Rapport de stage

Dans le laboratoire :

Laboratoire David de l'université de Versailles Saint-Quentin-en-Yvelines

Partenaire du réseau francilien en sciences informatiques (RFSI)

Ce projet a été partiellement financé par la région

Région Paris Ile-de-France via une subvention du DIM RFSI (projet EPINE)

Développeur logiciels et machine Learning

Mohamed Sofiane KASBADJI

Étudiant en Master 1 informatique

Tuteur : Lynda MOKDAD

2020/2021

Université Paris-Est Créteil

Remerciements

Avant tout développement sur cette expérience professionnelle, il apparaît opportun de commencer ce rapport de stage par des remerciements, à ceux qui m'ont beaucoup appris au cours de ce stage, et même à ceux qui ont eu la gentillesse de faire de ce stage un moment très profitable.

Tout d'abord, je voudrais adresser mes remerciements à Madame Nadia Lynda MOKDAD qui m'a formé et accompagné tout au long de cette expérience professionnelle avec beaucoup de patience et de pédagogie.

Deuxièmement, je tiens à remercier vivement Monsieur Jean-Michel FOURNEAU, pour son accueil, ses conseils ainsi pour le matériel de travail.

Enfin, je voudrais remercier également toutes les personnes du laboratoire David. Grâce à eux, j'ai pu m'accomplir totalement dans mes missions et mes connaissances.

Table des matières

Lexique

SF : Spreading Factor (facteur d'étalement de spectre)

BD : Base de données

MAJ : Mise à jour

IA : Intelligence Artificielle

Introduction

L'Internet des objets, désigne l'ensemble des infrastructures et technologies mises en place pour faire fonctionner des objets divers par le biais d'une connexion Internet et une des technologies qui permet cela, LoRa (LOng RAnge) qui est une technologie de modulation de fréquence par étalement de spectre permettant aux objets une communication sans fil longue distance et le laboratoire David s'intéresse beaucoup à cette technologie et tout ce qui concerne les recherches centrées sur les nouveaux défis du numérique dans la ville du futur ("Smart City").

Au cours de mes études à l'UPEC et de la réalisation de projets personnels, mes compétences personnelles sont bien cohérentes avec l'orientation du développement IA et logiciel. Le programme de formation, les bases techniques et l'ambiance d'études à l'UPEC me permettent d'avoir une base solide dans le développement logiciel et même d'apprendre de nouveaux langages et outils durant le déroulement du stage. Le stage obligatoire avant l'obtention du diplôme est une bonne occasion pour moi d'acquérir des expériences dans l'environnement de travail et de définir clairement mon orientation dans l'avenir.

Dans ce rapport, je vous présenterai le processus de mon stage chez le laboratoire David de l'université de Versailles Saint-Quentin-en-Yvelines.

Présentation générale du laboratoire

1. Information sur le laboratoire

- **Description :** Le laboratoire DAVID, créé en 2015, est un laboratoire de l'université de Versailles Saint-Quentin-en-Yvelines qui propose un projet scientifique centré sur les nouveaux défis du numérique dans la ville du futur ("Smart City").
- **Adresse :** Bâtiment Descartes — Campus des Sciences
  Université de Versailles Saint-Quentin-en-Yvelines
  45 avenue des États-Unis
  78035 Versailles
- **Logo du laboratoire :**

2. Les axes de recherche du laboratoire

2.1 Mobilité et ville numérique

Cet axe se consacre à l'élaboration de solutions numériques pour les mobilités urbaines et périurbaines, en matière de transport et d'organisation des territoires et de leurs activités.

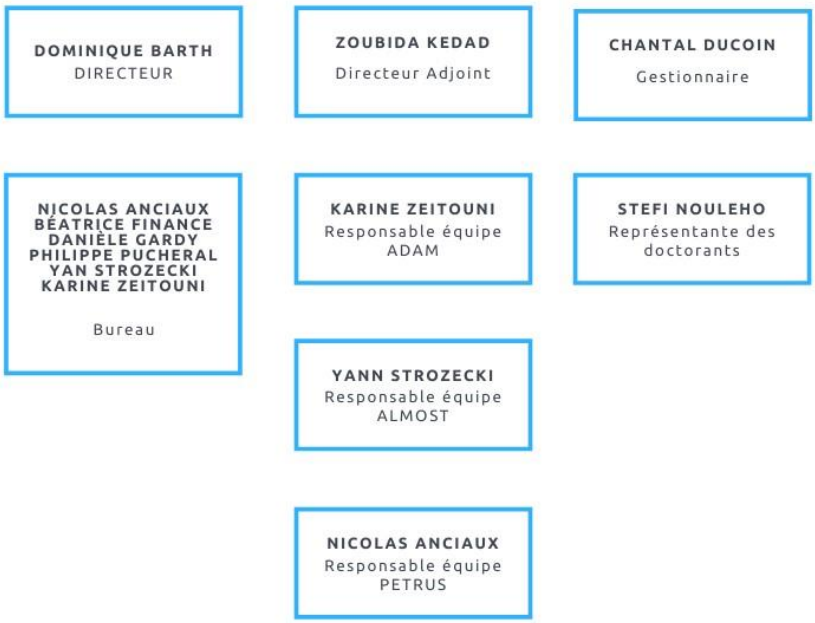
2.2 Donnés, justice et société

Cet axe, impliquant informaticiens, juristes et sociologues, se consacre à l'analyse et la prédiction de données de jurisprudence, au statut juridique des données et à l'usage des données sociologiques dans le respect de la vie privée.

2.3 Energie et ville durable

Cet axe se consacre aux outils d'intelligence artificielle pour la gestion intelligente de l'énergie en milieu urbain et à l'analyse et la restitution de données d'exposition à la population urbaine.

3. Organisation du laboratoire



3.1 Les équipes du laboratoire

3.1.1 Equipe ADAM

L'équipe ADAM (Ambient Data Access and Mining), dont l'expertise concerne la modélisation des données hétérogènes, l'intégration et la fusion sémantique de ces données, ainsi que l'extraction de connaissances à l'aide de requêtes complexes et de techniques de fouille.

3.1.2 Equipe ALMOST

L'équipe ALMOST (Algorithms and Stochastic Models), dont l'expertise concerne la modélisation, la résolution algorithmique et l'évaluation de performances, dans divers domaines d'application (smartcities, télécommunications, analyse moléculaire).

3.1.3 Equipe PETRUS

L'équipe PETRUS (Personnal and Trusted Cloud), dont l'expertise concerne les structures et algorithmes de gestion de données personnelles et leur sécurisation, la gestion de données à base de matériel sécurisé, les modèles de partage, et l'intégration de l'ensemble de ces techniques dans des architectures Privacy-byDesign permettant à l'individu de gérer son patrimoine numérique sous contrôle.

Information sur le stage

1. Information sur le poste de travail

- Position : stagiaire Informatique
- Equipe : ALMOST
- Durée : du 12/04/2021 à 31/07/2021
- Maître de stage : Nadia Lynda MOKDAD et Jean-Michel FOURNEAU
- Fonction : Développeur Python
- Technologies utilisées : Python / Qt Designer / PyQt5 - Missions :
    - Participer à la création d'algorithmes de machine Learning pour diminuer les collisions au sein des réseaux LoRaWAN
    - Création de l'outil « Mulane » qui représente les différentes méthodes d'amélioration de performances des réseaux LoRaWAN

2. Environnement de travail

Au laboratoire je fais partie de l'équipe ALMOST, nous travaillons du lundi au vendredi toutes les semaines, le lundi et jeudi en présentiel et les autres jours en télétravail. Une journée de travail commence à 9h et se termine à 17h. le laboratoire nous permet d'avoir des horaires flexibles. Je peux commencer même à 10h et terminer jusqu'à 18h30 ce qui nous permet de travailler plus confortablement.

Les développeurs et mon collègue de stage m'aidaient toujours en plus d'être gentille ce qui m'a permis de m'intégrer rapidement au sein de l'équipe du laboratoire.

3. Outils utilisés

Au cours de ce stage, le principal outil qu'utilise le laboratoire c'est le langage Python pour tous ce qui est du machine Learning pour l'implémentation des algorithmes donc automatiquement j'étais amené à utiliser ce langage que je connaissais déjà.

Pour l'interface graphique j'ai utilisé Adobe Illustrator pour la création des maquettes de l'outil pour ensuite passer du temps à apprendre des nouveaux outils pour moi qui sont Qt- Designer pour la création des composants et de l'outil PyQt5 pour l'implémentation des méthodes et en mesure que j'apprenais, mes recherches se sont approfondies ce qui m'a permis de pouvoir commencer l'implémentation de cette interface.

