Université Pierre et Marie Curie

projet MANET

# Compte-rendu : Projet ARA 2019–2020, Mobile Ad hoc NETworks

*Auteurs :*
Mariia Popova, William Fabre

*Professeur :*
Monsieur Lejeune,Favier

Année 2019-2020

# Table des matières

# Introduction

L'élection du leader est un de problèms fondamentaux de l'algorithmique répartie. Dans les réseaux mobiles, où la topology se changes tout le temps, ce problème de l'élection devient beaucoup plus compliqué. Les buts principaux de ce projet sont d'implémenter et de comparer deux algorithmes d'élection du leader différents, ainsi qu'étudier leurs comportement dans un réseau MANET. Ensuite, comparer les performances des algorithmes, notamment le nombre de messages échangés et un taux d'instabilité, en fonction de rayon d'émission du node.

## Préparation du projet et installation

Pour utiliser le projet il suffit d'utiliser le script :

Listing 1.1 – `../README`

```
1
2  This archive is composed of:
3  - the src/ directory containing the code source
4  - the launch.sh script to compile and run the programm
5  - the testscope.sh script to get the perfs
6  - the final rapport
7
8  To compile run the programm, you need java 5 installed and peersim.
9  ----------------------------------------------------------------------
10
11 [1] Compile and run:
12
13 $ ./launch.sh -p path_of_parent_directory_of_peersim -m mode_of_config
14
15 [2] Example:
16 $ ./launch.sh -p /home/user/ara -m 2
17
18 ----------------------------------------------------------------------
19
20 [3] Graphical Monitoring
21
22 To get the Graphical Monitoring of algorithme 1 do :
23 $ ./launch.sh -p path_of_parent_directory_of_peersim -m 1
24
25 To get the Graphical Monitoring of algorithme 2 do :
26 $ ./launch.sh -p path_of_parent_directory_of_peersim -m 2
27
28 (for more: ./launch.sh -h)
29
30 ----------------------------------------------------------------------
31 [4] Get perfs
32 testscope.sh script will create a directory 'test' with results,
33 the number of launches N and the step of scope augmentation you can set in params
34 (for more: ./testscope.sh -h)
35
36 To get the Number of messages and instability
37 monitoring of algorithme 1 do :
38 $ ./testscope.sh -p path_of_parent_directory_of_peersim -m 3
39
40 To get the Number of messages and instability
41 monitoring of algorithme 2 do :
42 $ ./testscope.sh -p path_of_parent_directory_of_peersim -m 4
43
44
45 ------------------------------
46 Best regards,
47 Mariia and William
48 26/01/2020
```

# Exercice 1 – Implémentation d'un MANET dans Peer-Sim

## Question 1

*En analysant le code de la classe PositionProtocolImpl, donnez l'algorithme général de déplacement d'un nœud. Il ne vous est pas demandé de copier/coller le code dans cette question.*

Le but de cet algorithme est de deplacer un noeud dans l'espace de deux dimenssions en direction d'une position choisi en fonction de la strategie de deplacement. L'algorithme de deplacement du noeud fonctionne comme suit :

Si le noeud ne bouge pas alors il est mis en mouvement. C'est a dire q'on lui affecte une vitesse `vi` aleatoire bornee par un `MinSpeed` et un `MaxSpeed`. Puis on lui choisi une desination qui est de type Position (element qui possede deux coordonnes). Il faut maintenant calculer la distance en metres entre le point de depart, qui est la position du Node et la position d'arrivee, que nous appellerons destination ; Grace a la formule suivante :

$$Soit \quad (dest, cur) \in Position^2, \tag{2.1}$$

$$distance = (dest.x - cur.x)^2 + (dest.y - cur.y)^2 \tag{2.2}$$

C'est la distance total pour arriver a destination. Maintenant il nous faut calculer la distance totale parcourable en une unite de temps. Elle est egale a la vitesse dans la metrique du systeme, c'est a dire, la vitesse `vi` en metres par secondes qu'il faut diviser par mille pour avoir des metre par millisecondes.

$$Soit \quad (vi) \in Vitesse \quad (m * s^{-1}), \tag{2.3}$$

$$distance\_to\_next = vi/1000 \tag{2.4}$$

Si la distance totale est inferieur a la distance parcourable en une unite de temps il faut alors calculer une nouvelle Position intermediaire et donc un x et un y :

$$Soit \quad (next_x, next_y) \in long^2 \tag{2.5}$$

$$Soit \quad (\theta_1, \theta_2) \in degree^2 \tag{2.6}$$

$$next\_x = (distance\_to\_next * \frac{(dest.x - cur.x)}{((dest.x - cur.x)^2 + (dest.y - cur.y)^2)}) + cur.x \tag{2.7}$$

$$\iff \tag{2.8}$$

$$next\_x = (distance\_to\_next * \sin\theta_1) + cur.x \tag{2.9}$$

$$\tag{2.10}$$

$$\tag{2.11}$$

$$next\_y = (distance\_to\_next * \frac{(dest.y - cur.y)}{((dest.x - cur.x)^2 + (dest.y - cur.y)^2)}) + cur.y \tag{2.12}$$

$$\iff \tag{2.13}$$

$$next\_y = (distance\_to\_next * \cos(\theta_2)) + cur.x \tag{2.14}$$

$$\tag{2.15}$$

Sinon, si la distance total est egale ou inferieur a la distance parcourable en une unite de temps, alors on affete les coordonnes de la position destination a notre Node. On vient verifier si la position a change, pour affecter une pause ; Sinon on incremente le temps du simulateur d'une unite de temps.

## Question 2

*Testez le simulateur en prenant la stratégie FullRandom comme SPI et SD. Le contrôleur graphique sera déclenché toutes les unités de temps, son timeslow pourra être environ de 0.0002. Le seul protocole à renseigner pour ce contrôleur est le PositionProtocol de la simulation, les autres sont pour l'instant optionnels et sans objet. Normalement vous devez voir graphiquement des points verts se déplacer sur l'écran. N'oubliez pas d'amorcer les instances de PositionProtocol via un module d'initialisation. Vous répondrez à cette question en donnant le contenu de votre fichier de configuration.*

Listing 2.1 – `../src/ara/config`

```
network.size 75
random.seed 10
simulation.endtime 50000

protocol.position PositionProtocolImpl
protocol.position.maxspeed 20
protocol.position.minspeed 5
protocol.position.width 1200
protocol.position.height 1200
protocol.position.pause 200

protocol.emit EmitterProtocolImpl
protocol.emit.latency 90
protocol.emit.scope 100
protocol.emit.variance TRUE

protocol.neighbor NeighborProtocolImpl
protocol.neighbor.periode 300
protocol.neighbor.timer 400
protocol.neighbor.listener TRUE

protocol.election GVLElection
protocol.election.periode 5000

initial_position_strategy FullRandom
initial_position_strategy.positionprotocol position
initial_position_strategy.emitter emit

next_destination_strategy FullRandom
next_destination_strategy.positionprotocol position
next_destination_strategy.emitter emit
next_destination_strategy.random_dest_period 20000

control.monitor GraphicalMonitor
control.monitor.electionprotocol election
control.monitor.positionprotocol position
control.monitor.emitter emit
control.monitor.neighborprotocol neighbor
control.monitor.monitorableprotocol election
control.monitor.time_slow 0.02
control.monitor.from 0
control.monitor.until 50000
control.monitor.step 1

init.i InitialisationGVLElection
```

