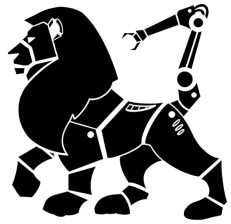


Projet Final - Algorithmes Répartis

Distributed Algorithm for Dynamic Network with LEACH and LEACH-C

30 Novembre 2024

Elèves :
Ahmad Harkous - Lise Pedemonte - Rémy Decourcelle - Emil Toulouse
-
Promotion EPITA 2025

Table des matières

1 Introduction

Dans ce projet, nous allons traiter de deux protocoles bien connus en algorithmes répartis : LEACH et LEACH-C. LEACH (Low Energy Adaptive Clustering Hierarchy) et sa version améliorée LEACH-C qui sont des algorithmes couramment utilisés dans les réseaux de capteurs sans fil pour optimiser la consommation d'énergie et prolonger la durée de vie des réseaux.

Nous devrons dans un premier temps implémenter ces deux algorithmes en utilisant SimPy, une bibliothèque de simulation en Python, pour créer des modèles de simulation. Nous devrons ensuite créer une version dynamique de ces modèles afin de simuler un environnement où les capteurs sont placés sur des cibles mouvantes.

Enfin, nous comparerons les métriques résultantes de ces simulations afin de déterminer quel modèle et quelle implémentation, entre LEACH et LEACH-C, se révèlent être les plus efficaces et adaptées à un contexte dynamique de réseaux de capteurs sans fil.

Notre notebook est consultable en cliquant sur le lien suivant : Distributed Algorithm for Dynamic Network with LEACH and LEACH-C

2 Représentation d'un Noeud, Paramètres et Constantes

En SimPy, nous avons créé une classe Node qui représente donc un capteur, un nœud. Nous lui avons attribué tout ce qui caractérise un capteur, c'est-à-dire :

- un ID
- ses coordonnées (x,y)
- son niveau d'énergie
- son statut (Normal, Cluster-Head, Mort)
- le Cluster-Head auquel il est associé
- un décompte du temps avant qu'il puisse être réélu Cluster-Head

Cette classe nous sert à simuler des capteurs que ce soit pour LEACH ou LEACH-C. Un nœud peut se déplacer (fonction move) et on peut recalculer sa dépense d'énergie (fonction consume_energy)

En ce qui concerne les paramètres et les constantes, ils sont aussi similaires aux deux algorithmes.

```
xm, ym => Dimension du champ (100, 100)
sink => La base, située aux coordonnées (0,-100)
n => Le nombre de capteurs
p => La probabilité optimale de devenir un cluster-head
Eo, E_elec, E_fs, E_mp, E_da => Constantes sur l'énergie dépensée dans divers cas
d_o => Le "seuil" de distance (np.sqrt(E_fs / E_mp))
packetLength => La taille d'un packet
rmax => Le nombre maximum de rounds
```

Pour nos tests, nous pouvons essayer de faire varier le nombre de capteurs, la probabilité optimale de devenir un cluster-head, la taille d'un packet et le nombre maximal de rounds.

3 Implémentation de LEACH

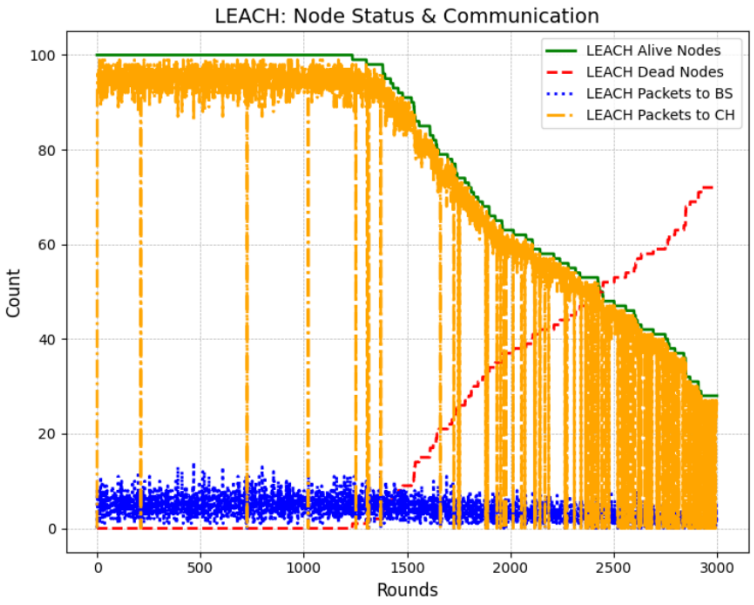
Comme l'algorithme de LEACH était déjà implémenté en Python, la première étape pour nous a été de comprendre cet algorithme, comprendre le rôle de chaque variable puis le déroulé du code.

