions d'ondes, les calculs d'énergie et le fonctionnement des algorithmes d'apprentissage. Le simulateur ne fonctionne plus sur Scipy, mais avec un échéancier classique pour les programmes de simulation, ce qui permet de gérer et de suivre les événements plus simplement. Le programme se décompose en deux parties. Une partie pour la simulation qui comporte les fichiers : Simu, graphic, Event, BS, Node, Battery. Et une autre partie chargée de l'apprentissage : learn, deep_q_network et dqn agent. Les parties simulation et apprentissage sont bien séparées, de sorte que l'on peut ajouter des algorithmes d'apprentissage sans avoir à modifier la partie simulation. Le point de liaison entre les deux parties se situe dans le fichier Event.py, dans la partie traitant de la réception de paquets, puisque c'est à ce moment-là que s'effectue le choix de paramètres pour l'émission.

Le simulateur présente également plusieurs fichiers de configuration. Le premier est le fichier Nodes.txt qui se présente comme suit :

```
rand 100 algo:TS radius:680
10 10 algo:exp3
```

FIG. 9 – *Fichier de configuration node*

Dans la première ligne, 100 correspond au nombre de capteurs, rand correspond à un positionnement aléatoire, algo au type d'algorithme, et radius au rayon autour de la base. 680 signifie donc que chaque capteur peut se trouver entre 0m et 680m autour de l'antenne principale. Dans la deuxième ligne, on place un capteur et un seul, aux coordonnées (10,10) autour de la base. D'autres positionnements ont été implémentés, tel qu'un positionnement en grille et un positionnement en ligne. Le deuxième fichier important est le fichier configAlgo.txt. Il contient plusieurs lignes de la forme suivante : NomAlgorithme Signature

```
static learn.Static()
rand learn.RandChoise()
ucb1 learn.UCB1()
exp3 learn.Exp3()
```

FIG. 10 – *Exemple d'un fichier d'algorithme*

Chaque signature devra être implémentée dans le fichier learn.py.
La dernière partie importante du simulateur est l'affichage de graphiques permettant d'analyser les résultats de la simulation et les fichiers de log des nœuds. Un fichier de log présentera plusieurs lignes contenant : facteur d'étalement, puissance, énergie, paquets transmis, nombre de collisions, nombre de paquets perdus. Chaque ligne correspond à une émission.
Parmi les graphiques, on peut afficher le placement des noeuds :

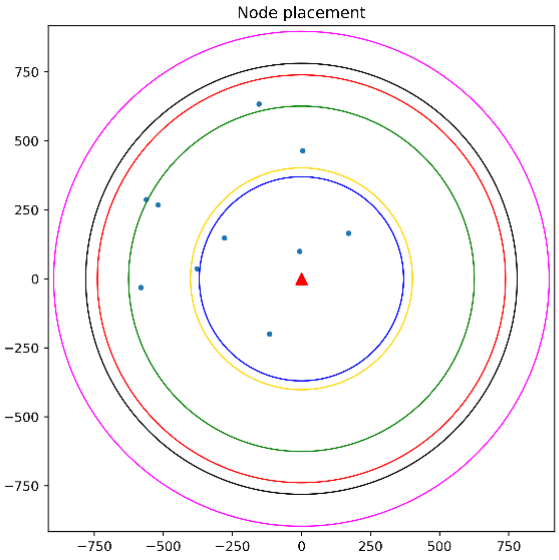

FIG. 11 – *Placement des noeuds autour de l'antenne centrale*

On a aussi le nombre de nœuds par facteur d'étalement, pour étudier le comportement du réseau :

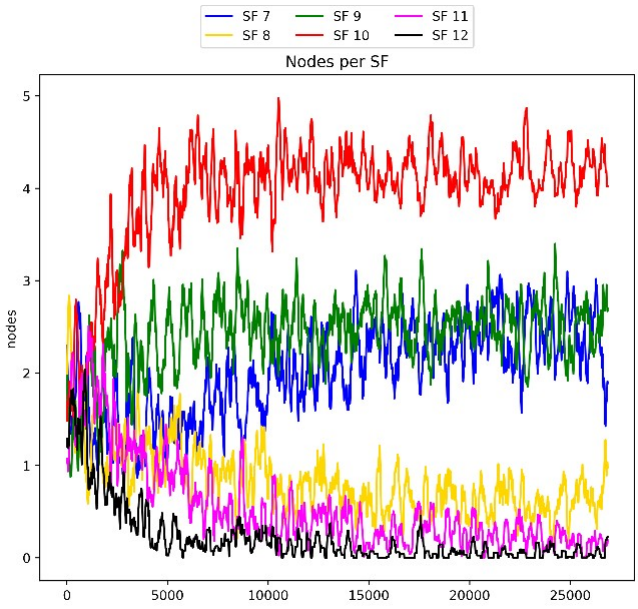

FIG. 12 – *Nœuds par facteurs d'étalement*

Enfin, on peut afficher le nombre de réémissions au cours du temps, afin de constater si notre réseau apprend intelligemment du point de vue de la fiabilité du réseau.