## Question 3

*Codez une classe implémentant l'interface Emitter. Testez de nouveau avec le moniteur graphique et assurez-vous que les portées sont représentées (cercle en bleu).*

Ceci est visible dans la classe Emitter.java

## Question 4

*Codez une classe implantant l'interface NeighborProtocol.*

Ceci est visible dans la classe NeighborProtocol.java

## Question 5

*Testez votre code, et remarquez sur le moniteur graphique l'apparition d'un lien graphique lorsque deux nœuds deviennent voisins.*

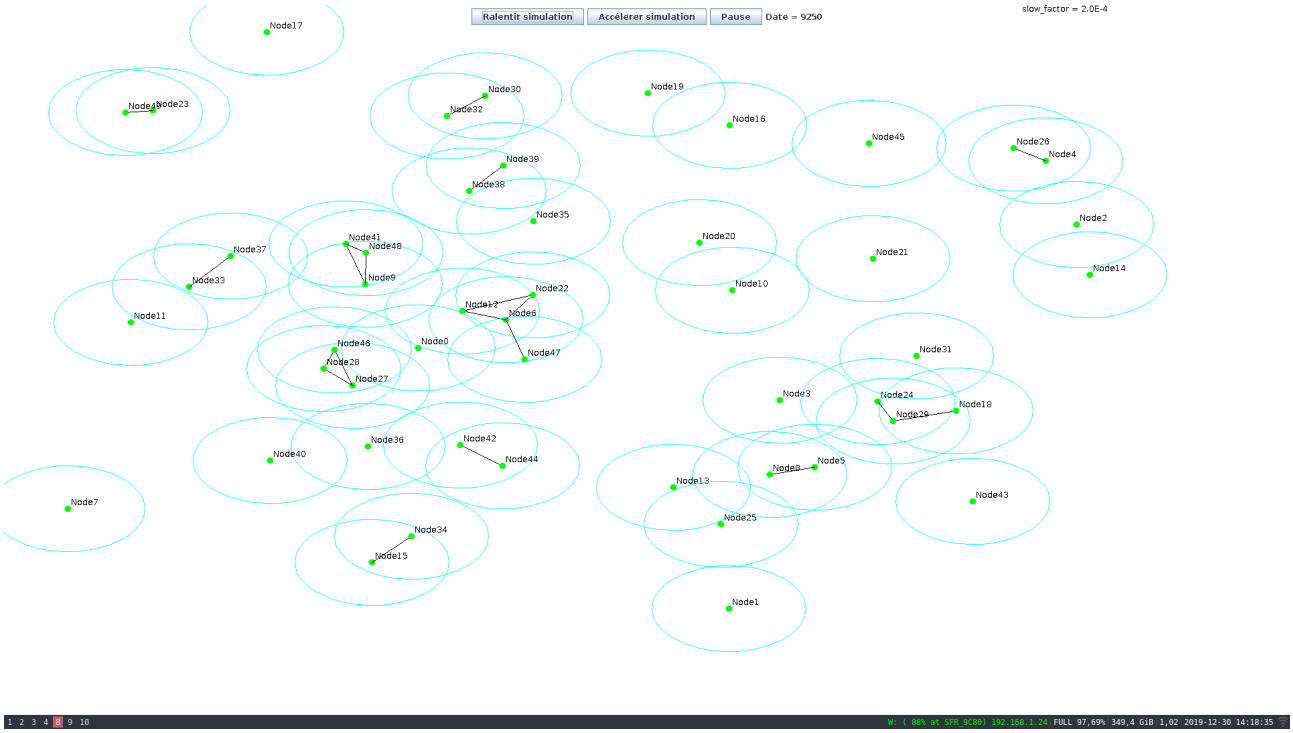


FIGURE 2.1 – Representation de l'apparition de lien lors de la simulation

Comme on peut le voir dans la figure , il y a bien apparition de liens entre les différents node s'ils sont respectivement dans le scope l'un de l'autre avec le scope visible par représentation d'un cercle autour du node.

## Question 6

*En analysant les codes des classes gérant le positionnement des nœuds qui font appel à un tirage aléatoire, on peut remarquer qu'ils utilisent un objet Random qui leur est dédié (attribut my_random initialisé au random de la classe PositioningConfiguration). Quelle en est la raison ?*

Le but de nos simulations est d'evaluer un ensemble d'algorithme avec differents types de placement, deplacements de nodes. la reproductibilite de ces simulations est importante pour des soucis de debug ou de l'analyse. Maitriser tres exactement la seed de randomization est un point cle pour avoir des algorithmes avec des placements aleatoires mais maitrises. On peut voir que la classe `PositioningConfiguration` recuperer `PAR_SEED_POSITIONING` depuis le fichier de configuration. De plus si d'autres classes utilisent ce même randomizer il faut que je comportement reste déterministe.

## Question 7

*Prenez connaissance des différentes stratégies et pour chacune expliquez ce qu'elle fait.* Pour chacune les strategies de placement de noeuds dans l'espace il existe deux types de placement : Un placement dit initial et un deplacement qui sont represente par deux fonctions : `getInitialPosition` et `getNextDestination`. Les strategies implementent ces fonctions si elles implementent leur interfaces : `InitialPositionStrategy` et `NextDestinationStrategy` ceci sera precise dans le titre des sections suivantes.

### Strategie 1 : ConnectedRandom (InitialPositionStrategy, NextDestinationStrategy)

`getInitialPosition` et `getNextDestination`

Le schema suivant s'applique pour definir une position initial ou trouver une nouvelle position a un noeud.