Une fois cela compris, nous avons eu à le traduire en SimPy. Cette étape s'est révélée être assez rapide. Nous avons créé une classe "LEACHSimulation". Pour faire une simulation LEACH, nous avons besoin des informations suivantes, qui sont donc les attributs de notre classe :

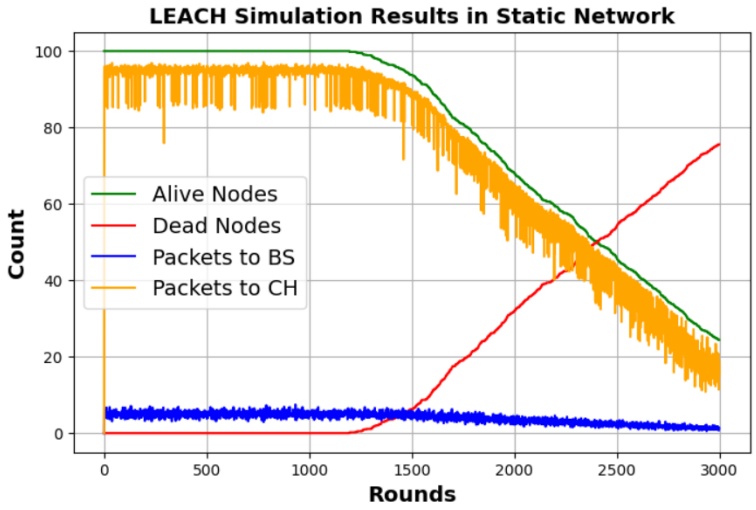
- La liste des noeuds de notre simulation
- Les informations sur notre base
- Le round actuel
- Les différentes statistiques que l'on veut garder en mémoire pour chaque round (le nombre de noeuds vivants, morts, le nombre de paquets envoyés aux cluster heads et à la base)

Plusieurs actions sont possibles dans notre simulation LEACH. Une première étape consiste à élire les cluster-head (fonction elect_cluster_heads), ensuite pour chaque nœud, il faut lui attribuer un cluster et calculer la consommation d'énergie provoquée par l'envoi d'un message à ce cluster-head (fonction assign_clusters). Enfin il faut calculer l'énergie perdue par les cluster-head lors de l'envoi de tous les messages vers la base.

Nous avons premièrement repris exactement l'algorithme de LEACH qui nous était donné et voici les résultats obtenus :



Afin de lisser le nombre de cluster-head pour ne pas tomber régulièrement sur des cas où aucun n'est élu, nous avons testé de répéter la simulation 10 fois et avons fait la moyenne de ces essais. Voici les résultats obtenus :



Cependant lors des tests avec les métriques, nous nous concentrerons sur les résultats donnés pour un round, afin de pouvoir par exemple déterminer le nombre de rounds où aucun cluster-head n'est élu.

4 Implémentation de LEACH-C

Voici les différentes approches que nous avons implémentées pour LEACH-C dans un réseau dynamique.

4.1 K-means Clustering

Nous avons d'abord testé l'algorithme K-means. Cet algorithme permet de regrouper les nœuds en clusters en fonction de leur proximité spatiale. Il s'agit d'un algorithme de partitionnement non supervisé.

Étapes principales de K-means :

1. **Initialisation :** Choisir aléatoirement k centres (centroïdes) initiaux.
2. **Assignation des points :** Chaque nœud est affecté au cluster dont le centroïde est le plus proche, selon la distance euclidienne.
3. **Mise à jour des centroïdes :** Calculer le barycentre de chaque cluster et mettre à jour les centroïdes.
4. **Répétition :** Répéter les étapes 2 et 3 jusqu'à convergence (c'est-à-dire lorsque les centroïdes ne changent plus).