Au niveau du logiciel j'avais le choix d'utiliser l'outil de texte que je voulais et moi j'ai toujours eu l'habitude d'utiliser Sublime text qui me permet de travailler facilement avec python avec ces extensions et les couleurs du texte par rapport au code.

Concernant la base de données j'ai utilisé Sqlite3 qui permet de gérer des base de données assez grande et suffisante pour le projet.

4. Programme du stage

Avant de commencer, mon maitre de stage m'avait proposé une liste de projets et de taches que je devrais faire durant les semaines de mon stage :

- Test des performances des algorithmes existant pour LoRaWAN (LoRaSIM et LoRaFREE, LoRaMAB)
- Proposition d'idées de méthodes pour améliorer les performances en diminuant les collisions dans le réseau
- Implémentation des algorithmes
- Outil qui regroupe toutes les méthodes (Mulane)
- Création de l'interface graphique
- Outil de comparaison des méthodes (Mulane & Comparaison)
- Réalisation d'une maquette pour l'analyse des résultats

Travaux réalisés

1. Tests des performances des simulateurs de base

Au début du stage j'ai commencé tout d'abord par le test des simulateurs qui existe déjà pour les réseaux LoRaWAN afin d'avoir une idée de ce qu'on attendra des méthodes qu'on va créer.

Après avoir testé les performances des simulateurs il a été temps de commencer à proposer des idées d'améliorations afin d'implémenter les algorithmes.

Avant de parler des méthodes élaborer, comme le but de ces méthodes est de diminuer les collisions dans un réseau LoRaWAN il est important d'expliquer les différents types de collisions dans ces réseaux.

Les collisions

Un réseau LoRaWAN peut rencontrer quatre types de collisions :

Frequency collision :

Une collision de fréquences arrive quand deux paquets ont une fréquence trop proche, on peut résumer cela en trois cas :

- Si la différence de fréquences des deux paquets est inférieure ou égale à 120KHz et la bande passante des deux paquets est égale à 500.
- Si la différence de fréquences des deux paquets est inférieure ou égale à 60KHz et la bande passante des deux paquets est égale à 250
- Si la différence de fréquences des deux paquets est inférieure ou égale à 30KHz.

SF collision

Si les valeurs des SFs des deux paquets sont égaux alors une collision de type SF collision est détectée.

Energy collision

Une collision de tel type détectée, si l'un des cas suivants est réalisé :

- La valeur absolue de la différence entre le rssi (Received Signal Strength Indication) qui est une mesure du niveau de puissance en réception d'un signal issu d'une antenne des deux paquets est inférieure à un seuil de 6 dB (La puissance seuil fonctionnelle), dans ce cas les deux paquets sont trop proches l'un de l'autre, les deux entrent en collision, renvoyer les deux paquets comme victimes, Sinon ;
- Si, la différence entre le rssi des deux paquets est strictement inférieure à un seuil de 6 dB (p1.rssi – p2.rssi < (La puissance seuil fonctionnelle)), dans ce cas p2 surpuissant p1, renvoyer p1 comme victime.
- Sinon ; p2 était le paquet le plus faible, renvoyer p2 comme victime.

Timing collision

En supposant que p1 est le paquet fraîchement arrivé et que c'est la dernière vérification, nous avons déjà déterminé que p1 est un paquet faible, donc la seule façon de gagner est d'être suffisamment en retard (seuls les n-5 premiers symboles de préambule se chevauchent).

En supposant que 8 symboles de préambule le rendent comme une victime.

2. Méthodes d'amélioration de performances

Dans mon stage on s'est intéressée beaucoup plus sur les collisions de SF et l'élaboration des méthodes s'est faite durant l'avancement, après chaque méthode élaborer et implémenter, des tests sont effectués pour comparer les résultats avec les résultats des simulateurs de base.

Les méthodes implémentées sont :

Static Random

Cette méthode se rapproche beaucoup du simulateur de base LoRaSIM avec une seule modification c'est l'initialisation du SF qui se fait d'une manière aléatoire.

Dynamic Random

La méthode fonctionne avec une contrainte sur le taux de collisions, son fonctionnement est simple, chaque nœud de la station de base transmet des paquets, si un paquet tombe en collision on envoie le paquet en utilisant une autre SF choisie au hasard parmi les SF de la station.

Dynamic P-Random

Son fonctionnement ressemble à celui de Dynamic Random mais en utilisant une valeur supplémentaire qui est le P dont la valeur est choisie par l'utilisateur, la valeur qu'on a choisie est 0.4.

Comme d'habitude, les nœuds de la station de base transmettent des paquets, si un de nos paquets rencontre une collision, alors on tire une valeur au hasard qui varie entre 0 et 1, si le nombre tiré est supérieur ou égal à P, on change de SF sinon on ne fait rien et on force la transmission de notre paquet avec le SF choisi initialement.

STEPS Full et STEPS Individual

Dans cette méthode on a introduit une nouvelle idée qui est le tableau des scores, pour faire simple, on a un tableau qui représente les probabilités de choisir un SF et la somme des valeurs du tableau sera 1 et lors d'une collision le choix du SF se fera donc de manière aléatoire selon les probabilités du tableau.

Chaque collision rencontrée en utilisant un SF, la probabilité de ce dernier sera diminuée dans notre tableau, et dans le cas contraire elle sera augmentée.

La somme du tableau sera toujours égale à 1 car on effectue une normalisation après chaque modification des valeurs.

Lors d'une collision le SF sera donc modifié pour récupérer le meilleur SF, et s'il y a plusieurs collisions d'affilée sur le même SF, réduire encore plus son poids pour éviter de continuer de l'utiliser.

Notre tableau de poids permet de choisir le meilleur paramètre de SF lors de la transmission, il est donc envisageable de récupérer ces données avec pour objectif de les transmettre aux nouveaux nœuds lors de leur création dans les futures simulations.

L'objectif est d'utiliser ces données lors de la mise en place d'une nouvelle simulation c'est pourquoi lors de la création de nœuds nous récupérerons les données du nœud le plus proche du nouveau nœud qui sont stockées dans une base de données.

STEPS Full et STEPS Individual sont les mêmes simulateurs dans leurs fonctionnements, la différence est que STEPS Individual est un simulateur dans des conditions plus réel que STEPS Full car il a moins d'informations sur les paquets qu'il a envoyé (il ne sait pas si un paquet a été reçu que s'il y a un ack).

STEPS E-Greedy et Boltzmann

Ces méthodes ont été créées par mon collègue de stage, les méthodes E-Greedy et Boltzmann effectuent des changements de SF dans le cas de collisions en utilisant l'approche STEPS avec une stratégie d'exploitation E-Greedy et Bolzmann respectivement.

3. Création de l'outil MULANE

MULANE qui veut dire Multi-methods based spreading factor for LorA Networks c'est l'outil qui va regrouper toutes les méthodes d'amélioration de performance des réseaux LoRaWAN.

Les étapes de création de l'outil sont :

- Dessin du design de l'outil
- Création des composants sur Qt Designer
- Implémentation de l'outil avec PyQt5 et intégration des méthodes
- Création d'un Logo pour l'outil