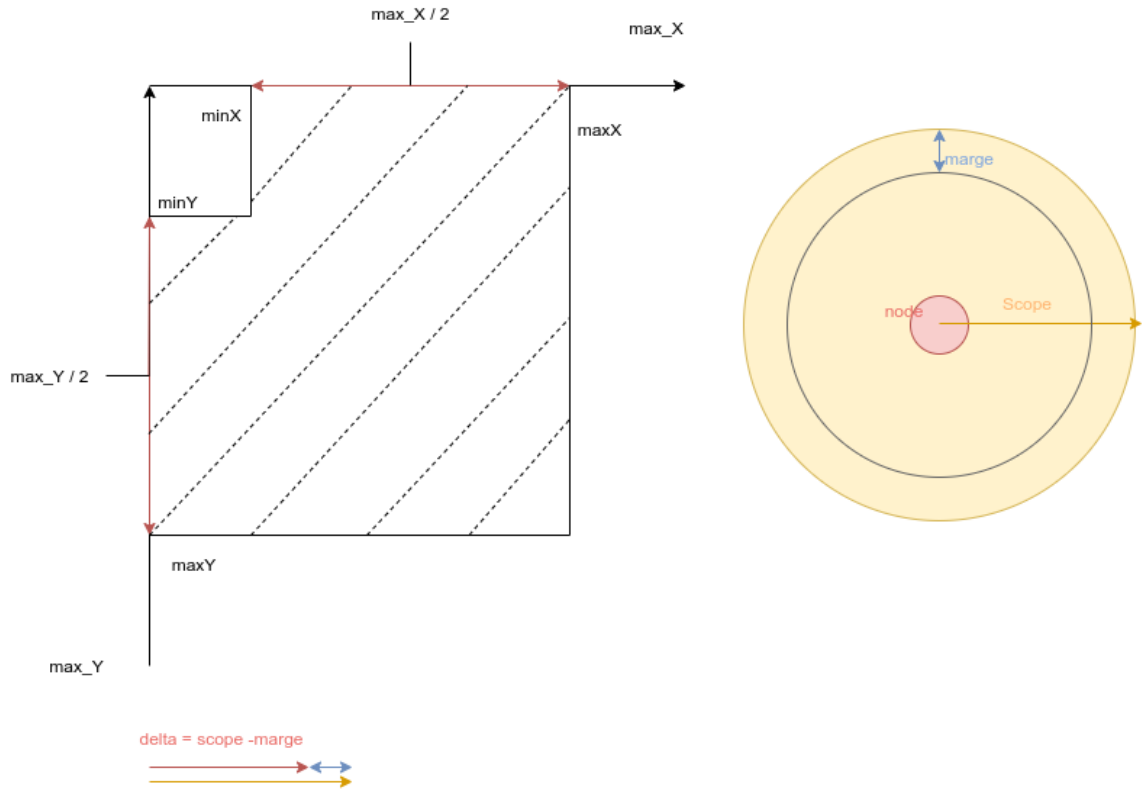


FIGURE 2.2 – Representation du calcule pour le placement connected Random

Les elements de la figure sont :

— A gauche, la grille de placement des noeuds avec `max_X` et `max_Y` qui sont des parametres du systeme. On peut aussi voir `max_X / 2` et `max_Y / 2`. Pour finir, `maxX, maxY, minX, minY` correspondent aux nouveaux min et max calcules par la fonction en additionnant ou soustrayant delta visible graphiquement. La zone hachuree correspond a l'ensemble des coordonnees possible pour une position randomizee.

— A droite, On peut observer le scope d'un noeud et un parametre de marge de l'algorithme Connected Random.

— En bas on peut observer graphiquement la taille de scope-marge

La figure ci dessus represente l'implementation pour les fonctions `getInitialPosition` et `getNextDestination`, il existe un argument de fonction speed qui n'est pas utilise malgre le fait que l'interface `NextDestinationStrategy` explique : "Retourne une nouvelle de destination du neoud host en fonction de la vitesse speed".

### Strategie 2 : FullRandom (InitialPositionStrategy, NextDestinationStrategy)

`getInitialPosition`

La strategie pour la position initiale est de creer en une table associative a partir de tous les noeuds du reseau et grace a la fonction `Network.get(i)` ("Returns node with the given index. Note that the same node will normally have a different index in different times. le ieme dans la liste des noeuds", ce qui signifie un ajout de randomization) affecter a chaque node le resultat de la fonction Destination decrite ci-apres.

`getNextDestination`

La strategie de choix de la destination suivante consiste a affecter une nouvelle position nextX et nexty tel que :

$$position = (my\_random.nextDouble() * maxX, \quad my\_random.nextDouble() * maxY) \tag{2.16}$$

### Strategie 3 : InitialPositionConnectedRing (InitialPositionStrategy)

`getInitialPosition`

Le but de cette fonction est de calculer la position des points en fonction d'un centre pour les placer autour de celui-ci. Elle utilise : `centre.getNewPositionWith` qui "Calcul une nouvelle position à partir d'un module et d'un angle depuis la position courante". Explicitons les calcules, nous reutiliserons les variables `maxX, maxY, scope maxX/2 et maxY/2`, nous ajoutons une variable centre qui est definie par les coordonnes `maxX/2,MaxY/2`, la taille du reseau, appelee `size` et une variable rayon qui nous aidera a placer les points autour du centre.

$$Soit \quad rayon \in Entier, \tag{2.17}$$

$$rayon = min(size * (scope/8), maxY/2.5) \tag{2.18}$$

$$\tag{2.19}$$

$$exemple \quad : \tag{2.20}$$

$$size = 10 \quad scope = 32 \quad maxY = 500 \tag{2.21}$$

$$rayon = min(10 * 4, 500/2.5) \tag{2.22}$$

$$rayon = min(40, 200) = 40 \tag{2.23}$$

Le calcule du rayon differe si les noeuds sont paires ou impaires. voici la fin du calcule s'ils sont paire :

$$Soit \quad rayon \in Entier, \tag{2.24}$$
$$Soit \quad numero\_de\_node \in Entierpaire, \tag{2.25}$$
$$rayon = rayon - (scope/2) \tag{2.26}$$
$$\tag{2.27}$$
$$exemple \quad : \tag{2.28}$$
$$rayon = 40 - (32/2) = -210 \tag{2.29}$$

Il faut maintenant un `delta_angle` qui servira au calcule de la position cournte de la fonction `center.getNewPositionWith`,

$$Soit \quad delta\_angle \in Entier, \tag{2.30}$$
$$delta\_angle = \frac{2 * \pi}{size} \tag{2.31}$$
$$\tag{2.32}$$
$$exemple \quad : \tag{2.33}$$
$$size = 10 \tag{2.34}$$
$$delta\_angle = \frac{2 * \pi}{10} = 0.6283 \quad (approx) \tag{2.35}$$
$$\tag{2.36}$$

l'appelle a la fonction :

```
1     centre.getNewPositionWith(rayon, delta_angle * host.getID())
2
```

placera donc les noeuds de 0 a 9 ainsi :

### Strategie 4 : InitialPositionConnectedRing (InitialPositionStrategy)

`getInitialPosition`

Le but de cette strategie est de definir une position initiale pour tous les noeuds en une passe. C'est un chainage entre les noeuds vont se placer en fonction d'un voisin. Il existe un traitement specifique pour le tout premier noeud. L'algorithme redefini des bornes pour positionner le premier noeud. appelons les valeurs definies par defaut `old_maxX` et `old_maxY`. Cette nouvelle position est ajoutee a une `hashmap<id, Position>` qui s'appelle `initial_position` qui est partagee par toutes les instances de cette classe.

$$Soit \quad (old\_maxX, old\_maxY) \in Entier^2 \tag{2.37}$$
$$Soit \quad (minX, maxX, minY, maxY) \in Entier^4 \tag{2.38}$$
$$Soit \quad p \in PositionminX = old\_maxX/3; \quad maxX = old\_maxX; \tag{2.39}$$
$$minY = old\_maxY/3; = old\_maxY; \tag{2.40}$$
$$p = ([minX, maxX], [minY, maxY]) \tag{2.41}$$
$$\tag{2.42}$$

Pour tous les autres noeuds, ils choisissent un voisin qui possede un id egale a la taille de la hashmap `initial_position`. ils choisissent au hasard, un angle, une distance bornee avec une verification que ces bornes sont bien definies grace a une fonction `bound`. Pour finir ils s'ajoutent a la hashmap partagee.

