Algorithme de création de groupes stables

Dans ce chapitre nous présentons un algorithme de création de groupes stables dans le temps au sein d'un réseau mobile ad hoc.

Dans les MANets, des événements tels que la disparition ou l'apparition de pairs et la partition du réseau, surviennent de manière fréquente.
C'est un problème quand on veut développer des applications distribuées pour réseaux mobiles :
– on ne peut pas considérer que l'ensemble des terminaux participants vont rester présents,
– on ne peut donc pas considérer la disparition d'un terminal comme un événement exceptionnel et permanent.

Dans ce chapitre nous présentons donc notre approche pour gérer la mobilité.
Nous voyons tout d'abord nos hypothèses de travail concernant la nature du réseau et le comportement des pairs le constituant. Nous motivons ensuite le choix de conception d'utiliser des informations inter-couches et présentons les éléments du protocole de la couche réseau avec laquelle notre algorithme interagit, OLSR [22].
Dans les sections suivantes, nous présentons les structures de données utilisées puis l'algorithme lui même, illustré par un exemple, avant de discuter du paramètrage de cet algorithme.
Enfin, nous détaillons notre méthode de validation avant de présenter nos résultats puis de conclure.

5.1 Hypothèses de travail

Dans cette proposition, nous faisons différentes hypothèses sur le comportement des utilisateurs, les protocoles réseaux utilisés par les terminaux, ainsi que sur les communications. Nous les présentons dans cette section.

5.1.1 Utilisateurs

Notre proposition est destinée à des utilisateurs piétons, qui collaborent via des terminaux mobiles équipés de cartes wifi et de protocoles réseaux ad hoc. Ils se déplacent en groupe et leur organisation peut être décrite par le modèle de mobilité *Reference Point Mobility Group* (RPGM) [41]. Leur vitesse varie entre 0 km/h et 4km/h.

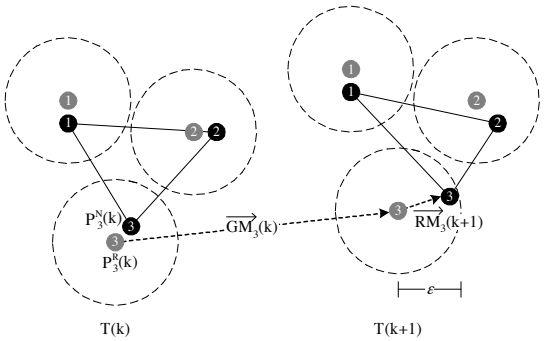

Figure 5.1 – Simulation de la mobilité des terminaux avec le modèle Reference Point Mobility Group, le RP est représenté ici en gris, extrait de [41]

Dans le modèle RPGM, les pairs se déplacent en un groupe, dont les mouvements sont décrits par un point de référence RP. Les terminaux se placent de manière à être à portée

de communication du RP, comme illustré par 5.1. Ce modèle est utilisé dans d'autres travaux proposant des algorithmes de construction de groupes mobiles, comme [43].
Par construction, la topologie du réseau est donc confinée dans un disque de 2 portées de communications de large, décrit par deux terminaux à portée maximale de communication du RP et situés à des positions diamétralement opposées . Cependant, comme il peut être plus économique d'effectuer deux communications de courte portée plutôt qu'une longue, les routes construites ne sont pas nécessairement de longueur inférieure à 2.

5.1.2 Protocoles réseaux et communications

Les terminaux des utilisateurs sont équipés de carte wifi utilisant 802.11b, la norme wifi la plus couramment utilisée. La portée de communication maximale théorique en extérieur est de 100m.
Les communications sont symétriques : à un instant, si A est à portée de communication de B, alors B est à portée de communication de A.
Nous faisons l'hypothèse d'un protocole de routage pro-actif, qui maintient les routes en l'absence de messages échangés et nous avons basé nos expérimentations sur le protocole OLSR [22]. Celui ci est présenté plus en détail dans la section suivante. Nous verrons notamment que OLSR construit des groupes de terminaux dont les communications sont symétriques.
Cette hypothèse est nécessaire car ici l'algorithme proposé utilise des informations inter-couche, issues de la couche de routage. Dans la section suivante, nous expliquons ce choix.

5.1.3 Définition d'un groupe stable

L'algorithme proposé cherche à construire des groupes de terminaux, les voisins, stables dans le temps.
Nous définissons un groupe stable comme un groupe de terminaux capables de communiquer en continu sur une période.
En d'autres termes, pour être des voisins stables à t, A et B doivent avoir été en contact depuis au moins δ_p secondes. Si par la suite ils ne peuvent pas établir le contact pendant δ_f secondes, ils arrêtent d'être voisins. On considère donc que A est capable de communiquer continuellement avec B au temps t si :
– B a reçu tous les messages diffusés par A depuis t- δ_p
– B recevra tous les messages diffusés par A depuis t et t+ δ_f
Si ces conditions sont vérifiées, B est un voisin stable de A et comme les communications sont par hypothèse symétriques, A est un voisin stable de B.
Le groupe stable de P est donc l'ensemble des pairs-voisins stables de P.