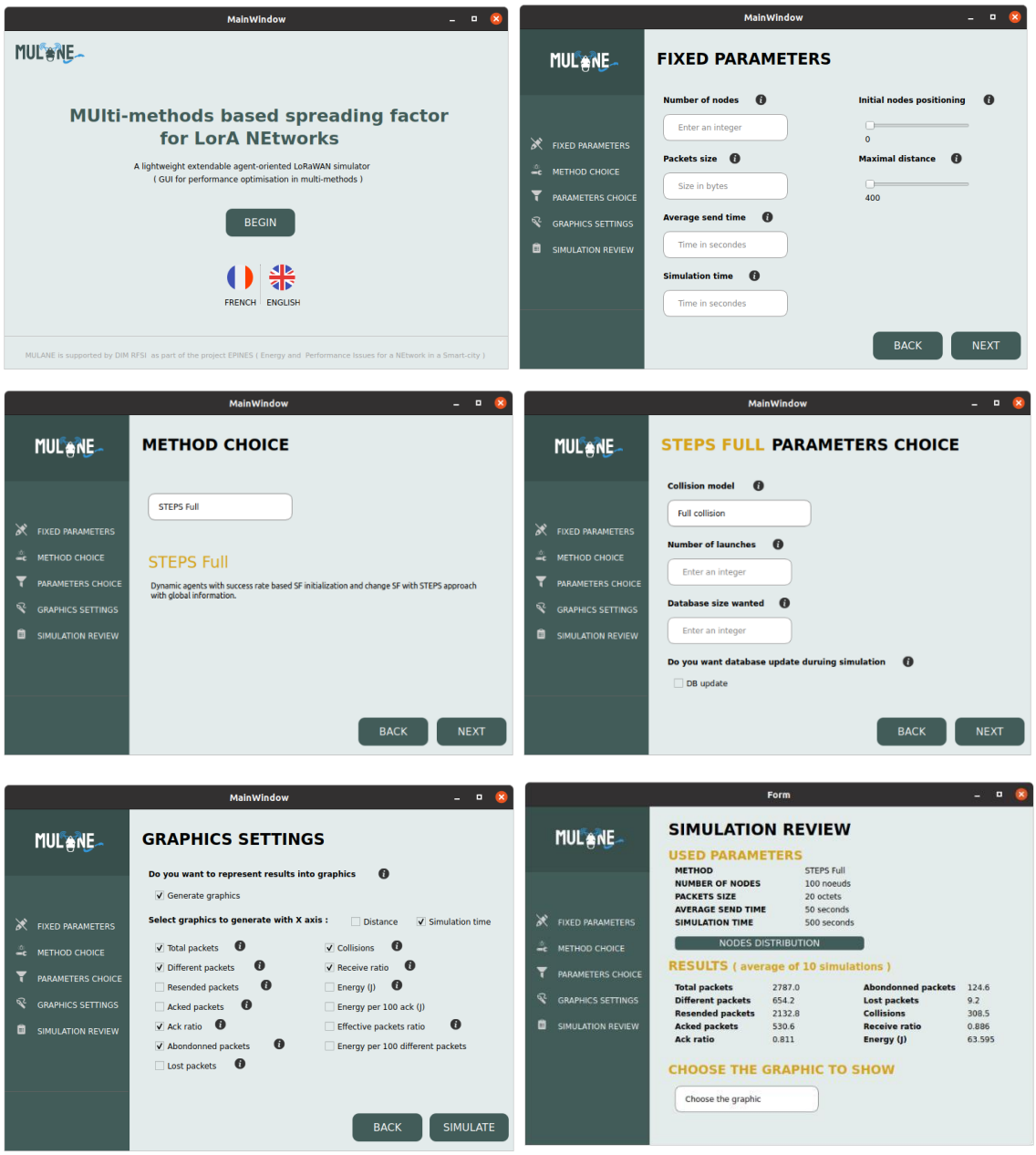
3.1 Dessin du design et création des composants

J’ai commencé par la création du design de l’outil en utilisant Adobe Illustrator afin de pouvoir visualiser et avoir une idée de comment l’outil se présentera mais aussi pour le présenter à mon maître de stage et voir son avis et de modifier facilement ce qu’il faut changer.

Après validation du design j’ai commencé à apprendre l’utilisation de Qt Designer qui est un outil Qt pour la conception et la construction d'interfaces utilisateur graphiques (GUI) avec Qt Widgets.

On peut composer et personnaliser des fenêtres ou boîtes de dialogue de manière WYSIWYG (ce que vous voyez est ce que vous obtenez) et les tester à l'aide de différents styles et résolutions.

L’outil a été simple d’utilisation alors il m’aurait fallu que quelques jours pour le prendre en main et j’ai commencé à créer tous les composants de mon design dans Qt Designer.



Composants de l'outil Mulane crée avec Qt Designer

3.2 Implémentation de l'outils et intégration des méthodes

Avant de commencer fallait que j'apprends à utiliser PyQT5, alors j'ai passé deux semaines à comprendre comment utiliser le langage et de le maitriser dans la mesure de pouvoir créé et rendre l'interface fonctionnelle.

L'outil Qt Designer permet de générer le code PyQt5 de tous les composants alors il fallait que je code les fonctions de chaque bouton, champs de texte et label et introduire les différentes méthodes créer.

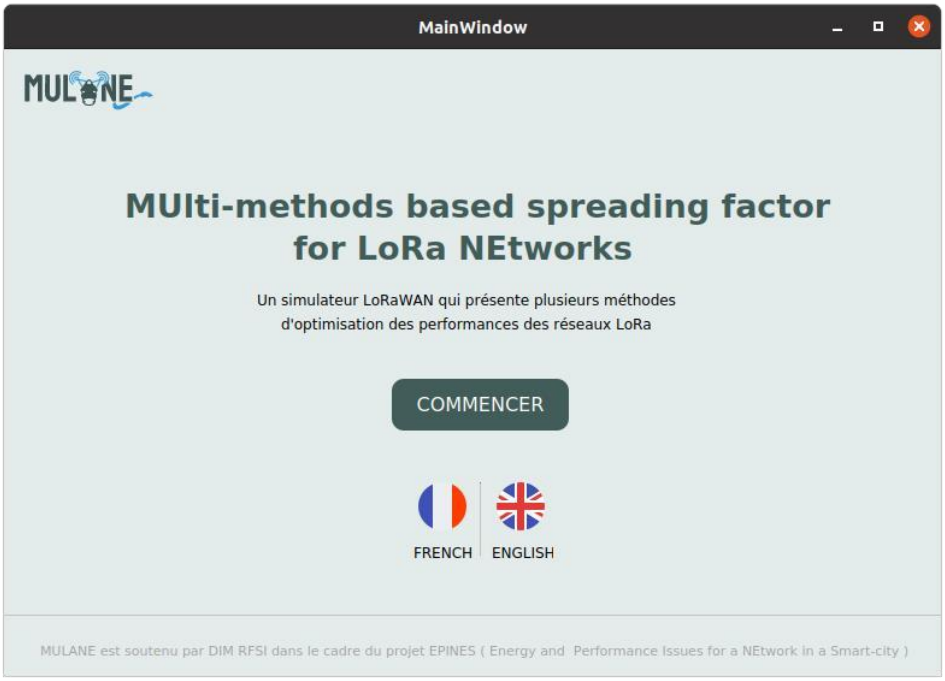
Cette étape a été longue mais plaisante de voir qu'on commence à pouvoir interagir avec une interface qui au début était juste un dessin sur Adobe Illustrator.

3.2.1 Description de l'outil

Dans cette section je vais expliquer comment l'interface fonctionne, le concept est simple et se fait en 4 étapes :

- **Choix des paramètres fixes** L'utilisateur choisis les paramètres fixes qui ne change pas pour toutes les méthodes
- **Choix de la méthode** L'utilisateur choisit la méthode avec laquelle il veut effectuer des simulations
- **Choix des paramètres** Chaque méthode à des paramètres spécifique que l'utilisateur peut définir
- **Paramètres graphiques** Avant de lancer la simulation, on peut choisir si on veut représenter les résultats sous forme de graphiques ou texte uniquement.

Fenêtre d'accueil

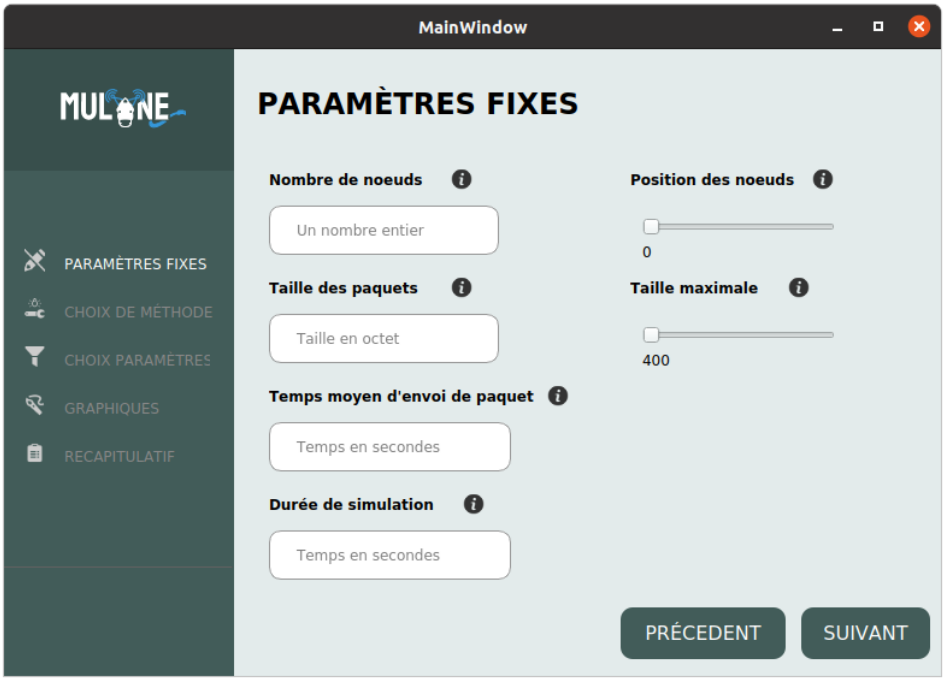


Fenêtre d'accueil

- Elle est simple, apporte la définition du nom Mulane et ce que fait l'outil
- Deux boutons pour choisir la langue préférer par l'utilisateur à savoir soit Français ou Anglais

- Le bouton “Commencer” permet de commencer une nouvelle simulation en passant à la fenêtre suivante.