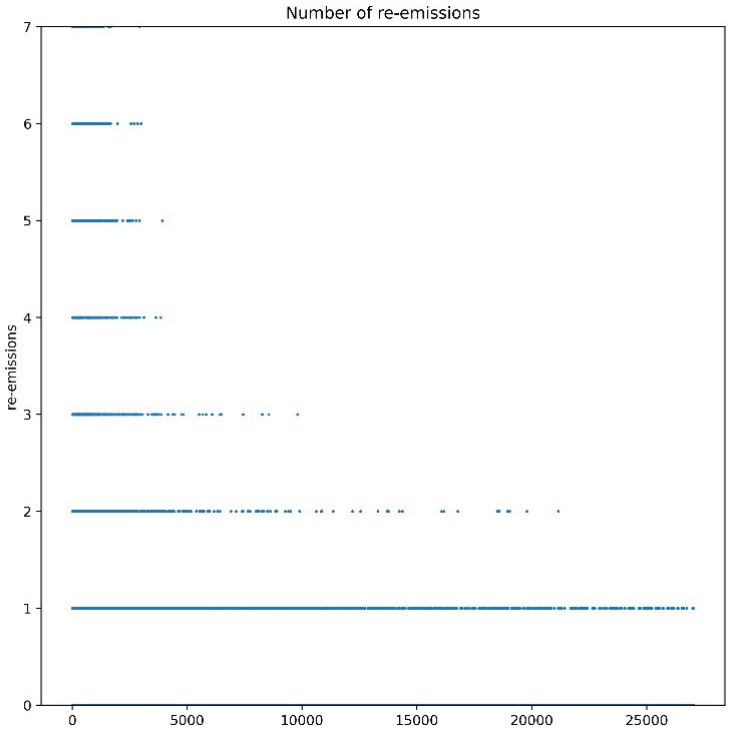

FIG. 13 – *Nombre de réémissions en fonction du temps*

Par exemple, on observe ici un réseau qui, au fur et à mesure du temps, réduit fortement son nombre de réémissions par périodes de temps, passant de 7 réémissions à 1, et ce, de moins en moins souvent.

6 Etat de l'art

Maintenant que l'on dispose d'un outil capable de simuler un réseau LoRa, on peut commencer à s'intéresser aux diverses méthodes déjà existantes pour réduire la consommation de façon intelligente.

6.1 Approche décentralisée

Certains articles traitent déjà d'utilisation de méthodes de bandit pour notre problème avec une topologie similaire [6]. Cependant, ils proposent un bandit de taille réduite, c'est-à-dire avec un choix de 3 ou 4 combinaisons de paramètres alors qu'il en existe plus d'une centaine. Cela pose problème sur le plan énergétique, laissant une marge d'optimisation, mais aussi sur les collisions, puisque plus le nombre de nœuds est important ou concentré, plus le risque de collisions augmentent et plus un nombre élevé de paramètres différents est intéressant. Par ailleurs, dans ces articles, seul un nœud fonctionne avec une méthode de bandit et les autres sur ADR, alors que le fonctionnement de plusieurs nœuds sur le même algorithme de bandit, révèle un problème de concurrence qui n'est pas étudié.

6.2 Approche centralisée

D'autres solutions[8], proposent de définir dès le départ une répartition des paramètres entre les différents nœuds, à l'aide d'heuristiques, là encore. Le souci de cette méthode étant à la fois le manque de précision mais aussi le manque d'adaptation lors d'un ajout d'un capteur, puisque celui-ci n'aura pas été prévu dans l'ordonnancement de départ.

6.3 Approche hybride

ADR a été mis en place par Semtech [11] mais dispose de peu de documentation et une optimisation basée entièrement sur des heuristiques ou de l'empirique. Comme expliqué plus tôt, ADR fait intervenir les deux parties, à savoir, l'antenne, pour réduire la consommation, et le capteur, pour augmenter la puissance. Une heuristique proposée sur l'antenne repose sur le calcul d'une valeur NSTEP, calculée à partir du ratio signal bruit aussi appelé SNR. 20 valeurs de ces SNR sont retenues et seule la valeur maximale est utilisée.

$$NSTEP = \frac{SNR_{max} - SNR_{required} - Margin}{3} \quad (1)$$

où margin est une valeur choisie et le $SNR_{required}$ une valeur dépendant de la table suivante:

SPREADING FACTORS, AND CORRESPONDING RX WINDOWS [7], ANTENNA SENSITIVITIES [8], AND $SINR_{Required}$ [9]

SF	RX windows (ms)	Antenna sensitivity (in dBm)	Required SINR for ADR (in dB)
SF7	5.1	-124	-7.5
SF8	10.2	-127	-10
SF9	20.5	-130	-12.5
SF10	41.0	-133	-15
SF11	81.9	-135	-17.5
SF12	163.8	-137	-20

FIG. 14 – *Table de choix de paramètres pour ADR*

7 Approche décentralisée et apprentissage par renforcement

On peut comprendre que la station dispose de moins d'informations qu'un nœud pour choisir des paramètres. En effet, les communications de l'antenne vers les nœuds étant limitées, il est plus facile pour un nœud de sélectionner lui-même ses paramètres. On choisit donc d'étudier une approche décentralisée. Pour permettre aux nœuds de modifier dynamiquement les paramètres afin d'établir un bon compromis entre consommation et transmission, on va faire appel à l'apprentissage par renforcement.

7.1 Apprentissage par renforcement

Le renforcement est la façon d'apprendre à un agent comment il doit se comporter dans un environnement. Comme le nom de cette méthode l'indique, il va commencer son activité en commettant de nombreuses erreurs. Au cours du temps, il va chercher à se rapprocher des meilleurs choix en comprenant que certaines décisions sont plus bénéfiques que d'autres. Cet agent va être récompensé par l'environnement dans lequel il évolue, et son but est d'augmenter ses récompenses. Pour cela, il va devoir accumuler des données propres à l'environnement, ce qui va se faire par le biais de l'exploration. On peut y faire une analogie à l'apprentissage humain. Ainsi le problème de l'apprentissage est modélisé par:

– un agent qui prend des décisions

– des actions représentant les choix de l'agent

– des récompenses qui sont les conséquences des actions

– différents états qui définissent le champ d'action suivant

– un environnement, comme par exemple un jeu

Le but pour l'agent sera d'obtenir un maximum de récompenses tout en respectant le modèle et l'environnement définis. Une idée serait que chaque nœud essaie de cohabiter avec les autres en cherchant quels paramètres seraient les plus efficaces pour lui : c'est le problème de bandit.

7.2 Bandit

Le problème de bandit manchot est défini comme suit : Soit un agent A face à un problème pour lequel il dispose de K stratégies. Il doit déterminer quelle est la meilleure stratégie à choisir sachant que le gain n'est pas connu à l'avance. Un exemple courant est l'exemple des machines à sous, où un joueur doit choisir sur quelle machine jouer pour maximiser ses gains. Il s'agit d'un type d'apprentissage par renforcement. Ce problème met en exergue la notion de compromis exploration/exploitation. En effet, contrairement à de l'apprentissage par renforcement plus classique, ici on ne peut pas se contenter d'effectuer une succession de tests aléatoires jusqu'à déterminer le meilleur résultat, mais il faut que tout au long de l'apprentissage on puisse minimiser la perte.