Avantages

— Rapidité et simplicité.
— Facilité d'intégration avec LEACH-C.

Limites

— Sensible à l'initialisation des centroïdes.
— Peut produire des clusters de tailles déséquilibrées.

4.2 Cuckoo Search Algorithm

Par la suite, nous avons utilisé l'algorithme Cuckoo Search (CS), une méthode d'optimisation inspirée par le principe de la recherche de Lévy.

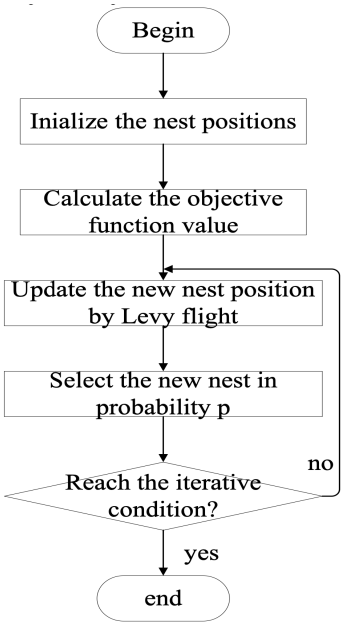
Nous nous sommes basés sur l'article de recherche Cuckoo Search.

Étapes principales de l'algorithme de recherche par Lévy pour la sélection des clusters

1. **Initialisation :** Générer une population initiale de solutions candidates, où chaque solution représente un ensemble de positions de capteurs dans le réseau. Ces solutions initiales définissent la répartition des capteurs dans l'environnement.
2. **Génération des solutions :** Pour chaque solution, une nouvelle position est générée en utilisant le vol de Lévy, selon la relation suivante :

   $$x_i^{t+1} = x_i^t + \alpha \cdot L(s)$$

   où α est un facteur d'échelle et $L(s)$ représente la distribution de Lévy. Cette étape permet aux capteurs d'explorer de nouvelles positions dans le réseau pour maximiser la couverture ou réduire la consommation d'énergie.
3. **Évaluation et remplacement :** Chaque position générée est évaluée en fonction de critères d'efficacité tels que la couverture du réseau ou la consommation d'énergie. Si une solution est meilleure que l'ancienne, elle remplace cette dernière.
4. **Abandon des solutions sous-optimales :** Une fraction p_a des solutions les moins efficaces est abandonnée et remplacée par de nouvelles solutions générées de manière aléatoire dans l'espace de recherche. Cela permet d'éviter de rester bloqué dans des solutions sous-optimales.
5. **Itérations :** Répéter le processus de génération, d'évaluation et de remplacement des solutions jusqu'à ce qu'un critère de convergence soit atteint, ou jusqu'à un nombre maximal d'itérations. Cela permet aux capteurs de se réorganiser dans des positions optimales dans le réseau, en maximisant la couverture et en réduisant la consommation d'énergie.



Avantages

— Efficace pour explorer des espaces de recherche complexes.
— Capacité à éviter les minima locaux grâce au vol de Lévy.

4.3 Comparaison des Algorithmes

— **LEACH-C avec K-means :** Utilise une approche centralisée simple et efficace pour le clustering, mais peut produire des résultats sous-optimaux dans des réseaux complexes.
— **LEACH-C avec Cuckoo Search :** Plus puissant pour les grands réseaux, mais au coût d'une complexité computationnelle accrue.

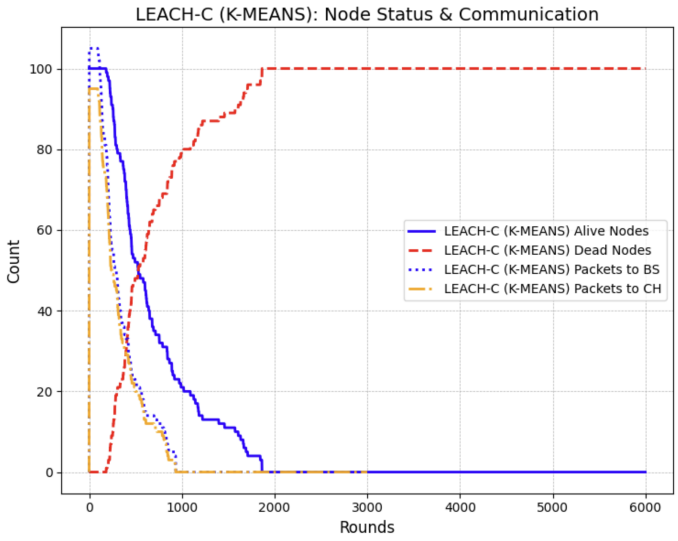


FIGURE 1 – Nous remarquons que, pour LEACH-C K-MEANS, le nombre de paquets envoyés au BS est élevé.