Fenêtre de choix de paramètres fixes



Fenêtre de choix des paramètres fixes

Les paramètres fixes sont : **Nombre**

de nœuds

désigne le nombre de nœuds dans le réseau, chaque nœud sera placé dans le réseau en fonction des paramètres **Position des nœuds** et **Taille maximale.**

Position des nœuds

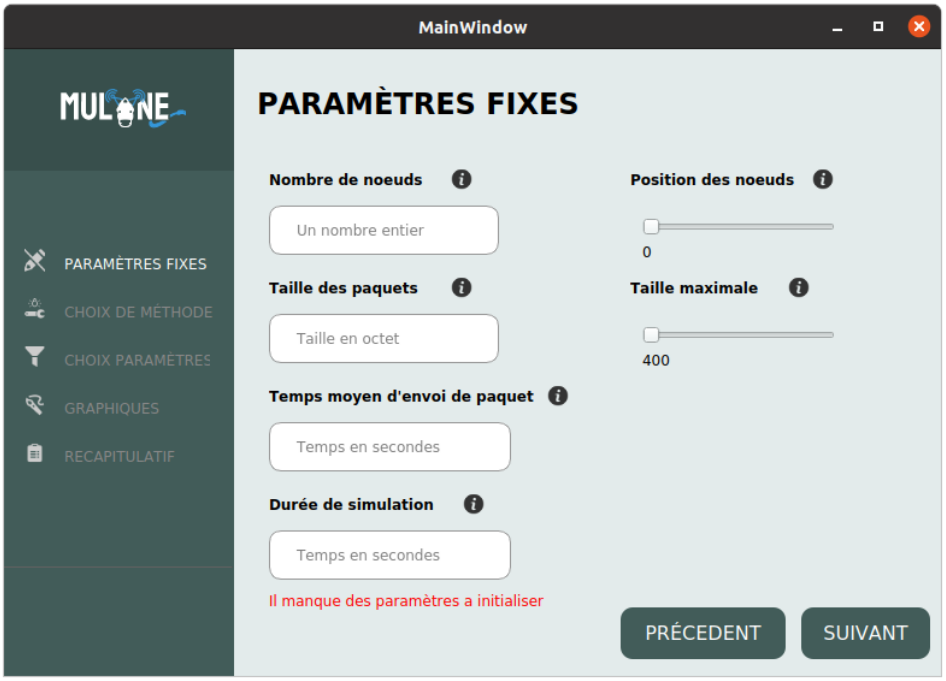
Sa valeur peut aller entre 0 et 50, ce nombre représente le nombre de cercle autour de la station de base où les nœuds seront placer donc à 0 le placement des nœuds est totalement aléatoire et plus le nombre est plus grand le placement est de moins en moins aléatoire.

Taille maximale

La valeur de ce paramètre peut être entre 400 (m) jusqu’à la distance maximale qui est de 8921.359 (m) et fixe tout simplement la distance max ou les nœuds pourront être placés.

En plus des autres paramètres qui sont Taille des paquets, Temps moyens d'envoi de paquet et durée de simulation.

- Dans cette fenêtre l'utilisateur peut entrer les paramètres qui ne change pas pour toutes les méthodes qui existe dans l'outil
- Après avoir rentré tous les champs, le bouton "Suivant" emmène à la prochaine étape qui est le choix de la méthode
- Les "i" placer à coter des labels sont là pour apporter les informations nécessaires pour comprendre l'utilité du champ.



Fenêtre blocage de passage à la prochaine étape

- Si l'utilisateur ne rentre pas tous les paramètres il ne pourra pas passer à la prochaine étape.

Fenêtre de choix de méthode

Fenêtre de choix de méthode

- La fenêtre permet à l'utilisateur de choisir la méthode grâce à un menu déroulant (le menu déroulant a été choisi pour la représentation des différentes méthodes afin de permettre de rajouter de nouvelles méthodes dans le futur)
- Lorsqu'une méthode est sélectionnée, une description de la méthode apparaît juste en dessous pour expliquer à l'utilisateur les spécificités de la méthode
- Le bouton Suivant est important là car pour chaque méthode choisie, une fenêtre dédiée à cette méthode.

Fenêtre choix de paramètres d'une méthode

Fenêtre de choix des paramètres de méthode

On prend comme exemple la fenêtre de la méthode STEPS Full car elle comporte la plupart des paramètres saisissables qui sont :

Modèle de collision

Ou mode de détection des collisions, on a deux modes de détection, collision complète et collision simple.

On sait qu'il existe quatre types de collisions (Frequency collision, SF collision, timing collision et Energy collision).

Le mode détection complète détectera tous ces types de collision alors que le mode collision simple ne détectera que les collision de type (Frequency collision et SF collision).

Nombre de lancements

Le nombre de fois que le simulateur sera lancé et les résultats seront la moyenne par rapport à la valeur de ce paramètre.

Taille de la base de données (BD) souhaitée

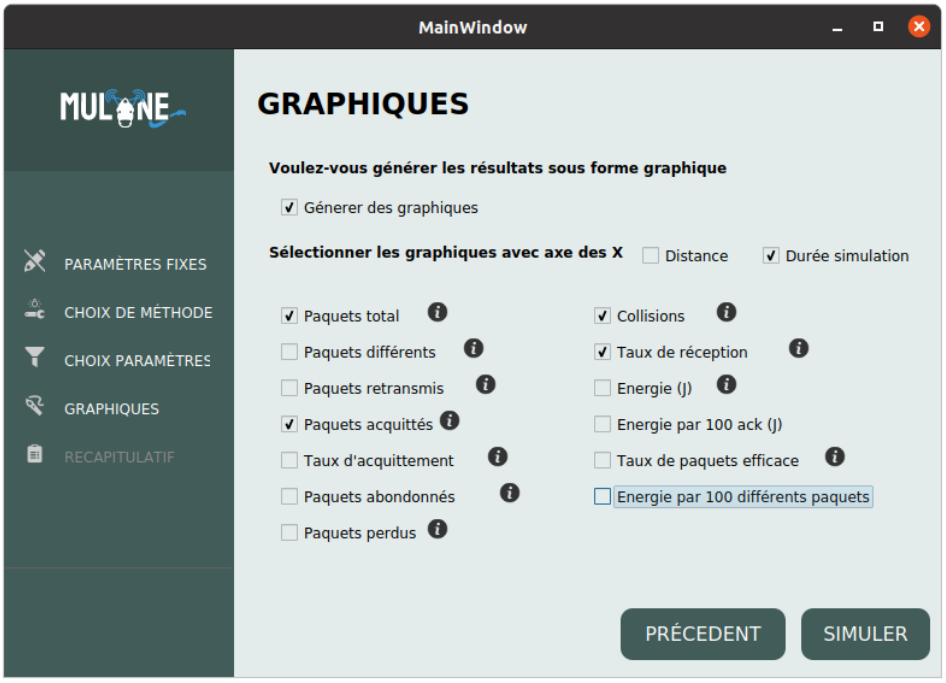
Représente la taille de BD que le simulateur démarre avec, si par exemple l'utilisateur choisi une valeur de 100, le simulateur va préparer une BD de 100 entrées et puis commencer la simulation.

Mise à jour de la BD

Durant la simulation l'utilisateur peut choisir de permettre la maj de la BD, c'est-à-dire permettre le partage d'expérience entre les nœuds ou bien de ne pas permettre la maj de BD afin de simuler sur un nombre d'entrées de BD fixe.

- L'utilisateur remplie les champs
- Là aussi si tous les champs ne sont pas remplis, l'utilisateur n'a pas le droit de passer à l'étape suivante.