Algorithme 3 : Algorithme de bandit générique

```
tant que i ≤ tempsMax faire
    Choisir une stratégie k parmi K;
    Appliquer la stratégie à l'environnement;
    Mettre à jour le vecteur de stratégies;
fin
```

L'ensemble des stratégies est généralement représenté par un vecteur de stratégie. Chaque stratégie, ou bras, est associée à une valeur qui permettra de choisir l'action suivante. Cette valeur est calculée différemment selon l'algorithme de bandit choisi.

FIG. 15 – *Vecteur de stratégie d'un bandit*

Il existe plusieurs algorithmes, notamment UCB1, EXP3 ou Thompson Sampling, dont les performances ont déjà été prouvées dans des articles. La principale différence entre ces méthodes est la façon dont est choisie le meilleur bras. Dans ce rapport, nous présenterons deux algorithmes de bandit avec deux approches différentes : Thompson Sampling et UCB1

7.2.1 Thompson sampling

Thompson Sampling [9] est une des méthodes de bandit les plus connues mise au point par William R.Thompson. Elle est basée sur la loi bêta pour le choix de la stratégie et la mise à jour des valeurs de bandit. Chaque bras est associé à un couple de paramètres (α,β) correspondant aux paramètres d'une loi bêta. Pour sélectionner un bras, on va donc calculer la loi bêta associée au couple de paramètres pour chaque bras, tirer une valeur au hasard et prendre le bras associé à la valeur la plus élevée. Pour mettre à jour notre bandit, on ajoute le reward à α et un moins le reward à β

$$\alpha = \alpha + reward; \beta = \beta + (1 - reward) \tag{2}$$

La loi bêta ayant une espérance de $\frac{\alpha}{\alpha+\beta}$ chaque fois que l'on a un gain, on augmente l'espérance en augmentant α sinon on la diminue en augmentant β.
De manière plus formelle, on considère un contexte χ, un ensemble d'actions A et des récompenses R. De manière successive, le joueur reçoit un contexte $x \in \chi$ effectue une action a et reçoit une récompense r. A chaque action est associée un ensemble de valeurs α et β. Ces valeurs sont mises à jour selon la reward perçue au tour précédent. Le choix d'action s'effectue comme suit :

– On calcule la distribution β pour chaque couple (α,β)
– Pour chaque stratégie on tire une variable aléatoire selon la distribution
– On sélectionne la stratégie avec la plus grande valeur

7.2.2 UCB1

UCB1 calcule une valeur appelée UCB_{Value} qui correspond à une moyenne agrémentée d'un rapport aux nombres de fois que l'on a utilisée le bras[2]:

$$UCB_{Value} = \hat{x}_{it} + \sqrt{\frac{2 \ln t}{t_i}} \tag{3}$$

avec $\hat{x}_{it}$ la moyenne pondérée des récompenses et t_i le nombre de fois où l'on a utilisé le bras i.
UCB1 va donc choisir en moyenne l'action avec la meilleure récompense, mais le deuxième terme va permettre de chercher dans les actions un peu moins utilisées avec une reward suffisamment proche.
L'exécution est donc la suivante :

– Choisir une action qui maximise la UCB_{value}
– Appliquer ce choix
– Mettre à jour la moyenne avec l'équation (11)

7.2.3 Noeud et Bandit

Dans notre cas pr écis, on d écide d' établir notre bandit comme suit. Le bandit s'effectue sur un vecteur contenant l'ensemble des couples (SF, Pw) possibles avec $7 \leq SF \leq 12$et $0 \leq Pw \leq 20$

FIG. 16 – *Vecteur de statégies*

7.3 Fonction de coût

On utilise une fonction de perte. En effet, on cherche à minimiser la d épense énergétique ce qui implique de ne pas avoir trop de collisions puisque cela rallongerait le temps actif de l'appareil.

Si le message est correctement transmis et que le noeud rec¸oit l'acquittement :

$$Cost = 1 - (energyCost/energyMax) \tag{4}$$

Sinon :

$$Cost = penalty \tag{5}$$

où energyCost correspond à l' énergie n écessaire pour émettre un message selon les param ètres (Sf, power) choisis, energyMax l'énergie pour transmettre un message avec (12,20) qui dans notre cas est le couple le plus énergivore et penalty la pénalité que l'on fixe au cas par cas selon l'algorithme.

7.4 Q-Learning

Dans le cas d'un bandit, on ne regarde que les actions que l'on peut effectuer sans tenir compte de l'environnement. Cependant, on pourrait préciser le choix de paramètres en prenant en compte à quelle réémission on se situe. Cela signifie que l'on prend en compte l' état dans lequel on est : on peut alors envisager le Q-Learning.

Le Q-Learning est une forme d'apprentissage par renforcement. Son fonctionnement est principalement basé autour d'une fonction appel ée fonction Q. La fonction Q est d éfinie de la fac¸on suivante : $Q(S_t, A_t)$ avec S l'état et A l'action qui doit être effectuée.

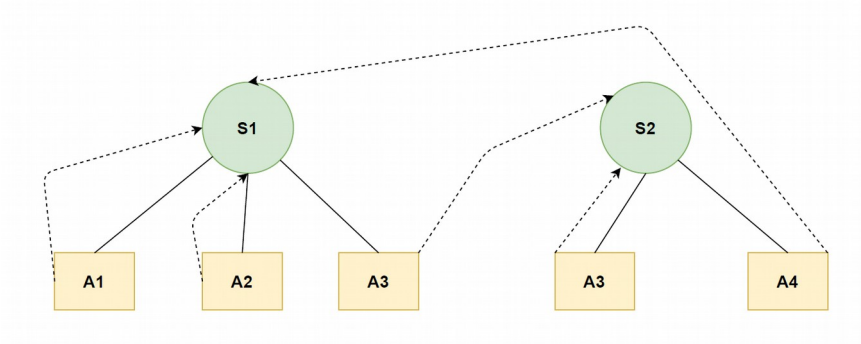

FIG. 17 – *Graphique état-action*

Cette fonction Q permet de déterminer le nombre de récompenses que l'on va obtenir dans le futur. Mais pour avoir cette fonction, il nous faut un environnement et différentes actions possibles dans cet environnement. A chaque état que l'agent parcourt, il obtient une récompense à l'instant t+i que l'on notera R.