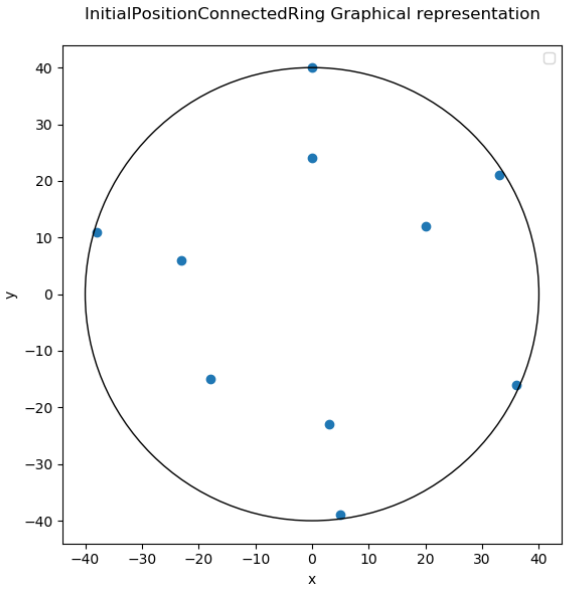


FIGURE 2.3 – Representation du placement InitialPotisionConnectedRing
Le cercle represente le rayon (scope) et on voit bien que tous les nodes sont places autour d'un centre en (0,0). Pour se faire nous avons choisi une position initial (0,0). Cette position initiale est necessaire lors de l'appelle a la fonction `getNewPositionWith` qui fait partie de la classe Position

### Stragerie 5 : NextDestinationConnectedOneMove (NextDestinationStrategy)

`getNewPositionWith`

Le but de cette strategie est de definir une nouvelle position pour un node nomme `host` tout en gardant une seule composante connexe sur le reseau. Toute division en plusieurs composantes apres deplacement d'un noeud n'est pas retenu et le noeud reste immobile jusqu'a son prochain deplacement.

Elle defini une variable qui represente le protocol du noeud `host` concerne par le deplacement appelee `pos_proto_host`

Elle defini mais n'utilise pas un set qui associe a un `ID` de node une liste de noeuds qui est la composante connexe de ce noeud nommee `initial_connected_component`. Ce set est initialise par la ligne commande :

```
1    PositionProtocol.getConnectedComponents(PositionProtocol.getPositions(
     position_pid), scope);
2
```

`getConnectedComponents` : "méthode statique renvoyant l'ensemble des composantes connexes du système, renvoie une map associant un id de la composante connexe à son ensemble de noeud"

`PositionProtocol.getPositions` : "méthodes statiques utiles pour toutes informations relative à la position des noeuds et à la topographie du système"

Cette algorithme a aussi besoin du `scope` et defini un node `current_moving` qui est le noeud precedemment mis en mouvement par l'appelle precedent de la fonction `getNewPositionWith`.

`1ER PARTIE DE L'ALGO :` Le but est de stoper le noeud host s'il est en mouvement. Si le noeud `current_moving` existe, alors il est stoppe et la fonction renvoie la valeur la position actuelle du noeud host. Si le noeud ne bouge pas alors current_moving est remis a `null`.

`2ND PARTIE DE L'ALGO :` Le noeud host n'est pas en mouvement et est autorise a bouger alors il choisi un voisin au hasard. un angle et une distance bornee par les bornes suivantes :

$$Soit \quad (distance\_min, distance\_max) \in ParametresDeBase \tag{2.43}$$

$$min\_distance = min(scope, max(distance\_min, 0)) \tag{2.44}$$

$$max\_distance = min(distance\_max, scope) \tag{2.45}$$

La nouvelle position est cree et on verifie ses bornes. On recupere localement la liste desPositions des noeuds du reseau et on verifie s'il y a une division dans le reseau apres mise a jour de la position du node (un split qui serait du au deplacement du noeud sur sa nouvelle position). Si ce n'est pas le cas il devient le noeud `current_moving` et renvoie la nouvelle position. Sinon il reste sur place.

### Strategie 6 : NextDestinationImmobility (NextDestinationStrategy)

`getNewPositionWith`

Cette strategie renvoie comme nouvelle position pour le node `host` la position actuelle du node host, il ne bouge pas.

### Strategie 7 : NextDestinationRandomPeriodicInitial (NextDestinationStrategy)

`getNewPositionWith`

Cette strategie est parametre par une periode a specifier dans le fichier de configuration appelee `random_dest_period`. C'est une strategie possedant deux etats. Soit les noeuds sont a la position initiale, soit les noeuds appliquent la strategie FullRandom avec un placement completement aleatoire. Les noeuds sont places de maniere aleatoire pendant `random_dest_period` cycles de simulation. Ensuite ils passent tous en position initiale et retourne en positionnement aleatoire lorsque tous les noeuds sont passes dans l'etat initial.

# Exercice 2 – Implémentation d'algorithmes d'élection de Leader sur un MANET

## Premier algorithme

## Question 1

*Dans la section III, expliquez pour chaque hypothèse, pourquoi elle est vérifiée (ou peut être vérifiée) dans notre simulateur.*

### 3.1.1 Node Value :

*"Each node has a value associated with it. The value of a node indicates its "desirability" as a leader of the network and can be any performance-related attribute such as the node's battery power, computational capabilities etc."*

l'interface ElectionProtocol est obligatoire pour implementer un protocol d'election. On peut voir y trouver une fonction avec le description suivante :

Listing 3.1 – `../src/ara/manet/algorithm/election/ElectionProtocol.java`

```
1  /*
2   * renvoie la valeur associ e au noeud, plus cette valeur est elev e plus le
3   * site a des chances d' tre    lu
4   */
5  public int getValue();
```

### 3.1.2 Unique and Ordered Node IDs :

*"All nodes have unique identifiers. They are used to identify participants during the election process. Node IDs are used to break ties among nodes which have the same value."*

Soit un node host, host.getID() a pour definition :

```
1     Returns the unique ID of the node. It is guaranteed that the ID is unique during
      the entire simulation, that is, there will be no different Node objects with the
      same ID in the system during one invocation of the JVM. Preferably nodes should
      implement hashCode() based on this ID.
2
```

### 3.1.3 Links :

*"Links are bidirectional and FIFO, i.e. messages are delivered in order over a link between two neighbors."* Les liens ne sont pas complètement FIFO. En effet par test nous avons pu voir que deux messages devant arriver au même temps ont un ordre non défini. Ce problème n'a pas été réglé car lors de notre tentative de rajouter un protocol FIFO (provenannt des tp d'introduction à la matière ARA) il y avait des problèmes de délivrance de messages. Cette couche n'est donc plus visibile dans notre projet et reste une amélioration possible.