<u>Fenêtre des paramètres graphique</u>

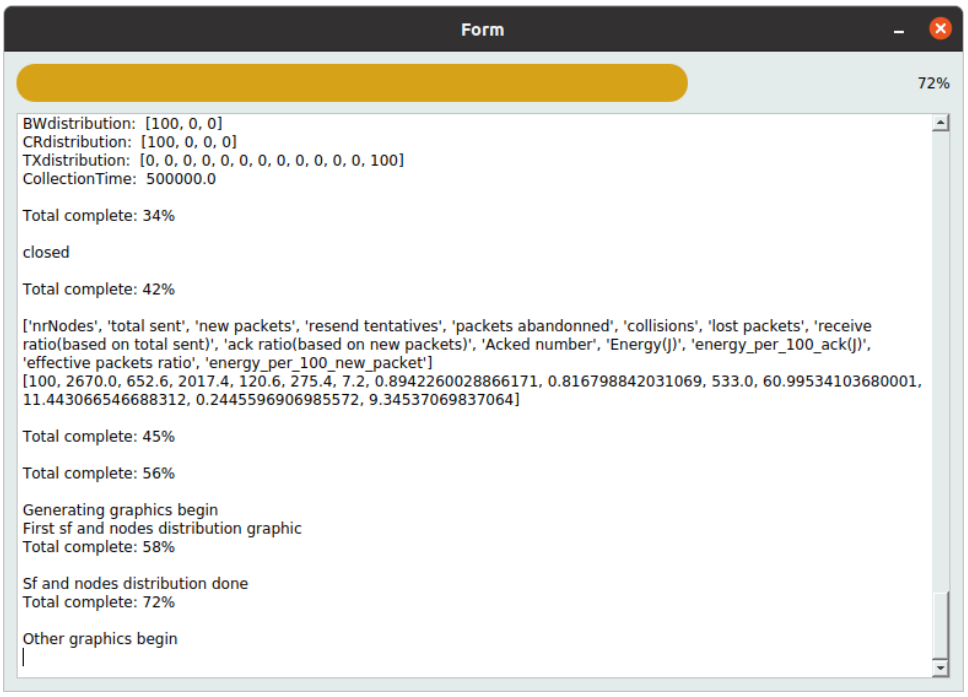


Fenêtre de choix des paramètres graphiques

- L'utilisateur a le choix de soit générer des graphiques avec les résultats ou non
- S'il choisit d'en générer il pourra choisir lesquelles générer
- Il peut aussi choisir de générer les graphiques soit en fonction de la distance ou bien de la durée de simulation et même les deux au même temps

- Si l’axis des X n’est pas choisis, les deux seront générer

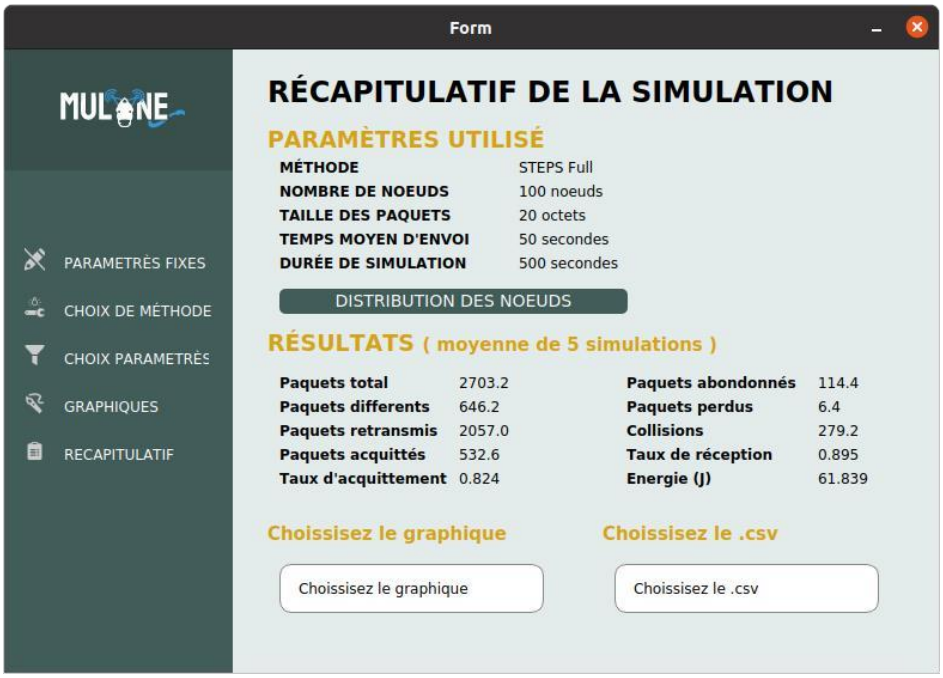
Quand tous les champs sont remplis l’utilisateur pourra appuyer sur le bouton “SIMULER” pour lancer la simulation



Fenêtre de progression de la simulation

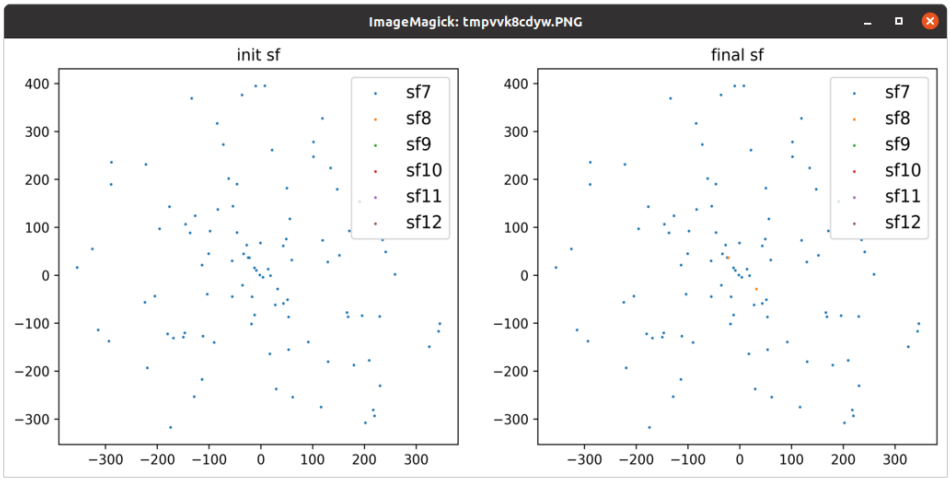
Pendant la simulation une fenêtre qui indique l’état d’avancement de la simulation, le pourcentage du progrès ainsi que toutes les sorties que génère le simulateur.

Fenêtre des résultats



Fenêtre récapitulatif de la simulation

- La fenêtre affiche dans une première partie un rappel des paramètres utilisés ainsi que la distribution des nœuds par rapport à la station de base et le SF utilisé par chaque nœud



Représentation de la répartition des nœuds et SF utilisé

- Dans la deuxième partie les résultats de la simulation à savoir :

Paquets total : Le nombre de transmissions total effectuée par les nœuds du réseau on peut dire que (paquets total = paquets différents + paquets retransmis).

Paquets différents : Le nombre de paquets unique envoyé par les nœuds du réseau.

Paquets retransmis : le nombre de retransmission de paquets effectuée par les nœuds du réseau, un paquet est retransmis si sa première transmission n'était pas concluante.

Paquets acquittés : Le nombre de paquets acquittés par la station de base.

Taux d'acquittement : Représente le nombre de paquets acquittées / nombre de paquets différents

Paquets abandonnés : nombre de paquets abandonnés par les nœuds, un paquet est abandonné si il a été retransmis plus de 8 fois.

Paquets perdus : Nombre de paquets perdus dans le réseau pendant la transmission.

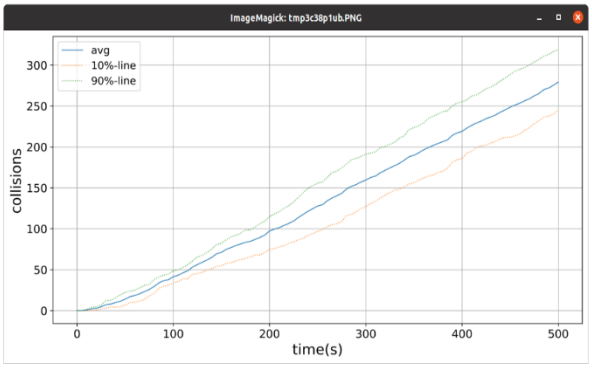
Collisions : Nombre de collisions détecté dans le réseau durant la transmission

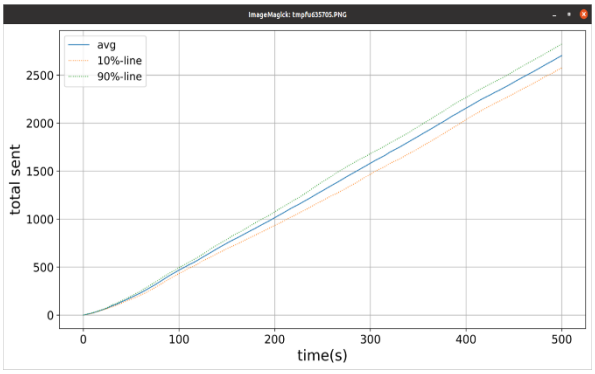
Taux de réception : Représente le nombre de paquets reçus / nombre de paquets total