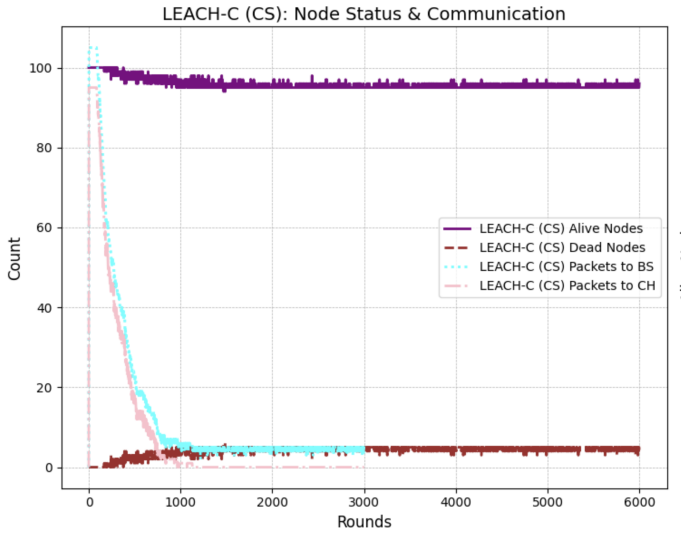


FIGURE 2 – Nous observons de meilleures performances que l'algorithme K-means en ce qui concerne le nombre de nœuds vivants (alive nodes).

5 Analyse des Performances et des Métriques

5.1 Comparaison des performances entre LEACH et les deux algorithmes utilisés dans LEACH-C

Nous avons testé les performances de LEACH en comparaison avec les deux algorithmes utilisés dans LEACH-C, à savoir l'algorithme K-means et l'algorithme Cuckoo Search, sur le scénario suivant :

$(l = 2000, p = 0.05, n = 100)$ où l représente la taille du paquet (packet length), p est la probabilité de devenir un chef de cluster (cluster-head probability), et n est le nombre total de nœuds (nodes).

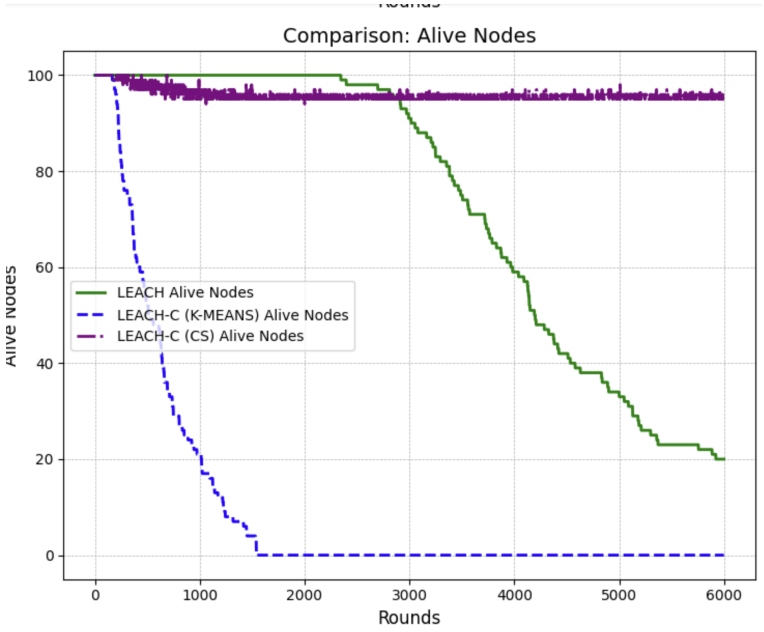
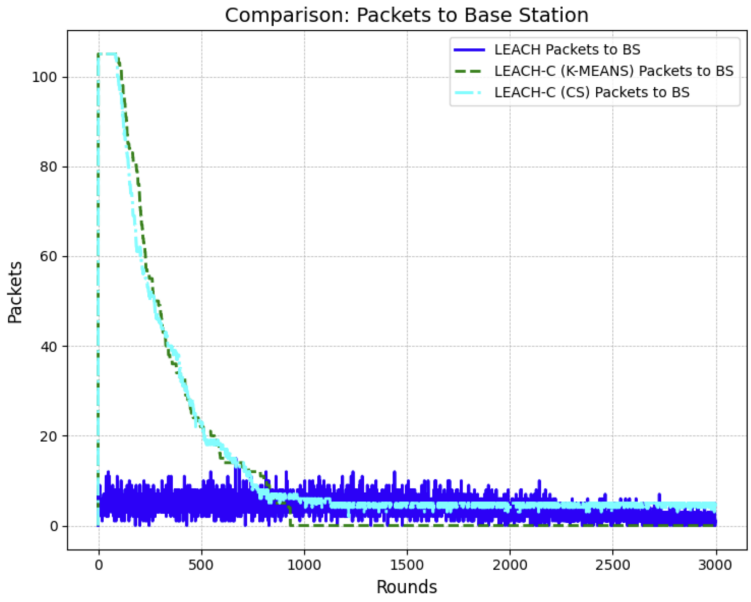