La fonction Q nous donne l'espérance du nombre de récompenses que l'on peut obtenir à partir d'un état donné et d'une action choisie par l'agent. L'ensemble des données est contenu dans ce que l'on appelle une Q-Table qui correspond à une matrice associant un état-action à sa valeur d'espérance.

$$Q(s_t,a_t)_{new} = Q(s_t,a_t)_{old} + \alpha * [r + \gamma * \max_{a_{t+1}}[Q(s_{t+1},a_{t+1})] - Q(s_t,a_t)_{old}] \quad (6)$$

1. max : on prend la valeur de la meilleure action future.
2. α : permet de moyenner la modification sur beaucoup de transitions. (learning rec)

γ peut être fixée sur des valeurs plus ou moins grandes selon l'importance des conséquences de nos actions ou être nulle si hypothèse de finalité, il y a.

Algorithme 4 : Q-Learning

```
Initialiser Q[s,a], pour tous les états et toutes les actions à une valeur aléatoire.
tant que t < tempsMax faire
    r = aleaUniformVal();
    si r >  alors
        a = aleaAction();
    sinon
        a = max(Q[s]);
    fin
    Exécuter a;
    Observer reward r et l'état s';
    Q[s,a] := Q[s,a] + α[r + γ max_a' Q(s',a') − Q(s,a)];
    s = s';
    a = a';
fin
```

7.5 Amélioration envisageable

On pourrait essayer d'améliorer le bandit en limitant la taille de celui-ci. Pour cela, la station ou l'antenne, sera un support pour communiquer des paramètres. Au départ, le bandit ne contiendrait que (12,20) de manière à établir la communication. Dès lors, la station transmet, grâce à une grille de recommandation, les paramètres optimaux. À partir de cela, on construit un vecteur de paramètres comprenant le SF supérieur et inférieur et une puissance de plus ou moins 5dBm. Le bandit s'effectuera sur ce vecteur. Après 7 essais infructueux, on passe à (12,20) de manière à demander de nouveaux paramètres. La limite est de savoir si une telle grille est effectivement réalisable. Si tel est le cas, cela signifierait que les nœuds n'auraient plus qu'à se soucier des collisions et plus de la transmissibilité.

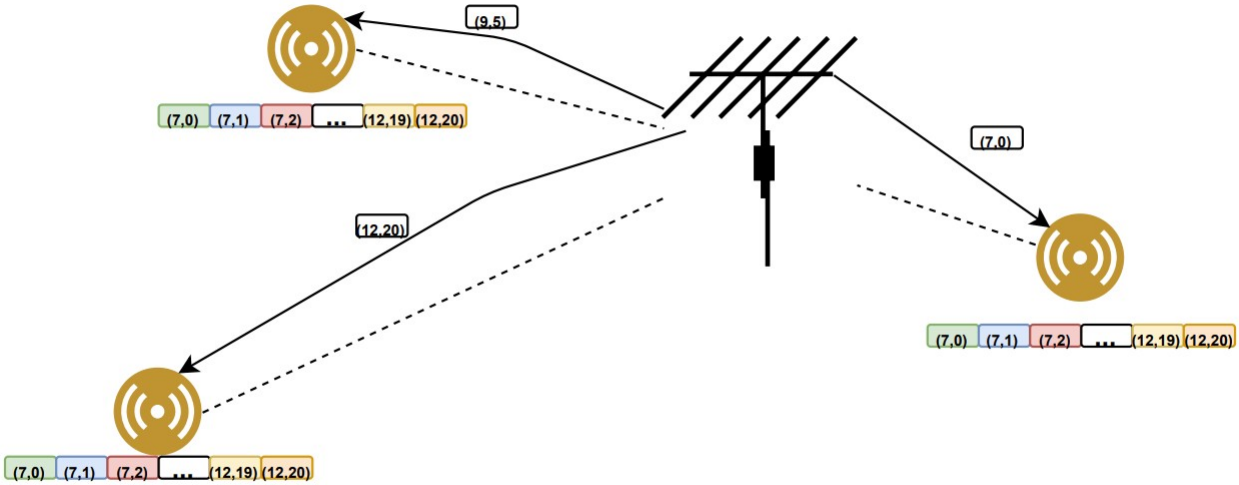

FIG. 18 – *Antenne qui recommande aux noeuds des paramètres*

7.6 Comparaison des différents algorithmes

Avant d'observer les résultats, on peut mettre en avant la compatibilité des algorithmes avec notre problème:

– UCB1 : Cet algorithme semble intéressant dans la mesure où il détecte le bras le plus efficace, mais va continuer à explorer les bras qui ont quasiment les mêmes rendements. Cela est d'autant plus pertinent, que l'on cherche à déterminer le paramètre optimal mais aussi à limiter les collisions, et donc éventuellement, à opter pour un choix un peu moins performant mais moins sujet aux collisions.

– Thompson Sampling : Thompson Sampling fonctionne autour de la distribution beta. Cela implique donc le tirage aléatoire d'une valeur et ainsi une part de hasard lors du choix de paramètres, même si cette part diminue au cours du temps, ce qui pourrait causer une instabilité dans la convergence du système global.

– Q-Learning : Le Q-Learning est basé sur un ensemble d'état-action. Il faut ainsi définir des états, même si, ici, le problème s'y prête plutôt mal. De plus, l'exploration par -greedy est rendue compliquée du fait d'un besoin d'optimalité pendant la phase d'apprentissage en elle-même. Il faut alors se pencher vers d'autres méthodes d'exploration comme Holtzmann qui n'ont pas été essayées dans le cadre de ce stage. Un décroissant se sied mal à notre réseau, puisqu'il est difficile de dire quand il est nécessaire de réexplorer, ce que d'autres méthodes incorporent déjà.