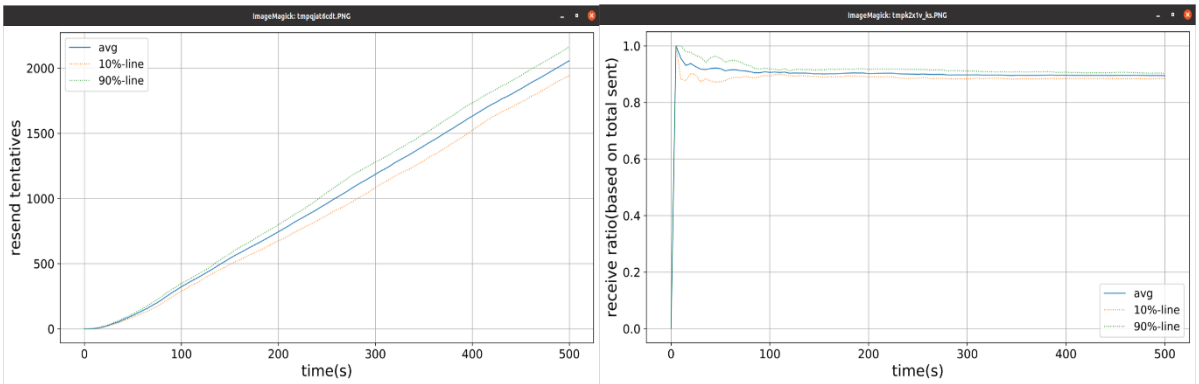
Energie : L'énergie consommé par les nœud pour effectuer les transmission nécessaire

- Et dans la troisième partie l'utilisateur peut choisir le graphique qu'il veut afficher en plus des fichiers .CSV afin de lui permettre de vérifier les valeurs

L'utilisateur pourra donc visualiser les différents graphiques qu'il a cocher depuis le menu déroulant, par exemple dans notre simulation on a choisi de générer les graphiques de : paquets total, paquets acquittés, collisions et taux de réception







Résultats des simulations sous forme de graphiques

Et de même pour les fichiers .CSV, il faut savoir que pour chaque graphique générer, son fichier .CSV aussi il est généré et l'utilisateur pourra visualiser les données avec lesquelles les graphiques ont été dessiné, par la suite il pourra soit vérifier les données, dessiner un autre type de graphique à l'aide de ses données.

	A	B
1	sim time	collisions
2	0	0
3	5000	2
4	10000	2
5	15000	2
6	20000	2
7	25000	4
8	30000	4
9	35000	4
10	40000	4
11	45000	4
12	50000	5
13	55000	5
14	60000	5
15	65000	6
16	70000	6
17	75000	8
18	80000	9
19	85000	13
20	90000	13
21	95000	13
22	100000	13
23	105000	13

Collisions en fonction de temps

	A	B	C	D
1	x	y	distance	collisions
2	-19.54017628	-23.45116542	30.52500039	1
3	214.4471867	-14.00328005	214.9039034	0
4	12.78123521	155.0745938	155.6004165	0
5	81.95404787	332.1726193	342.1331831	0
6	9.979289283	8.880503129	13.35850105	0
7	-32.76900868	-169.1656209	172.3102296	0
8	-222.9247223	244.945408	331.2003694	4
9	110.1513331	-291.5468964	311.6615295	2
10	136.6529455	-114.778751	178.4606096	2
11	-2.948895646	133.7589595	133.7914617	1
12	339.1876743	-51.87423013	343.1314823	0
13	-121.2951936	-25.02496377	123.8497993	3
14	116.7039284	-144.7745413	185.9555719	0
15	25.53573676	8.565486931	26.93401972	0
16	-50.14135651	41.87195857	65.3254663	2
17	257.3976922	194.6862998	322.7325941	0
18	62.95818954	-299.1027204	305.6569498	1
19	-52.42164265	-166.7133331	174.7608768	0
20	111.9083397	342.2204934	360.0532497	1
21	332.3745308	-7.572232048	332.4607758	3
22	2.030091914	-264.2418852	264.2496834	1
23	303.9815645	-172.5273252	349.5289251	2

Collisions en fonction de la distance

3.3 Création du logo pour l'outil

L'idée du logo a été donné par mon maitre de stage et elle vient du nom de l'outil "Mulane" Mul-Ane. . . Âne l'animal, alors il fallait absolument que le logo comporte un âne.

- Le design du logo a été aussi fait en utilisant Adobe Illustrator
- La couleur verte bleuté de la police est la même utilisé dans le design des fenêtres
- Le A de mulAne est remplacé par un visage d'un âne
- Les oreilles de l'âne sont remplacées par des antennes pour amener l'aspect réseau
- Et bien sûr la petite queue de l'âne à côté du mulaNE pour appuyer sur le fait qu'il y'a un âne dans le logo.

4. Mulane & Comparaison

Une autre version de l'outil Mulane c'est Mulane & Comparaison, c'est le même outil mais en plus il contient tous les simulateur de base + crée afin de permettre aux développeur de comparer les nouvelles méthodes avec les méthode de base:

LoRaSIM

Le simulateur LoRaSim est le simulateur de loRaWAN le plus populaire et est basé sur un simulateur d'événements discrets de Python de la bibliothèque de simulation SimPy.

Le simulateur LoRaSim prends comme paramètres un nombre de nœuds, un temps moyen de diffusion, ainsi que le temps de simulation. Il existe un paramètre experiment qui est un entier permettant de changer les paramètres radios de la simulation.

Tous les autres simulateurs pour LoRaWAN sont basés sur LoRaSim, ces derniers étendent LoRaSim pour plusieurs applications IoT.

Mais le simulateur LoRaSim bien que le plus populaire ait quelques inconvénients, il ne prend pas en compte le système d'acquittements, ni la taille des paquets.

LoRaFREE

Le simulateur LoRaFREE est une extension du simulateur LoRaSim qui a pour objectif de synchroniser les transmissions pour obtenir une meilleure collecte de données fiable, pour cela la collecte de données au lieu de des intervalles périodiques ou apériodiques connus par la passerelle et les terminaux. Le planning est ensuite envoyé à l'ensemble des terminaux pour éviter les collisions. Les transmissions sur un même SF sont donc effectuées de manière séquentielle et les transmissions simultanées s'effectuent sur des SFs différents.

De plus contrairement à LoRaSim le simulateur LoRaFREE prends lui en compte les acquittements ainsi que la taille des paquets.

LoRaMAB

Ce simulateur au lieu d'utiliser le système de tableau de STEPS, il utilise l'algorithme EXP3 (Exponential Weights for Exploration and Exploitation) qui permet d'orienter de manière autonome la décision des dispositifs LoRa vers les facteurs d'étalement les moins sollicités sans système de pénalité pour les échecs.

En plus de tous les autres méthode cité dans Mulane, et grâce au menu déroulant de choix des méthode il été facile pour moi de rajouter tous ces méthodes.

Synthèse des travaux restant

Après une réunion avec mon maître de stage pour définir les taches qui me reste à faire et donc pour le moments les tâches restante du stage sont :

- Réalisation d'une maquette pour l'analyse des résultats en utilisant de vrai équipements dont une antenne LORIX fournie par Monsieur Jean-Michel FOURNEAU ainsi que des cartes réseaux.
- Potentiellement la création d'un site web vitrine pour l'outil Mulane qui présentera l'outil et permettra de le télécharger directement.

Les apports du stage

Au cours de ce stage, j'ai beaucoup appris. Les apports que j'ai tiré de cette expérience professionnelle peuvent être regroupés autour de deux idées principales : les compétences acquises, les difficultés rencontrées et solutions apportées.

1. Les compétences acquises

Ce stage m'a fait découvrir le monde du développement et a approfondi mes connaissances et à améliorer mes compétences de programmation.

De cette expérience j'ai retenu que travailler en autonomie ne signifie pas seulement travailler en toute liberté, cela implique de devoir prendre des initiatives pour planifier et exécuter des tâches, de faire des bilans, d'évaluer son efficacité, de réfléchir aux procédures utilisées. La réflexion est essentielle dans l'acquisition de l'autonomie

2. Les difficultés rencontrées

À cause de la pandémie je n'étais pas en contact permanent avec mon maitre de stage et le développeur et nos rencontres s'effectuaient par des appels et des sessions de Teams et Zoom afin de pouvoir leur présenter l'avancer de mon travail et discuté des prochaines étapes.
Aussi l'utilisation de nouveaux outils a fait que j'ai fait quelques erreurs au début mais cela m'a permis d'un autre coté d'apprendre de ses erreurs et avancé.