### 3.1.4 Node Behavior :

*"Node mobility may result in arbitrary topology changes including network partitioning and merging. Furthermore, nodes can crash arbitrarily at any time and can come back up again at any time."* Cette propriéte correspond aux differentes strategies de noeuds que nous avons decrite. Par exemple la strategie FullRandom peut autoriser des partitionnement ou des fusions. Pour finir les crash ne sont pas modélisés dans notre système.

### 3.1.5 Node-to-Node Communications :

*"A message delivery is guaranteed only when the sender and the receiver remain connected (not partitioned) for the entire duration of message transfer."*

Dans la classe EmitterProtocolImpl on peut voir la fonction recvMsg :

Listing 3.2 – `../src/ara/manet/communication/EmitterProtocolImpl.java`

```
1     } catch (CloneNotSupportedException e) {
2     }
3
4     return emp;
5   }
6
7   /**
8    * Cette methode recupere le bon protocole et fait une delivrance de partir du
9    * host, du pid du message et du message
10   *
11   * @param host Noeud host
12   * @param msg  Message a dlivrer
13   */
14  public void deliver(Node host, Message msg) {
15
16    Protocol p = (Protocol) host.getProtocol(msg.getPid());
17    ((EDProtocol) p).processEvent(host, msg.getPid(), msg);
18
19  }
20
21  /**
22   * Cette methode gere la reception d'un message. Le message est recu sur phost
23   * si pemitter est non null et dans le scope.
```

On peut voir dans cette fonction qu'on verifie bien ligne 19 que le la distance entre les noeuds est bien inferieur ou egale a la taille du scope qui permet la communication entre deux noeuds.

### 3.1.6 Buffer Size :

*"Each node has a sufficiently large receive buffer to avoid buffer overflow at any point in its lifetime."* La taille de la mémoire en chaque noeud du graph est virtuellement infini et la seule borne est la mémoire de la machine qui simule cette taille infinie.

## Question 2

Ceci correspond à la classe : VKT04Statique.java

## Question 3

Ceci correspond à la classe : VKT04.java

## Question 4

Ceci correspond à la classe : VKT04.java

## Deuxieme algorithme

## Question 5

*L'algorithme utilise des horloges logiques. A quoi servent-elles ? Pourquoi chaque nœud ne peut incrémenter uniquement sa propre horloge ?* Ad hoc est un système réparti, alors on ne possède pas de temps globale. Voilà pourquoi on utilise des horloges logiques pour avoir des relations de causalité entre événements. Dans un algorithme donné, des horloges logiques permettent de construire un vue globale du système pour chaque noeud. A l'arrivé des messages contenants des informations sur les voisins on vérifie si on possede la derniere vision de l'état des voisins. Si ce n'est pas le cas, on le modifie. De plus, un

algorithm donné se base sur le nombre importants de broadcasts parallels et transits, donc il faut bien définir des changements parallèles et la causalité. Voilà pourquoi on utilise des vecteurs, qui garde l'état de chaque noeud.

Une horloge locale d'un noeud permet de définir des événements locaux liés avec un état de ce noeud, notamment la connexion et la déconnexion de voisin dans un contexte de cet algorithme. Supposons que chaque node possède qu'une horloge logique associée à sa connaissance du système à chaque instant. Dans ce cas là, à l'arrivé d'une nouvelle information à propos des voisins, on ne soit pas capable de définir si cette information est toujours actuelle, ou on possède déjà un vue du systèmes le plus récent. Par exemple, un noeude se connecte à une composante connexe, et ce noeud a une horloge plus grande que les autres, alors il remplacera les tables actuels de noeuds. Un autre exemple : si un noeud se connecte à un groupe de noeuds et sa propre horloge est le plus petit, le groupe apprendra jamais qu'il a un nouveau voisin.

## Question 6

*Pourquoi le knowledge est émis dans sa totalité à la détection de l'arrivée d'un nœud dans le voisinage ?* Le knowledge est émis dans sa totalité à un nouveau voisin pour le faire connaître les membres de la composante connexe auquelle il viens de connecter. Cela lui permettra à déduire qui est le leader de la composante.

## Question 7

*Quel est l'intérêt de créer des edits lors de la déconnexion d'un voisin ou de la réception d'un knowledge, au lieu d'envoyer le knowledge dans son ensemble ?* Envoie du message de type Edit lors de la déconnection d'un noeud ou de la réception d'un knowledge permet de diminuer la taille du message. De plus cela permet d'éviter des comparaisons inutiles de knowledges tables, car des noeuds d'une composante connexes possèdent déjà la même information grâce à l'échange de knowledge pendant la connexion.

## Question 8

*Quel est le contenu d'un edit ?* Message de type Edit (Source, Added, Removed, old_clock, new_clock) contient 5 tableaux :

- * l'ensemble des identifiants des noeuds "source" qui ont eu des changements
- * les identifiant des noeuds à ajouter pour chaque noeud de l'ensemble "source"
- * les identifiant des noeuds à supprimer pour chaque noeud de l'ensemble "source"
- * la dernière valeur de l'horloge logique annoncé par un noeud de "source" au système avant les modifications
- * la nouvelle valeur de l'horloge qui correspond à une horloge locale du noeud de l'ensemble "source" après les modifications

## Question 9

*Qu'implique l'adjectif reachable ligne 46 ?* Le mot "reachable" dans un contexte de l'appel d'élection du leader implique qu'on doit consulter les voisins de nos voisins etc., en parcourant knowledge table, pour analyser tous les membres de la composante connexe.

## Question 10

*Implémentez l'algorithme dans PeerSim et vérifiez qu'il fonctionne avec le moniteur graphique.*

Ceci est visible dans la classe GVLElection.java

## Question 11

*Considérons maintenant qu'il puisse y avoir des pertes de messages suite aux collisions des ondes radio (on ne vous demande pas de les implémenter).*

*Quel impact ceci aurait sur les valeurs des horloges (old_clock et knowledge[source].clock) lors des réceptions de edit ?*

Une fois on suppose que des pertes de messages peuvent avoir lieu, les modifications effectués lors de la réception du message Edit ne sont plus possible. Ca provient du fait que dans l'algorithme du base on vérifie si notre connaissance sur l'état du voisin correspond à l'état de ces dernières modifications annoncées, donc on teste une égalité des valeurs d'horloges. Si le message est perdu, l'égalité n'apparaîtra jamais et l'état du voisin reste échangeable pour ce noeud.