8 Résultats

On réalise des mesures de performance des différentes approches et dans diverses conditions afin d'établir l'efficacité selon le contexte. On va dans un premier temps comparer les algorithmes d'une approche décentralisée avec suggestion de paramètres pour déterminer quelle méthode gère le mieux les collisions seules. Ensuite, on observera la capacité d'un algorithme à s'adapter à un déplacement du nœud avant de voir la vitesse de convergence des algorithmes dans des conditions normales. Sauf mention contraire, les nœuds sont placés aléatoirement autour d'une base dans un rayon maximale de

680m puisqu'à cette distance il est déjà nécessaire d'utiliser les paramètres maximales à savoir (11, 20), en effet ces paramètres sont pour un milieu en intérieur.
L'impact des paramètres d'apprentissage est à ne pas négliger, aussi bien la pénalité, que le pas d'apprentissage. En effet, pendant la phase de test, des écarts significatifs de performances ont été mesurés pour le même algorithme mais avec des paramètres différents pour des problèmes différents. Il conviendra donc de prendre une décision quant aux paramètres selon la topologie. Ici, les paramètres sont choisis de manière à ce que l'algorithme présente des résultats intéressants sur l'ensemble des tests, et ne nous les faisons donc pas varier entre chaque test ou entre différents types de tests. Dans tous les tests, on considère des capteurs ayant la même périodicité d'émission à savoir une émission toutes les 30 minutes, hors réémissions et jusqu'à 7 réémissions en cas d'échec de transmission.

8.1 Gestion de collisions

Chaque algorithme est mis dans les conditions suivantes : 500 nœuds qui sont conseillés (approche hybride), c'est-à-dire que la taille de leur bandit est réduite aux paramètres dont on sait qu'ils vont permettre la transmission des données. Ainsi, un échec de transmission correspond forcément à une collision entre paquets émis. Les mesures s'effectuent sur l'équivalent de 50 jours.

Algorithme	Taux de collisions	Consommation(mAh)
ADR	30%	42167
Thomspon Sampling	4.4%	10507
UCB1(lr=0.01)	4.3%	13257
EXP3(lr=0.01	4.5%	13501
Qlearning($=0.1$)	5.1%	17053

TAB. 1 – *Gestion des collisions des algorithmes*

On peut voir sur le TAB. 1 qu'ADR gère assez mal les collisions avec 30% de collisions et le résultat le plus énergivore. Ensuite, les algorithmes s'en sortent de manière assez semblable mais Thompson Sampling présente l'avantage d'avoir la consommation énergétique la moins élevée pour à peu près le même nombre de collisions que les autres.

8.2 Déplacement d'un noeud

Pour ce test, l'objectif est de savoir quel algorithme donne la convergence la plus rapide après qu'un nœud ait été bougé. On part d'une position rapprochée soit 1 mètre de la base. Puis après 3411742279ms soit 39 jours environ, le nœud est déplacé brusquement à 600m (sans progression graduelle). On mesure alors le nombre d'émissions nécessaires avant qu'il y ait convergence vers un paramètre. De par sa nature, il est difficile d'évaluer la convergence d'ADR. En effet, l'algorithme augmente et diminue régulièrement ses paramètres, donc les résultats ne sont pas présentés ici.

Algorithme	Nb émissions avant convergence)
Thomspon Sampling	1537
UCB1(lr=0.01)	360
EXP3(lr=0.01	313
Qlearning($=0.1$)	130

TAB. 2 – *Gestion du déplacement d'un noeud*

TAB. 2 montre que UCB1 et EXP3 montrent des performances similaires. Tandis que Thompson Sampling sous-performe avec un temps de convergence plus long. Dans ce test, la valeur de p énalité est prépondérante et on obtient des performances différentes avec des valeurs différentes. Cependant, si on diminue la valeur de la p énalité, des difficult és de convergence peuvent appara ître tandis que si on l'augmente trop, certains paramètres seront moins accessibles lors d'un changement de position.

8.3 Résultats d'une approche centralisée par DQN

Dans l'optique de comparer une approche centralisée aux approches décentralisées, on met en place un Deep-Q-Network. Pour rappel, dans une approche centralisée, l'antenne est chargée d'apprendre les paramètres (Voir annexes) Pour 100 capteurs, fonctionnant pendant 50 jours avec lr = 0.00008,$\gamma = 0$et décroissant, on observe les résultats suivants :

Taux de collisions	Consommation(mAh)
10%	3521

TAB. 3 – *Résultats du DQN*

Ces résultats seront analysés dans la section 9.4.

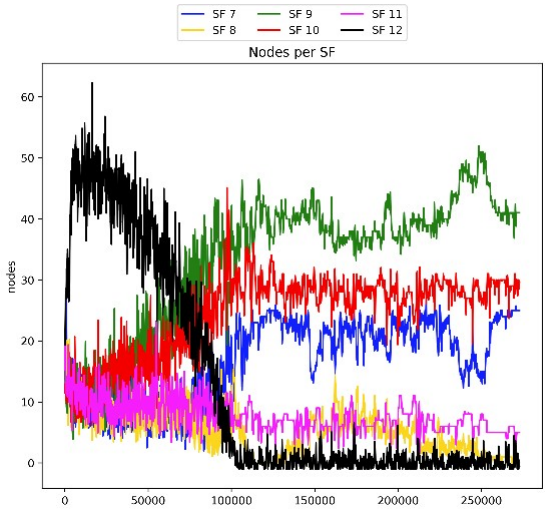

FIG. 19 – *Répartition des facteurs d'étalement choisis par paquets*

On peut voir que le DQN converge relativement rapidement, pour un syst ème centralis é, dans le choix du facteur d'étalement. Il faut en moyenne 100 000 émissions pour que le système soit stable soit 1000 émissions par capteurs.

8.4 Mesures pour une simulation normale

On considère cette fois-ci des conditions classiques pour chaque algorithme, c'est-à-dire 100 nœuds répartis aléatoirement jusqu'à 680m autour de la base de mani ère uniforme. Chaque nœud émet toutes les 30 minutes hors r éémissions avec un maximum de 7 r éémissions de message. La simulation est effectuée sur 50 jours.