Conclusion

Ce stage a été très enrichissant pour moi, car il m'a permis de découvrir le domaine du développement logiciel et de l'IA.

Il m'a permis de participer concrètement au développement et de la maintenance des applications, j'ai pu approfondir mes connaissances en informatique.

Ce stage m'a aussi permis de comprendre que j'avais les bases nécessaires pour m'adapter facilement dans les différents domaines du développement ce qui m'a poussé à m'orienter beaucoup plus vers l'intelligence artificielle ce qui m'a permis de décrocher des entretiens pour ma recherche d'Alternance que j'espère seront concluant.

Annexes

- Code de la méthode Dynamic random pour le changement de SF aléatoirement

```
if (node.packet.collided == 1):
    # OneBS random
    if (versionChoice == 1):
        #print "J'ai changé de SF : node", node.nodeid
        node.packet.sf = random.randint(7,12)
```

- Code de la méthode Dynamic P-Random pour le changement de SF

```
if (node.packet.collided == 1 or node.packet.lost):
    p = random.random()
    if p >= 0.40:
        node.packet.sf = random.randint(7,12)
    else:
        pass
```

- Code de la méthode STEPS Full pour le système de punition

```
if node.packet.lost:
    node.buffer += PcktLength_SF[node.packet.sf-7]
    node.lost = node.lost + 1
    node.lstretans += 1
    global nrLost
    nrLost += 1
    node.SFs[node.packet.sf-7] = node.SFs[node.packet.sf-7]*0.8
    node.SFs = simple_norm(node.SFs)
elif node.packet.perror:
    node.losterror = node.losterror + 1
    global nrLostError
    nrLostError += 1
elif node.packet.collided == 1:
    node.buffer += PcktLength_SF[node.packet.sf-7]
    node.coll = node.coll + 1
    node.lstretans += 1
    global nrCollisions
    nrCollisions = nrCollisions +1
    node.SFs[node.packet.sf-7] = node.SFs[node.packet.sf-7]*0.8
    node.SFs = simple_norm(node.SFs)
elif node.packet.acked == 0:
    node.buffer += PcktLength_SF[node.packet.sf-7]
    node.noack = node.noack + 1
    node.lstretans += 1
    global nrNoACK
    nrNoACK += 1
elif node.packet.acklost == 1:
    node.buffer += PcktLength_SF[node.packet.sf-7]
    node.acklost = node.acklost + 1
    node.lstretans += 1
    global nrACKLost
    nrACKLost += 1
else:
    node.lstretans = 0
    global nrACKed
    nrACKed = nrACKed + 1
    node.SFs[node.packet.sf-7] = node.SFs[node.packet.sf-7]*3
    node.SFs = simple_norm(node.SFs)
if node.packet.lost == False and node.packet.collided == 0:
    global nrReceived
    nrReceived = nrReceived + 1
    node.SFs[node.packet.sf-7] = node.SFs[node.packet.sf-7]*0.9
    node.SFs = simple_norm(node.SFs)
```

- Code de la normalisation de la somme des probas

```python
def simple_norm(list_sfs):
    sum_exp = sum(list_sfs)
    result = [i / sum_exp for i in list_sfs]
    return result
```

- Code qui permet de lancer les simulateur avec l'outil MULANE

```python
for i in range(sim_times):
    if(simulator_name=="STEPS_full"):
        print("STEPS full")
        sf_file,res_file,res_file_on_time,result = STEPS_full.run(nrNodes,avgSendTime,simtime,plsize,db_size,is_update_db = is_update_db,fullcollision = fullcollision,max_dist=maxDist
        sys.stderr.write("Total complete: 34%\n")
        flush_then_wait()
    elif(simulator_name == "STEPS_individual"):
        print("STEPS individual")
        sf_file,res_file,res_file_on_time,result = STEPS_individual.run(nrNodes,avgSendTime,simtime,plsize,db_size, fullcollision = fullcollision, is_update_db = is_update_db,max_dist
        sys.stderr.write("Total complete: 34%\n")
        flush_then_wait()
    elif(simulator_name =="STEPS_Boltzman"):
        print("STEPS boltzmann")
        sf_file,res_file,res_file_on_time,result = STEPS_Boltzman.run(nrNodes,avgSendTime,simtime,plsize,db_size,fullcollision = fullcollision, is_update_db = is_update_db,max_dist =
        sys.stderr.write("Total complete: 34%\n")
        flush_then_wait()
    elif(simulator_name == "STEPS_greedy"):
        print("STEPS greedy")
        sf_file,res_file,res_file_on_time,result = STEPS_greedy.run(nrNodes,avgSendTime,simtime,plsize,db_size,fullcollision = fullcollision,is_update_db = is_update_db,max_dist = max
        sys.stderr.write("Total complete: 34%\n")
        flush_then_wait()
    elif(simulator_name == "STEPS_p_random"):
        print("Dynamic P-Random")
        sf_file,res_file,res_file_on_time,result = STEPS_p_random.run(nrNodes,avgSendTime,simtime,plsize,fullcollision = fullcollision,max_dist = maxDist,regular_placement=regular_pla
        sys.stderr.write("Total complete: 34%\n")
        flush_then_wait()
    elif(simulator_name == "STEPS_random"):
        print("Dynamic Random")
        sf_file,res_file,res_file_on_time,result = STEPS_random.run(nrNodes,avgSendTime,simtime,plsize,fullcollision = fullcollision,max_dist = maxDist,regular_placement=regular_place
        sys.stderr.write("Total complete: 34%\n")
        flush_then_wait()
```

- Quelque codes de PyQT5 pour la création de boutons et label

```python
        self.label_11 = QtWidgets.QLabel(self.fixed)
        self.label_11.setGeometry(QtCore.QRect(320, 190, 21, 17))
        self.label_11.setText("")
        self.label_11.setPixmap(QtGui.QPixmap("Resources/inf3.png"))
        self.label_11.setObjectName("label_11")
        self.label_12 = QtWidgets.QLabel(self.fixed)
        self.label_12.setGeometry(QtCore.QRect(370, 280, 21, 17))
        self.label_12.setText("")
        self.label_12.setPixmap(QtGui.QPixmap("Resources/inf3.png"))
        self.label_12.setObjectName("label_12")
        self.label_13 = QtWidgets.QLabel(self.fixed)
        self.label_13.setGeometry(QtCore.QRect(350, 370, 21, 17))
        self.label_13.setText("")
        self.label_13.setPixmap(QtGui.QPixmap("Resources/inf3.png"))
        self.label_13.setObjectName("label_13")
        self.label_14 = QtWidgets.QLabel(self.fixed)
        self.label_14.setGeometry(QtCore.QRect(710, 100, 21, 17))
        self.label_14.setText("")
        self.label_14.setPixmap(QtGui.QPixmap("Resources/inf3.png"))
        self.label_14.setObjectName("label_14")
        self.label_15 = QtWidgets.QLabel(self.fixed)
        self.label_15.setGeometry(QtCore.QRect(660, 190, 21, 17))
        self.label_15.setText("")
        self.label_15.setPixmap(QtGui.QPixmap("Resources/inf3.png"))
        self.label_15.setObjectName("label_15")
        self.suivant1_12 = QtWidgets.QPushButton(self.fixed)
        self.suivant1_12.setGeometry(QtCore.QRect(540, 460, 111, 51))
        self.suivant1_12.clicked.connect(lambda: self.stackedWidget.setCurrentWidget(self.home))
        self.suivant1_12.setStyleSheet("QPushButton{\n"
    background-color: #425C59; /* blue */\n"
    border: none;\n"
    color: white;\n"
    padding: 4px 20px;\n"
    text-align: center;\n"
    text-decoration: none;\n"
    font-size: 16px;\n"
    margin: 4px 2px;\n"
    border-radius : 10px;\n"
}\n"
\n"
QPushButton:hover\n"
{\n"
    background-color : #fff;\n"
    color : black;\n"
    border : 2px solid #3E5555;\n"
}")
        self.suivant1_12.setObjectName("suivant1_12")
        self.suivant1_11 = QtWidgets.QPushButton(self.fixed)
```