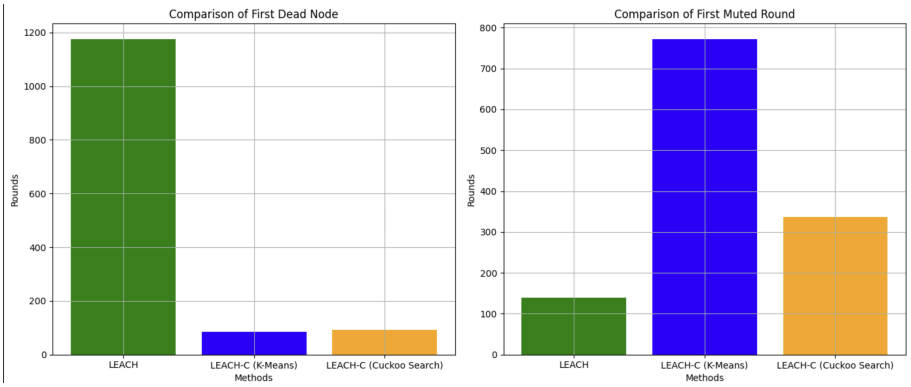
FIGURE 3 – LEACH-C offre une meilleure longévité globale car il maintient un nombre de nœuds vivants beaucoup plus élevé sur une durée prolongée.



Le graphique montre le nombre de paquets envoyés à la station de base par round :
— Pour LEACH, le nombre de paquets reste environ constant.
— Avec LEACH-C (K-means), les paquets envoyés diminuent rapidement dès le début, ce qui montre une inefficacité énergétique.
— Avec LEACH-C (Cuckoo Search), le nombre de paquets diminue plus lentement et ne descend pas jusqu'à zéro.

Conclusion : LEACH-C (Cuckoo Search) est plus efficace au début pour maintenir la communication avec la station de base.

Comparons à présent l'arrivée du premier nœud mort et du premier round où aucun cluster-head n'est élu :



Premier nœud mort (FDN) :

Analyse : LEACH offre une meilleure durée de vie initiale. Cependant, comme c'est un algorithme probabiliste, les performances peuvent ne pas toujours être aussi bonnes.

Durée d'activité réseau (First Muted Round, FMR) :

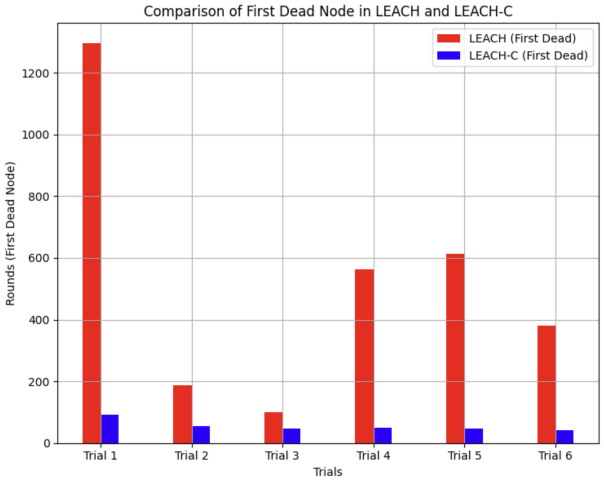
Analyse : LEACH-C (K-MEANS) maintient une activité fonctionnelle beaucoup plus longue, grâce à un équilibre énergétique initial. Cela montre une bonne stabilité dans l'élection du CH.