Algorithme	Taux de collisions	Consommation(mAh)
ADR	28%	5982.24
Thomspon Sampling	11.3%	2300
UCB1(lr=0.01)	3.1%	2100
EXP3(lr=0.01	7%	2593
Qlearning(= 0.1)	12.1%	3245
DQN	10%	3521

TAB. 4 – *Test classique du réseau*

Ce qu'on observe ici, ce sont d'abord les mauvaises performances d'ADR. Ensuite, on peut comparer l'approche centralisée et l'approche décentralisée en remarquant qu'en général, une approche décentralisée donne de meilleurs résultats même si l'approche centralisée ne peut pas être considérée comme décevante. Pour l'approche décentralisée, on constate que les méthodes de bandits sont supérieures à Q-Learning qui pèche de par la difficulté d'établir une politique d'exploration aussi efficace que les méthodes de bandit. Ici, Q-Learning est en -greedy ce qui implique des changements de paramètres impromptus.

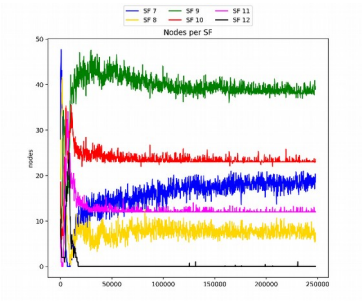

FIG. 20 – *UCB1*

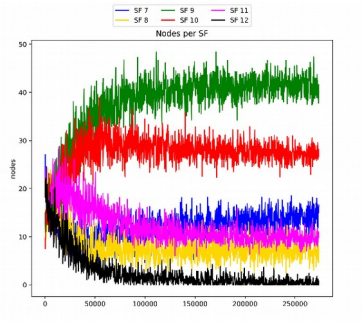

FIG. 21 – *Thompson Sampling*

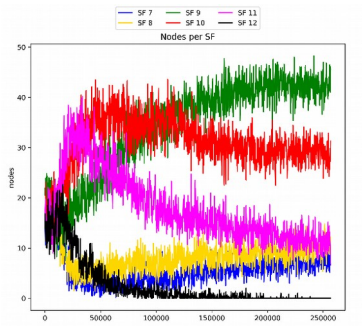

FIG. 22 – *Exp3*

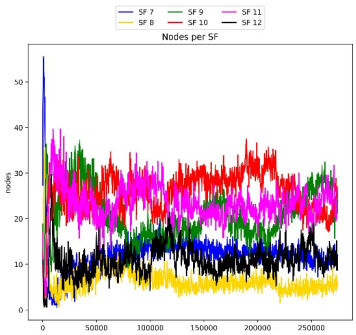

FIG. 23 – *Q-Learning*

Ces graphes représentent la répartition des nœuds par facteur d'étalement. Cela nous permet d'analyser à quelle vitesse converge le système de manière globale et pas seulement localement. On peut voir qu'UCB1 est celui qui converge le plus rapidement avec environ 45 000 émissions avant de converger, suivi de Thompson Sampling en un peu plus de 50 000 puis Exp3 et Q-Learning.

8.5 Analyse et interprétation

8.5.1 Approche décentralisée

Dans un premier temps, il semble clair que le bandit l'emporte sur le Q-learning. En effet, l'exploration est problématique avec ce dernier et par exemple, maintenir un -greedy provoque des instabilités dans le réseau, même si globalement les résultats sont plutôt satisfaisants.

Si on compare les différents algorithmes de bandit, UCB1 et Thompson Sampling se démarquent clairement. Le premier montre la meilleure convergence, puisqu'il lui suffit de tester une seule fois toutes les stratégies pour définir la meilleure et ensuite jongler autour de cette stratégie si jamais il y a des collisions. Thompson Sampling se montre performant lorsque le bandit est restreint à un ensemble de paramètres qui portent sur l'antenne. Il est donc le meilleur en gestion de collisions pures. Les résultats montrant que Thompson Sampling converge lentement après déplacement (TAB. 2) doivent prendre en compte que lorsque l'algorithme doit choisir, sans restrictions, une action parmi un large choix, alors il a du mal à converger. Cela est dû à la distribution bêta qui n'accepte pas de valeurs négatives et donc de pénalité en tant que reward. Lors d'une collision, le seul moyen de pénaliser l'action est d'attribuer une pénalité de 0, là où les autres algorithmes permettent des pénalités négatives comme -10 par exemple. Thompson Sampling serait particulièrement adapté à un système hybride où l'antenne transmettrait un ensemble de paramètres à partir duquel le capteur effectue son choix de paramètres.

Enfin, si on met à part le Q-Learning en epsilon-greedy et ADR, toutes ces méthodes donnent de bons résultats. Ici les valeurs comparées mettent l'accent sur très peu de temps et pourtant on voit que les écarts ne sont pas si significatifs. Sur le très long terme, c'est-à-dire dès un an, les écarts d'énergie et de collisions deviennent négligeables voire inexistants.

8.5.2 Approche centralisée

L'approche centralisée ne semble pas autant en recul qu'on ne pourrait l'espérer. Le DQN peut fonctionner en ne transmettant qu'un minimum d'information, à savoir l'énergie consommée, ou un indice de cette consommation, du capteur vers l'antenne, et, un couple de paramètres, de l'antenne vers le capteur, ce qui devrait être possible avec 255 octets. La gestion des collisions reste raisonnable même si inférieure à des approches décentralisées. Enfin, la convergence est aussi rapide voire meilleure que le Q-Learning.

9 Conclusion et Perspectives

Au cours de ce stage, j'ai eu l'occasion de découvrir la technologie LoRa qui permet de transmettre de faibles quantités de données provenant de capteurs via un principe d'étalement de spectre. J'ai étudié et mis en place différents algorithmes d'apprentissage par renforcement afin de trouver un moyen de proposer de manière dynamique des paramètres de transmissions. Ces algorithmes présentent une amélioration par rapport à l'algorithme de base qui est ADR. UCB1 et Thompson Sampling se détachent quand il s'agit d'une approche décentralisée, même si une approche centralisée via du Deep Reinforcement Learning reste possible et avec de bonnes performances.
Cependant, tous ces résultats doivent être testés en conditions réelles mais aussi dans des configurations différentes, c'est-à-dire à l'intérieur ou à l'extérieur, en ville ou en campagne, en milieu dense ou peu dense. Un système hybride alliant la connaissance de l'antenne et celle du capteur serait probablement la méthode apportant le meilleur compromis. Dans ce cas, l'étude et la recherche d'un algorithme qui optimiserait au mieux la gestion de collisions sur le nœud serait intéressant.
Enfin, pour l'approche centralisée, d'autres algorithmes de Deep Reinforcement Learning pourraient être étudiés tels que Soft Actor Critic qui est connu pour avoir souvent de bonnes performances.