- Code de fonction qui permet la génération de graphiques et fichier .CSV par rapport au temps de simulation

```python
def plot_single_run_ontime(res_file_ontime,fig_name,variable_list,simtime):
    data_results = np.loadtxt(res_file_ontime, delimiter=',', dtype=float)
    x = data_results[:,0]
    data_results = data_results[:,1:]

    times = np.linspace(0,simtime,101)
    res_10 = None
    res_avg = None
    res_90 = None
    for time_point in times:
        data_at_point = data_results[np.where((x==time_point))[0]]
        if data_at_point.shape[0]>0:
            data_avg = np.mean(data_at_point,axis=0).reshape((1,-1))
            data_10 = np.percentile(data_at_point, 10, axis=0).reshape((1,-1))
            data_90 = np.percentile(data_at_point, 90, axis=0).reshape((1,-1))
            if res_avg is None:
                res_avg = data_avg
                res_10 = data_10
                res_90 = data_90
            else:
                res_avg = np.vstack((res_avg,data_avg))
                res_10 = np.vstack((res_10,data_10))
                res_90 = np.vstack((res_90,data_90))
    times = times/1000
    for var in variable_list:
        i = global_varlist.index(var)
        fig = plt.figure(figsize=(12, 6.5))
        #plt.plot(data[:,0],data[:,i+1],label = simulators[j],linewidth = '2',marker = markers[j],linestyle=':',markersize=10)
        plt.plot(times,res_avg[:,i],label = 'avg',linewidth = '1')
        plt.plot(times,res_10[:,i],label = '10%-line',lineStyle=':',linewidth = '1')
        plt.plot(times,res_90[:,i],label = '90%-line',lineStyle=':',linewidth = '1')
        plt.xlabel('time(s)',  fontsize=18)
        plt.ylabel(var,  fontsize=18)
        plt.yticks(size = 14)
        plt.xticks(size = 14)
        plt.legend(loc='best', fontsize = 13)
        plt.grid(True)
        plt.savefig(fig_name+"_"+var+"_ontime.png",dpi=600,bbox_inches = 'tight')
```

- Code de la fonction qui permet la génération de graphiques et fichier .CSV par rapport a la distance

```
def plot_single_run(res_file,fig_name,variable_list,regular_placement = 0):
    data_results = np.loadtxt(res_file, delimiter=',', dtype=float)
    x = data_results[:,0]
    y = data_results[:,1]
    distance = np.sqrt(x**2+y**2)
    max_distance = max(distance)
    data_results = data_results[:,2:]
    if regular_placement > 0 :
        interval = max_distance/(regular_placement)
    else:
        interval = 200
    now_distance = 0
    x_plot = []
    res_10 = None
    res_avg = None
    res_90 = None
    while now_distance <= max_distance:
        data_in_range = data_results[np.where((distance>=now_distance-interval/2)&(distance<now_distance+interval/2))[0]]
        if data_in_range.shape[0]>0:
            x_plot.append(now_distance)
            data_avg = np.mean(data_in_range,axis=0).reshape((1,-1))
            data_10 = np.percentile(data_in_range, 10, axis=0).reshape((1,-1))
            data_90 = np.percentile(data_in_range, 90, axis=0).reshape((1,-1))
            if res_avg is None:
                res_avg = data_avg
                res_10 = data_10
                res_90 = data_90
            else:
                res_avg = np.vstack((res_avg,data_avg))
                res_10 = np.vstack((res_10,data_10))
                res_90 = np.vstack((res_90,data_90))
        now_distance += interval
    x_plot[-1] = max(distance)
    x_plot = np.array(x_plot)
    x_plot = x_plot.reshape((-1,1))
    res_avg = np.hstack((x_plot,res_avg))
    res_10 = np.hstack((x_plot,res_10))
    res_90 = np.hstack((x_plot,res_90))
    for var in variable_list:
        i = global_varlist.index(var)
        fig = plt.figure(figsize=(12, 6.5))
        #plt.plot(data[:,0],data[:,i+1],label = simulators[j],linewidth = '2',marker = markers[j],linestyle=':',markersize=10)
        plt.plot(res_avg[:,0],res_avg[:,i+1],label = 'avg',linewidth = '1')
        plt.plot(res_10[:,0],res_10[:,i+1],label = '10%-line',lineStyle=':',linewidth = '1')
        plt.plot(res_90[:,0],res_90[:,i+1],label = '90%-line',lineStyle=':',linewidth = '1')
        plt.xlabel('distance',  fontsize=18)
        plt.ylabel(var,  fontsize=18)
        plt.yticks(size = 14)
        plt.xticks(size = 14)
        plt.legend(loc='best', fontsize = 13)
        plt.grid(True)
        plt.savefig(fig_name+"_"+var+".png",dpi=600,bbox_inches = 'tight')
```

- Code qui permet la génération du graphique de la répartition des nœuds avec les SFs utilisés

```
def plot_sf(sf_file,fig_name):
    data_sf = np.loadtxt(sf_file, delimiter=',', dtype=float)
    sf_final = data_sf[:,-1]
    sf_init = data_sf[:,-2]
    fig, ax = plt.subplots(1,2,figsize=(12,5))
    for sf in range(7,13):
        to_scatter = data_sf[np.where(sf_init==sf)]
        ax[0].scatter(to_scatter[:,0],to_scatter[:,1],label = 'sf'+str(sf),s=1)
        ax[0].legend(loc='best', fontsize = 13)
        ax[0].set_title('init sf')
        to_scatter = data_sf[np.where(sf_final==sf)[0]]
        ax[1].scatter(to_scatter[:,0],to_scatter[:,1],label = 'sf'+str(sf),s=1)
        ax[1].legend(loc='best', fontsize = 13)
        ax[1].set_title('final sf')

    plt.savefig(fig_name+"_sf.png",dpi=600,bbox_inches = 'tight')
    #plt.show()
```

- Code qui permet l'ouverture des graphique a partir de l'outil MULANE

```
def on_combobox_changed2(self, value):
    f = open("figdata.txt","r")
    data = f.readline()
    data = data.split(",")
    simulator_name = data[0]
    avgSendTime = data[1]
    simtime = data[2]
    db_size = data[3]
    sim_times = data[4]
    f.close()
    path = './figs/single/'+simulator_name+'_'+'_'+str(avgSendTime)+'_'+str(simtime)+'_'+str(db_size)+'_'+str(sim_times)+'_'
    #print(simulator_name + ' ' +avgsendtime)
    self.meth = value
    if(value == "Total packets (distance)"):
        print(1)
        im = Image.open(path + 'total sent.png')
        im.show()
    elif(value == "Total packets (time)"):
        print(1)
        im = Image.open(path + 'total sent_ontime.png')
        im.show()

    elif(value == "Different packets (distance)"):
        print(2)
        im = Image.open(path +'new packets.png')
        im.show()
    elif(value == "Different packets (time)"):
        print(2)
        im = Image.open(path +'new packets_ontime.png')
        im.show()

    elif(value == "Resended packets (distance)"):
```

- Code de la barre de progression durant la simulation

```
class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(781, 521)
        Form.setStyleSheet("background-color : #E3EBEB;\n"
"")
        self.progressBar = QtWidgets.QProgressBar(Form)
        self.progressBar.setGeometry(QtCore.QRect(10, 10, 761, 31))
        self.progressBar.setStyleSheet("QProgressBar{\n"
"    border: solid grey;\n"
"    border-radius: 15px;\n"
"    color: black; \n"
"    text-align : right;\n"
"}\n"
"QProgressBar::chunk {\n"
"    background-color: #D6A218;\n"
"    border-radius :15px;\n"
"}")
        self.progressBar.setProperty("value", 0)
        self.count = 0
        self.progressBar.setMaximum(100)
        self.progressBar.setObjectName("progressBar")
        self.plainTextEdit = QtWidgets.QPlainTextEdit(Form)
        self.plainTextEdit.setGeometry(QtCore.QRect(10, 50, 761, 461))

        self.plainTextEdit.setStyleSheet("background-color : white;")
        self.plainTextEdit.setObjectName("plainTextEdit")
        """

        """
        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)
```

- Code de la fonction qui permet de déterminer quel méthode lancer depuis MULANE

```
def what_start(self):
    if(self.simName == "lorasim"):
        self.start_random()
    if(self.simName == "STEPS_full"):
        self.start_steps()
    if(self.simName == "STEPS_individual"):
        self.start_steps()
    if(self.simName == "STEPS_greedy"):
        self.start_egreedy()
    if(self.simName == "STEPS_random"):
        self.start_random()
    if(self.simName == "STEPS_p_random"):
        self.start_prandom()
    if(self.simName == "STEPS_Boltzmann"):
        self.start_boltz()
```

Les tests des performances des différentes méthode se faisait suivant la procédure suivante :

For j from 1-25:
Algorithme_de_la_méthode_en_test if
j in [1,5,10,15,20,25]:
Data_of_mtd_j += result of sim