*Comment pourrions-nous résoudre efficacement ce problème (encore une fois , il n'est pas demandé de l'implémenter)*

La solution peut être de stocker des messages qui ont la valeur d'horloge supérieure à la valeur actuel de connaissance de noeud-host. Après, un noeud host effectue le broadcast pour demander un état complet connu par des voisins, associé à ce noeud, duquel le noeud host n'a pas reçu un message de l'horloge attendu, mais supérieure. Si la valeur de l'horloge de l'information reçue est supérieure ou égale au valeur de l'horloge du message stocké, on modifie la connaissance du noeud. Si la valeur correspond à la valeur du message perdu, on modifie la connaissance du noeud et applique des changements annoncées dans un message stocké. S'il y a la perte de messages consécutifs, on répète la demande jusqu'à ce qu'on n'a plus de messages en stockage.

# Exercice 3 – Étude expérimentale

## Étude du MANET

### Question 1

*Codez un contrôleur PeerSim qui logue au moment de son déclenchement (à l'instant t) :*

Ceci correspond à la classe Echantillon.java

*Tracez deux courbes qui prennent en abscisse la portée et en ordonnée :*

| SCOPE | AVG | VARIAN | ECARTYPE |
|---|---|---|---|
| 10 | 74,0938 | 1,0443357 | 1,021927444 |
| 15 | 72,8932 | 2,132048 | 1,460153417 |
| 20 | 71,199265 | 3,0677357 | 1,751495275 |
| 25 | 69,19755 | 7,444554 | 2,728471 |
| 30 | 66,60122 | 10,856718 | 3,294953414 |
| 35 | 63,730675 | 11,454374 | 3,384431119 |
| 40 | 60,927666 | 10,705435 | 3,271916105 |
| 45 | 57,67347 | 11,19165 | 3,345392354 |
| 50 | 54,301525 | 12,085005 | 3,476349378 |
| 55 | 50,753845 | 13,506486 | 3,675117141 |
| 60 | 47,21383 | 14,683654 | 3,831925626 |
| 65 | 43,482803 | 17,876562 | 4,228068353 |
| 70 | 39,615482 | 20,303667 | 4,505959054 |
| 75 | 36,261147 | 21,719477 | 4,660415969 |
| 80 | 32,61825 | 21,509182 | 4,637799263 |
| 85 | 29,399742 | 19,794657 | 4,449118677 |
| 90 | 26,5655 | 17,533619 | 4,187316444 |
| 95 | 23,73342 | 14,779007 | 3,844347409 |
| 100 | 21,28886 | 11,391274 | 3,375096147 |
| 105 | 19,064852 | 9,370501 | 3,061127407 |
| 110 | 16,983139 | 9,280732 | 3,046429385 |
| 115 | 14,848684 | 8,8740425 | 2,978933114 |
| 120 | 12,860526 | 7,9126935 | 2,812951031 |
| 125 | 11,394495 | 6,898599 | 2,626518418 |
| 130 | 9,967378 | 5,4308906 | 2,330427128 |
| 135 | 8,921147 | 3,4729683 | 1,863590164 |
| 140 | 7,5822945 | 2,509846 | 1,584249349 |

Figure 4.1 – Jeu de données récupérées pour tracer les courbes. On peut notamment y voir le scope qui croit de 5 en 5, AVG représente la connexité moyenne pour un scope donné et la variance ainsi que l'écart type pour ce scope donné.

#### 4.1.1 la connexité moyenne à t end

*Testez votre code, et remarquez sur le moniteur graphique l'apparition d'un lien graphique lorsque deux nœuds deviennent voisins.*

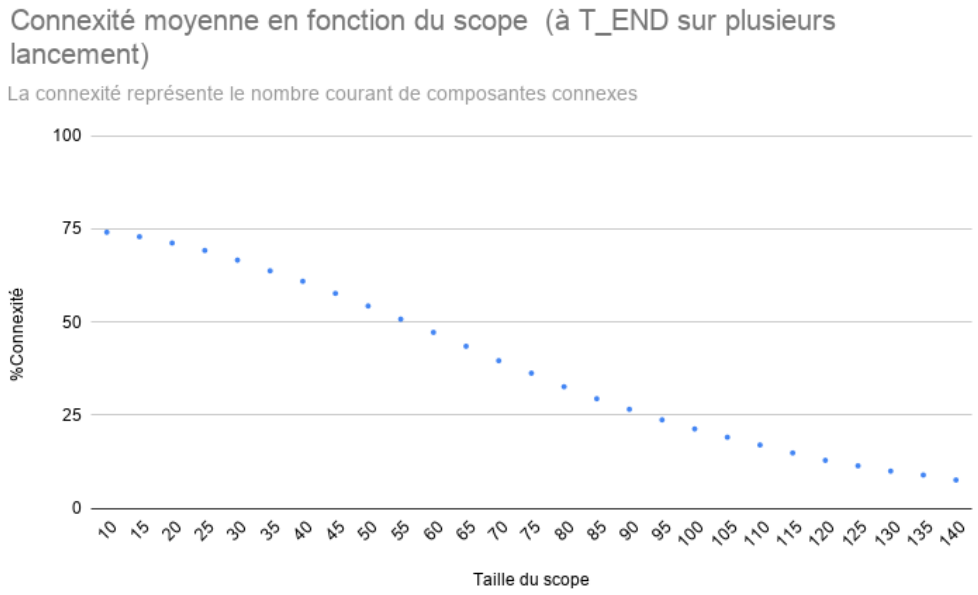


FIGURE 4.2 – Cette Courbe rerésente avec un pas de 5 sur l'axe des x la connexité moyenne. Celle ci représente le nombre moyen de composantes connexe pour un scope donné. Attention nous avons laisse 0 comme valeur minimale pour le nombre de composante connexe moyen mais c'est bien sur une valeur non atteignable dans notre simulation qui comporte 75 noeuds.