10 Annexes

10.1 Approche centralisée

Dans l'approche centralisée, c'est l'antenne qui va apprendre quels paramètres les capteurs vont utiliser. Pour cette approche, on utilisera du Deep Q-Learning.

10.1.1 Deep reinforcement learning

Les méthodes d'apprentissage par renforcement classiques, telles que le Q-Learning sont basées sur un espace état-action. Il faut ainsi pouvoir segmenter et représenter ces états-action. Cela ne pose pas de problème pour des jeux simples. Cependant, dans le cas d'un réseau plus compliqué avec beaucoup plus d'états où il faut pouvoir représenter les différents paramètres de réception de message, cela devient rapidement difficile. Les choses se compliquent encore lorsque ces états sont dans un espace continu comme par exemple dans un jeu vidéo, où le personnage peut se déplacer librement. Pour représenter ces états, on a alors recours à un réseau de neurones en parallèle de l'apprentissage par renforcement, ce que l'on appelle du deep reinforcement learning.

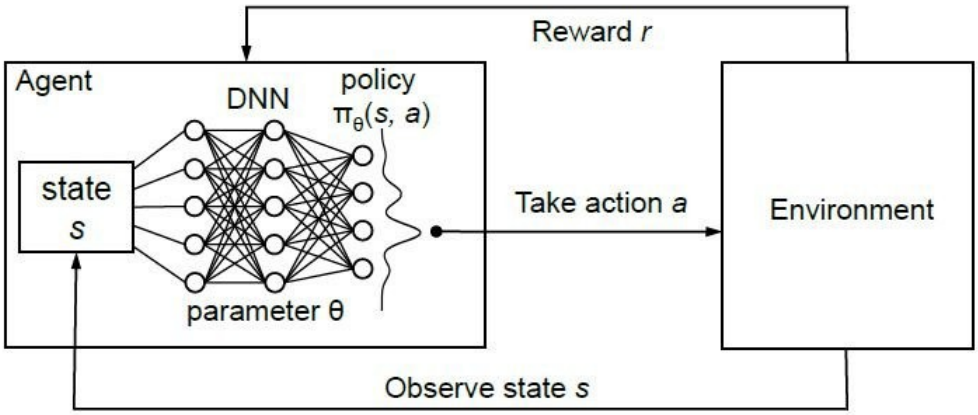

FIG. 24 – *Deep Reinforcement Learning*

10.1.2 Dueling Deep Q network

Le Deep Q-learning est une combinaison de Q-Learning et de réseau de neurones. Comme expliqué précédemment, le problème du Q-Learning est la représentation de l'espace état-action. L'idée est ici d'introduire un réseau de neurones en amont du choix de l'action. Le réseau de neurones sera entraîné jusqu'à ce qu'il soit en mesure de prédire, à partir de données en entrées, l'ensemble des reward pour les actions disponibles. L'apprentissage s'effectue donc à l'aide du reward obtenu qui est calculé via l'équation de Bellman :

$$Q(s_t, a_t)^\pi = r + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1})^\pi \quad (7)$$

Le choix s'effectue alors de manière -greedy avec :

– Soit une action aléatoire pour favoriser l'exploration des différentes actions et améliorer la qualité des actions

– Soit l'action qui maximise l'espérance de gain pour avoir le meilleur résultat possible : c'est l'exploitation

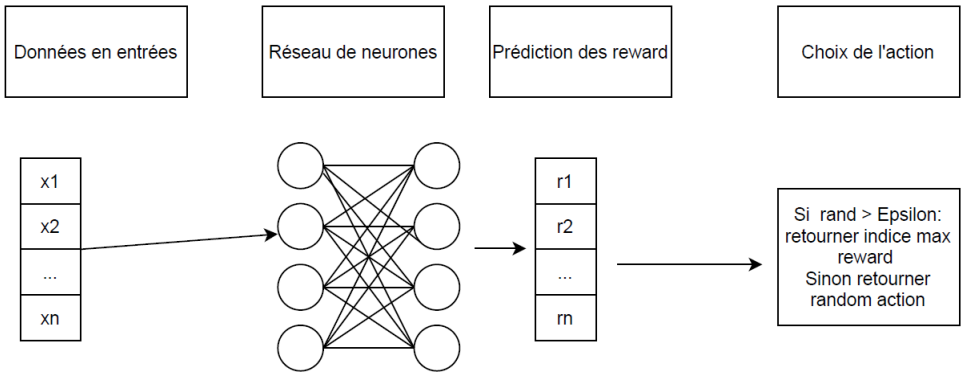

FIG. 25 – *Deep Q Network*

L'erreur quadratique moyenne permet de calculer l'erreur lors de la prédiction de la reward, par l'équation suivante :

$$L_i(\theta_i) = E_{(s,a,r,s')\sim U(D)}[(r + \gamma max_{a'}Q(s',a',\theta) - Q(s,a,\theta))^2] \quad (8)$$

Cette fonction de perte est utilisée pour optimiser le réseau de neurones.

L'originalité du Dueling DQN sur un DQN normal est le remplacement de la prédiction en sortie par deux valeurs appelées la Valeur V et l'Avantage A. Ainsi le flux de la dernière couche est divisé en deux. V permet de prédire une valeur associée à l'état tandis que A donne l'avantage pour chaque action. Pour prédire une action, on applique la formule suivante :

$$Q(s_t,a_t) = V(s_t) + (A(s_t,a_t) - mean(A(s_t,a_t'))) \quad (9)$$

L'avantage du dueling sur le DQN normal est la vitesse de convergence qui est primordiale dans notre problème.