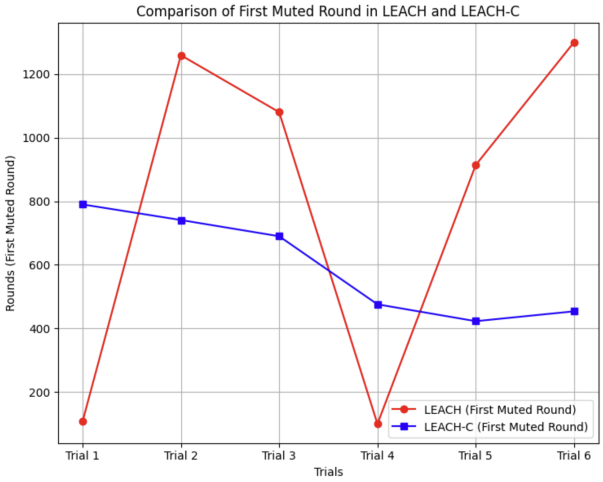
5.2 Scénarios de simulation

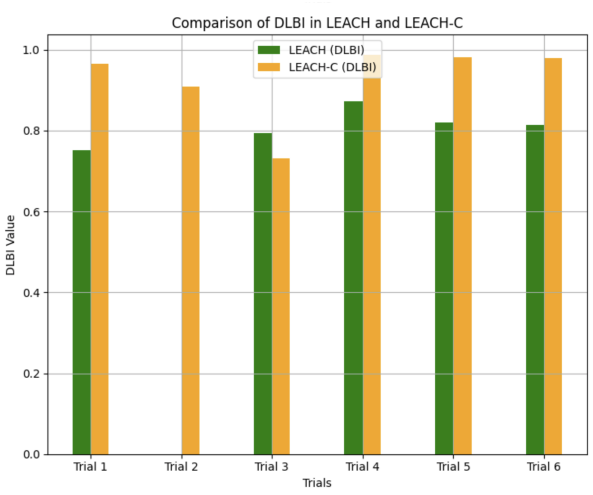
Nous avons également effectué la simulation avec différents paramètres de test en utilisant la configuration du sujet :

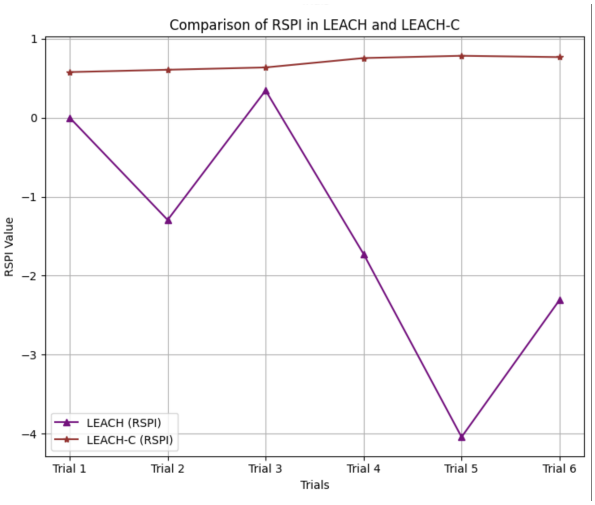
$$(l, p, n) \in \{(2000, 0.05, 100), (2000, 0.5, 100), (2000, 0.95, 100), (4000, 0.05, 100), (4000, 0.05, 200), (4000, 0.1, 200)\}$$

Nous obtenons les résultats suivants :