### 4.1.2 la variance (ici l'écart type) de la connexité moyenne à t end pour l'autre

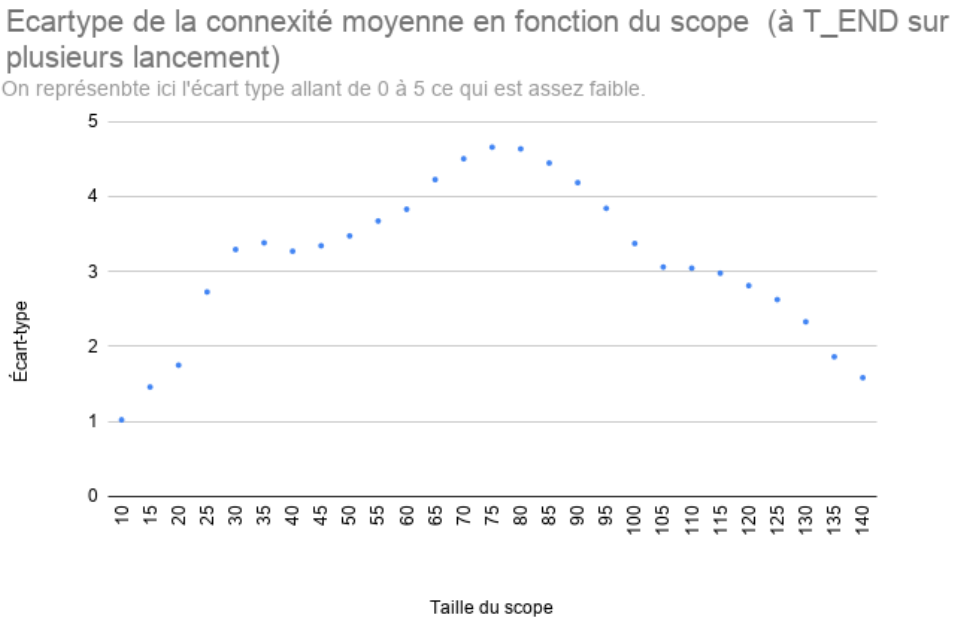


FIGURE 4.3 – On peut ici voir ici l'écart-type, qui signifie "à quel point cette connexité moyenne va varier pendant l'éxécution, l'écart type est compris entre 1 et 5 ce qui est faible pour notre jeu de donné variant entre un peu moins de 10 et à peu près 80

## Question 3

*Pourquoi est-ce pertinent de choisir la stratégie FullRandom comme stratégie de po- sitionnement pour cette expérience ? Interprétez, commentez et expliquez vos résultats.* Il est intéressant d'avoir la stratégie FullRandom d'après nos courbes. Cette stratégie nous montre qu'elle est un bon random à travers les deux courbes.

En effet, on observe, dans la première courbe on peut voir qu'elle n'est pas abrupte, les changements se font lents et sa forme en S semble signifier qu'elle est bornée (une sorte de sigmoïde à pente très douce). Lorsque le scope est petit les noeuds ont tendance à être seuls, il y a donc plus de composantes connexes, puis lorsque le scope augmente les noeuds se connectent et le nombre de composantes connexes augmente, cette courbe est donc très certainement borné par le nombre de noeuds égale au maximum de composantes connexes et 1 seule composante connexe regroupant tous les noeuds. On observe dans la seconde courbe qui malgré le fait que nos noeuds soient en grand nombre (75), le nombre de composantes connexes a tendance à varier peu. En effet, l'écart-type est faible et possède un maximum en taille de scope de 75 où y a beaucoup de connexions/déconnexions de noeuds. On peut aussi voir un sursaut un peu fort entre 25 et 25 où il semble manquer des points dans la courbe. Une amélioration aurait pu être un échantillonage plus fort pour augmenter la précision, mais cela ne change rien à la remarque générale, l'écart type est faible. On peut en déduire avec un écart type faible et une pente douce pour la connexité moyenne que la stratégie random est un bon random, en effet il a des résultats cohérents si le scope augmente le nombre de composantes connexes doit augmenter de manière régulière et ne pas avoir des comportements inattendu sinon nous ne pourrions pas faire des mesures à partir de cette stratégie.

## Question 5

*Codez un protocole décorateur d'Emitter qui s'intercalera au-dessus de l'emitter réel et en dessous du protocole d'élection.*

Ceci est trouvable dans la classe WrapperEmitter qui utilise le DP Wrapper.

## Question 6

*Implémentez un mécanisme permettant de calculer le taux d'instabilité.*

Voici le code du calcule du taux d'instabilité :

Listing 4.1 – `../src/ara/manet/Echantillon.java`

```
1  private float getTauxInst(int t) {
2
3    int N = Network.size();
4    int Err = 0;
5    int taux_inst = 0;
6    Map<Long, Position> positions = PositionProtocol.getPositions(position_pid);
7    Emitter em = (Emitter) Network.get(0).getProtocol(emitter_pid);
8    Map<Integer, Set<Node>> connected_components = PositionProtocol.getConnectedComponents(positions, em.
     getScope());
9
10   for (Map.Entry<Integer, Set<Node> > entry : connected_components.entrySet()) {
11
12     long max = -1;
13
14     for (Node n : entry.getValue()) {
15       max = Math.max(max, n.getID());
16     }
17
18     for (Node n : entry.getValue()) {
19       ElectionProtocol ep = (ElectionProtocol) n.getProtocol(election_pid);
20
21       if (ep.getIDLeader() != max) {
22         /*Err*/
23         Err = Err + t;
24         //System.out.println(t + " Node " + n.getID() + " Err ");
25       } else {
26         //System.out.println(t + " Node " + n.getID() + " Ok");
27       }
28
29       }
30     }
31   //System.out.println(" T = " + (float)Err/(N*t));
32   return (float)Err/(N*t);
33 }
```

## Question 7

*Faites une étude comparative des deux algorithmes selon les deux métriques pour plusieurs valeurs de connexité. Vous ne vous intéresserez qu'aux valeurs des deux métriques à l'instant t end de l'expérience. Des courbes commentées et justifiées sont attendues.*

L'étude ne sera portée que sur l'algorithme 2. En effet l'algorithme 1 en version dynaqiue fonctionne très bien en statique jusqu'à 450 noeuds (avant de ralentir sur nos machines) mais il existe un nombre incertains de cas qui n'ont pas pu être traiter pour pouvoir tester notre algorithme.

Pour l'instant l'algorithme 1 en dynamique est capable d'élire un leader dans sa composante connexe en 1000 cycles environ. Il propage une vague de message leader qui permet de désigner une élection meilleur que les autres et va récupérer les valeurs des ses fils sous forme de ack messages pour ensuite diffuser sur toute la composante le nouveau leader. Il existe aussi un message de type Beacon qui permet à tous dans cette même composante de savoir que le leader n'a pas disparu. Lors de la non reception d'un message de type beacon 6 fois (il existe un timer qui va compter 6 périodes de temps à définir dans le config), alors le noeud aura perdu son leader et redémarrera une élection. Cette aspect de l'algorithme est fonctionnelle est statique sur des graphes juqu'à 450 noeuds et est visible en

mettant la période entre deux beacon largement supérieur à 6 fois la période d'attente d'un message beacon. Une nouvelle éléction se déclenche à un autre endroit du graph et réussi à réélire le noeud le plus grand qui se remet à émettre des beacons.

Nous avons donc commencé la détection de nouveau voisins et la perte de voisins. Il existe deux listes pour cela. En Effet il faut finir une élection qui a commencé alors tout message leader pendant une élection doit être préserver dans une liste qui doit être potentiellement fusionné à la fin de l'élection de leader. Une seconde liste est la liste des nouveaux voisins qui sert à ne pas oublier de leur transmettre un message de leader lorsque l'éléction sera terminée.