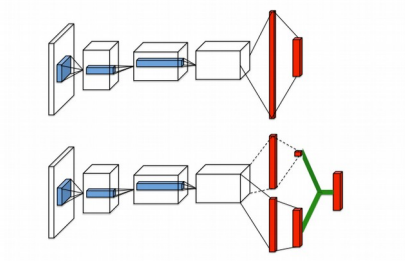

FIG. 26 – *Simple DQN et Dueling DQN*

10.1.3 Méthodologie

10.1.3.1 Utilisation de l'antenne

L'antenne dispose d'un DQN servant à transmettre des paramètres basées sur le Spreading Factor utilisé et le RSSI du message. Il transmettra un couple de paramètres (SF, Pw). L'antenne pourrait être préentraînée au préalable, puisque le Deep Q Network met un certain temps à converger, mais ce ne sera pas le cas ici. Il est envisageable de réutiliser les résultats du réseau de neurones pour prédire des paramètres rapidement.

10.1.3.2 Structure du réseau de neurones

Le réseau choisi se présente comme un réseau à 3 couches comprenant : une couche avec une fonction d'activation tanh, une couche linéaire sans fonctions d'activation, une couche SiLU et une couche linéaire en sortie. En entrée, la station prend en compte la valeur du RSSI qu'elle a calculé, la sensibilité qu'elle a utilisée et le facteur d'étalement correspondant. Mais aussi les sinus et cosinus du RSSI et de la sensibilité. En sortie, elle renvoie le couple de paramètres qu'elle définit comme optimal. Chaque couche contient 96 neurones qui correspondent aux 96 actions possibles. En effet, certaines valeurs de puissance ont été retirées (-2, -1; 3, 4, 5 et 7, 8) puisque, bien que les utiliser, fournisse une portée moindre, la consommation énergétique ne s'en trouve pas impactée. Il est donc inutile de les considérer dans le cadre de l'antenne. La fonction d'optimisation est AdamW, une variante d'Adam qui permet une convergence plus rapide en appliquant des coefficients pour les poids. La learning rate est fixée à 0.00008, mais étant donné qu'AdamW fait varier cette valeur durant l'apprentissage, ce learning rate est en fait une borne supérieure du learning rate. La fonction d'erreur correspond à l'erreur quadratique moyenne qui est généralement utilisée pour les DQN.

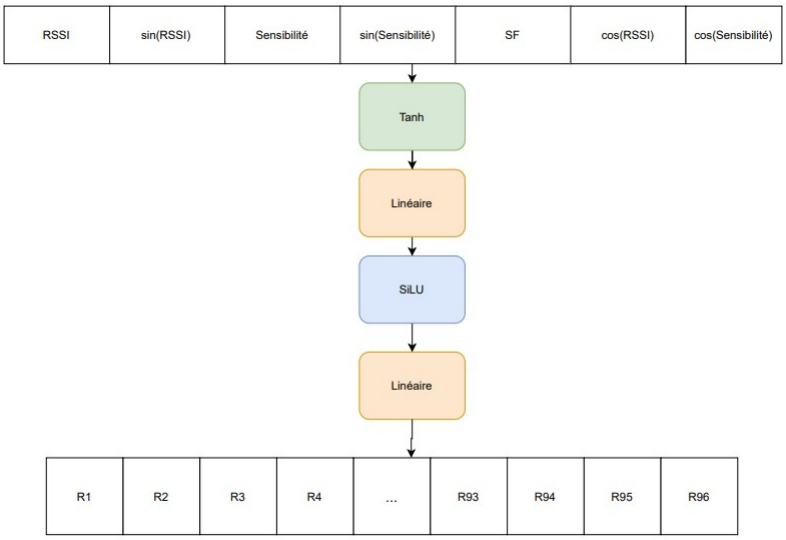

FIG. 27 – *DQN*

10.1.3.3 Fonctionnement

Le DQN se comporte comme suit : Les capteurs n'effectuent pas de bandit mais un algorithme simple. Ils disposent de deux jeux de paramètres, un qui sera fixé par l'antenne et un autre statique fixé à (12,20). Le premier sera utilisé en priorité et permettra de voir si les paramètres fonctionnent et permettent une bonne transmission du message, qui contiendra la quantité d'énergie dépensée pour envoyer le message. L'antenne recevra alors ce reward et mettra à jour le DQN. Si ce paramètre ne fonctionne pas, alors on utilise le second jeu de paramètres (12,20). Si c'est le cas, l'antenne sait que les paramètres n'étaient pas efficaces et appliquent donc une forte pénalité de -10.

Références

[1] In: (). DOI: https : / / sx1272 - lora - calculator . software . informer . com download/.

[2] In: (). DOI: https://www.math.univ-toulouse.fr/ ~jlouedec/demoBandits.html.

[3] LoRa Alliance. In: (). DOI: https://lora-alliance.org/about-lorawan/.

[4] Konstantin Mikhaylov, Juha Petäjäjärvi, and Jussi Haapola. "D2D communications in LoRaWAN Low Power Wide Area Network: From idea to empirical validation". In: ().

[5] The things network. In: (). DOI: https : / / www . thethingsnetwork . org / docs / lorawan/spreading-factors/.

[6] "Node-based optimization of LoRa transmissions with Multi-Armed Bandit algorithms". In: (). DOI: https : / / www . researchgate . net / publication / 326112741 _ Node - based_optimization_of_LoRa_transmissions_with_Multi-Armed_Bandit_ algorithms.

[7] Gianni Pasolini et al. "Smart City Pilot Projects Using LoRa and IEEE802.15.4 Technologies". In: ().

[8] Brecht Reynders, Wannes Meert, and Sofie Pollin. "Power and Spreading Factor Control in Low Power Wide Area Networks". In: (). DOI: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6264067/.

[9] Daniel J. Russo et al. In: (). DOI: https://web.stanford.edu/ ~bvr/pubs/TS_Tutorial.pdf.

[10] Semtech. In: (). DOI: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan.

[11] Semtech. "Understanding ADR". In: (). DOI: https://lora- developers.semtech.com/documentation/tech-papers-and-guides/understanding-adr.

[12] Benny Vejlgaard et al. "Interference Impact on Coverage and Capacity for Low Power Wide Area IoT Networks". In: (). DOI: https://doi.org/10.1109/WCNC.2017.7925510.