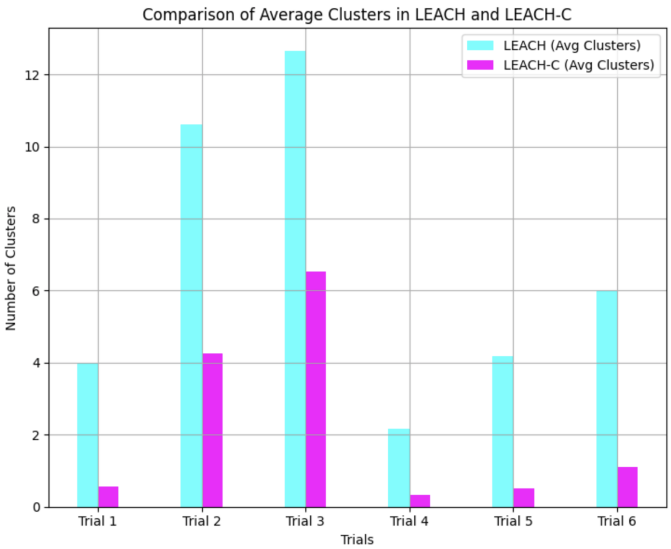

Comparison of First Dead Node in LEACH and LEACH-C
LEACH (First Dead)
LEACH-C (First Dead)
Rounds (First Dead Node)
Trials
Trial 1
Trial 2
Trial 3
Trial 4
Trial 5
Trial 6


Comparison of First Muted Round in LEACH and LEACH-C
LEACH (First Muted Round)
LEACH-C (First Muted Round)
Rounds (First Muted Round)
Trials


Comparison of DLBI in LEACH and LEACH-C
LEACH (DLBI)
LEACH-C (DLBI)
DLBI Value
Trials


Comparison of RSPI in LEACH and LEACH-C
LEACH (RSPI)
LEACH-C (RSPI)
RSPI Value
Trials

Trials


Comparison of Average Clusters in LEACH and LEACH-C
LEACH (Avg Clusters)
LEACH-C (Avg Clusters)
Number of Clusters
Trials

En observant ces différents scénarios, nous pouvons conclure que LEACH-C est la meilleure solution à utiliser. En effet, cet algorithme offre des résultats plus stables que ceux de LEACH et généralement meilleurs.

Le seul inconvénient de LEACH-C par rapport à LEACH est que son temps d'exécution est plus long.

6 Comparaison entre Réseaux Statiques et Dynamiques

Les différences que nous observons entre l'implémentation de ces algorithmes pour une solution statique et une solution dynamique sont surtout observables pour LEACH-C.

Cela est dû au fait que LEACH est un algorithme probabiliste, nous n'avions donc pas à nous soucier de la manière d'élire un cluster car cela ne se fait qu'avec des probabilités. Ainsi, nous n'avions pas à prendre en compte dans notre implémentation des axes de réflexions comme le fait de sélectionner des nœuds ayant le plus de voisins proches possibles ou des cluster-heads qui permettraient d'avoir la meilleure économie en énergie dépensée pour l'envoi de messages.

En ce qui concerne LEACH-C, nous avons ressenti une différence dans les critères de sélection des cluster-heads. Comme les capteurs sont en mouvement, il ne suffit pas simplement d'élire des cluster-heads en fonction de leur énergie restante mais il faut aussi penser au voisinage de ces derniers, à une méthode pour trouver quels sont les nœuds les plus propices à faire économiser de l'énergie à un maximum de capteurs.

7 Conclusion

Ce projet nous a donné l'opportunité d'explorer en profondeur les protocoles LEACH et LEACH-C qui sont essentiels pour les réseaux de capteurs sans fil. En implémentant ces algorithmes dans un environnement dynamique à l'aide de SimPy, nous avons pu simuler des scénarios réalistes où les capteurs sont placés sur des cibles mouvantes.

À travers nos analyses et comparaisons des performances et des métriques, nous avons pu évaluer l'efficacité de LEACH et de LEACH-C dans un contexte dynamique. Les résultats obtenus ont mis en lumière les forces et les faiblesses de chaque algorithme. Nous avons donc pu déterminer quel modèle est le plus adapté pour optimiser la consommation d'énergie et prolonger la durée de vie des réseaux de capteurs sans fil.

L'implémentation de différentes approches telles que le clustering K-means et l'algorithme Cuckoo Search dans LEACH-C a enrichi notre compréhension des stratégies possibles pour améliorer les performances des réseaux dynamiques.