Le problème se situe actuellement sur un état de noeud qui tente une nouvelle éléction après la perte de son leader et qui reste sans acquittement des ses voisins. Il est en cours d'éléction donc il refuse tout message de type leader et les enfiles dans sa liste de leader à fusionner.

Listing 4.2 – `../src/ara/config`

```
network.size 75
random.seed 10
simulation.endtime 50000

protocol.position PositionProtocolImpl
protocol.position.maxspeed 20
protocol.position.minspeed 5
protocol.position.width 1200
protocol.position.height 1200
protocol.position.pause 200

protocol.emit EmitterProtocolImpl
protocol.emit.latency 90
protocol.emit.scope 100
protocol.emit.variance TRUE

protocol.neighbor NeighborProtocolImpl
protocol.neighbor.periode 300
protocol.neighbor.timer 400
protocol.neighbor.listener TRUE

protocol.election GVLElection
protocol.election.periode 5000

initial_position_strategy FullRandom
initial_position_strategy.positionprotocol position
initial_position_strategy.emitter emit

next_destination_strategy FullRandom
next_destination_strategy.positionprotocol position
next_destination_strategy.emitter emit
```

FIGURE 4.4 – Configuration pour le teste ALGO2

| scope | taux instabilite | nb messages |
|---|---|---|
| 0 | 0,00% | 5625 |
| 40 | 1,33% | 21375 |
| 50 | 2,67% | 29175 |
| 60 | 2,67% | 40725 |
| 70 | 4,00% | 59325 |
| 80 | 4,00% | 77475 |
| 90 | 10,67% | 100050 |
| 100 | 13,33% | 121725 |
| 110 | 30,67% | 152250 |
| 120 | 21,33% | 202050 |
| 130 | 42,67% | 254775 |
| 140 | 12,00% | 333300 |
| 150 | 17,33% | 424725 |
| 160 | 0,00% | 508125 |
| 170 | 2,67% | 620550 |
| 190 | 0,00% | 816750 |
| 200 | 1,33% | 916725 |
| 210 | 1,33% | 1007025 |
| 220 | 0,00% | 1190250 |

FIGURE 4.5 – Jeu de données récupérées pour tracer les courbes. Ces données ont été récupéré grâce à notre script qui calcule automatiquement avec un pas sur le scope le taux d'instabilité et le nombre de message envoyé pour un lancement d'algo. Ceci représente le jeu de données de l'algorithme 2 avec un nombre de messages croissant en fontion du scope et un taux d'instabilité extrêmement variable mais qui possède un pic avec un scope de 130. On voit que l'algo est très stable avant 90 et très stable après 160.

### 4.5.1 Le taux d'instabilite en fonction du scope croissant

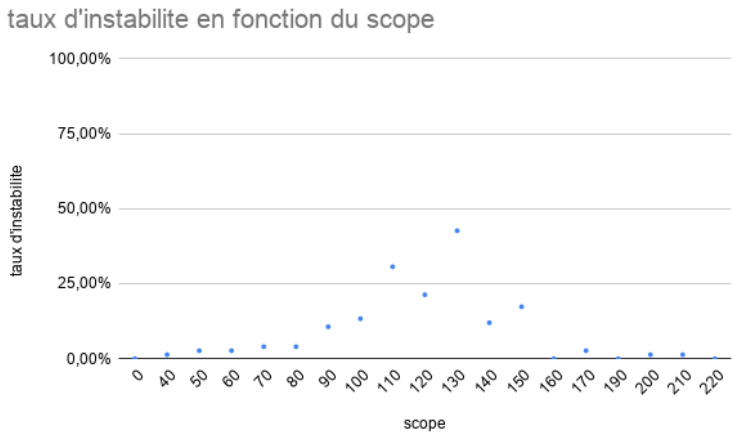


FIGURE 4.6 – Lorsque le scope est petit on ne connait personne. On est donc automatiquement son propre leader, la situation est stable. Lorsque le scope est grand tout le monde est connecté, le nombre de message de type EDIT diminue et il y a beaucoup moins d'intéractions, la situation est aussi stable. L'instabilité survient lorsque le nombre de connexion et déconnexion est maximum, en effet le choix du leader dépend pour chaque node de sa connaissance de sa composante connexe. De plus les temps de latence peuvent jouer un rôle sur l'instabilité en réduisant la précision de la construction des tables de connaissances du réseau.

### 4.5.2 Le nombre de message en fonction du scope croissant

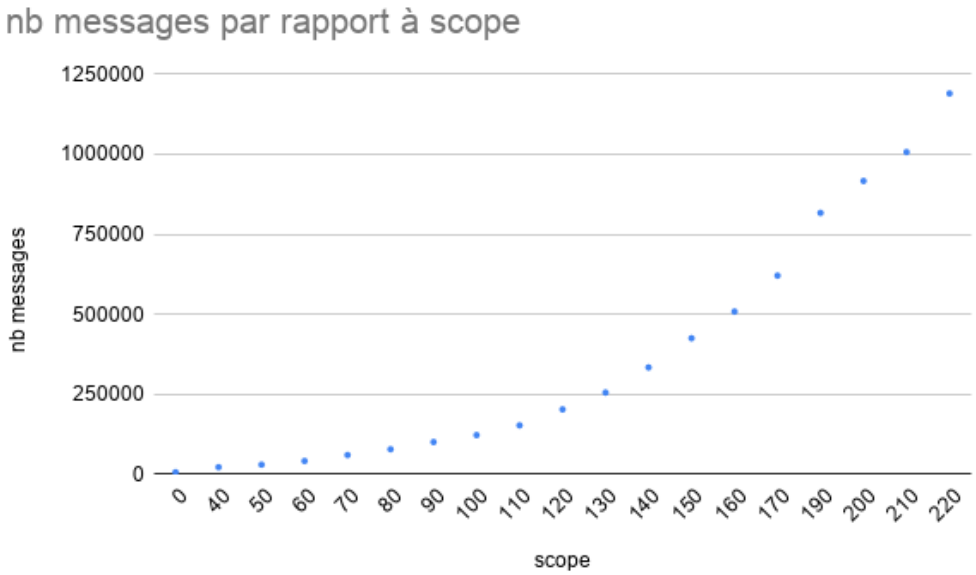


FIGURE 4.7 – Plus le scope est grand plus on connaît de monde, plus le nombre d'échanges de messages est grand. Alors pour maintenir l'information sur la connexité du système on a un grand nombre de message de type EDIT en transite.

# Conclusion

Ce projet nous a permis de mieux comprendre la difficulté à implémenter depuis un papier de recherche (qui ne donne pas de pseudo-code) un algorithme répartie dans le contexte de réseaux mobiles ou simplement avec du pseudo-code. Il nous a aussi permis d'apprendre à rester critique vis à vis des sources que nous prenons pour commencer à implémenter un algorithme. L'implémentation d'algorithme repose énormément sur le teste et l'importance d'un simulateur adapté enrichi de fonctions permettant de maîtriser chaque aspect de l'algorithme fut capitale dans notre projet.