5.2 Choix de conception : information inter-couche

5.2.1 Motivation : optimisation des ressources

On préfère habituellement (e.g. pile OSI) maintenir chaque tâche différente dans un couche séparée, afin d'être modulable. Cependant l'utilisation d'information inter-couche *(cross-layering)*, permet d'optimiser l'utilisation des ressources en évitant de reconstruire plusieurs fois la même information.
Cette technique est très utilisée dans les protocoles pour réseaux mobiles ad hoc.

C'est le cas par exemple de Chapar [52] le système d'événements développé pour Transhumance [81]. Il utilise les nœuds MPRs d'OLSR, que nous présentons plus bas, comme distributeurs d'événements en se basant sur leur propriété d'accessibilité par les autres membres du réseau.
Plusieurs algorithmes de création de grappes, que nous avons présentés dans l'état de l'art, utilisent la puissance du signal reçu, une information issue de la couche physique, pour déterminer la distance à l'émetteur.
Dans nos travaux, nous proposons un protocole de niveau applicatif qui interagit avec la couche de routage OLSR pour y récupérer des informations de présence.

5.2.2 Un algorithme de routage pro-actif : OLSR

Dans un réseau, un protocole de routage permet de choisir un chemin, c'est à dire une suite de nœuds du réseau, pour acheminer un paquet entre la source d'un message et son destinataire.
Notre travail s'inscrit dans les travaux de recherche menés lors du projet ANR Transhumance, destinés à produire un intergiciel pour MANets.
La proposition d'un nouvel algorithme de routage ne relevant pas du projet Transhumance, les membres du projet ont été amenés à choisir un algorithme de routage, le protocole et son implantation se devant d'être complètement validés.
Bien que de nombreux algorithmes aient été proposés dans la littérature, la plupart n'offrent pas d'implantation et sont validés par simulation uniquement. Ceux qui ont été implantés l'ont souvent été à des fins de prototypages, et ne sont pas utilisables. Seuls les protocoles DSR [49], AODV [83] et OLSR [22] bénéficient de véritables implantations, et le choix de l'équipe Transhumance s'est porté sur OLSR du fait de sa solide implantation OLSR-Unik [65], et de sa communauté active, notamment sur la liste IETF-manet.
OLSR est un protocole de routage pro-actif à état de lien proposé par le projet HIPERCOM-INRIA en 2001. Dans un protocole à état de lien, comme OSPF [76] - qu'on oppose aux protocoles à vecteurs de distances, comme RIP [40] et aux protocoles à vecteurs de chemin comme BGP [64] - tous les terminaux inondent le réseau avec la liste de leurs voisins, afin que chacun puisse reconstituer le graphe de routage.
Le terme pro-actif signifie que les routes sont maintenues, par opposition aux protocoles réactifs, tel que AODV, où les routes sont construites à la demande. Il existe d'autres types de protocoles, mais ces deux classes sont les plus génériques. Les deux approches ont leurs avantages et leurs inconvénients, et [46] montre que pour un trafic sporadique, les algorithmes pro-actifs sont plus adaptés.
Les travaux dans lesquels s'inscrit notre algorithme visent au support d'applications collaboratives sur MANet, et le trafic, étant généré par des utilisateurs humains, est donc sporadique.

5.2.2.1 Structures de données d'OLSR

OLSR maintient sur chaque terminal une table de routage construite, comme nous le verrons dans la section suivante, grâce aux messages TC. Ces tables contiennent une liste de tuples, chacun contenant les informations suivantes :
– adresse destination R_dest_addr.
– prochaine adresse R_next_addr : adresse à laquelle il faut envoyer le message pour qu'il soit routé jusqu'à R_dest_addr.

– nombre de sauts *R_ dist* : la distance en terme d'arcs dans le graphe de routage entre le terminal et *R_ dest_ addr*.
– adresse de l'interface locale *R_ iface_ addr*.

Par ailleurs, grâce aux messages HELLO, OLSR construit et maintient, comme nous le verrons dans la section suivante :

– l'ensemble N de ses voisins à 1 saut,
– l'ensemble *N2* de ses voisins à 2 sauts,
– l'ensemble *MPR Selector* qui contient la liste des terminaux ayant sélectionné le terminal local comme MPR

Par la suite, dans nos travaux nous utilisons des informations issues des tables de routages.

5.2.2.2 Fonctionnement d'OLSR

OLSR établit tout d'abord la topologie du réseau. Tout d'abord, l'ensemble des communications possibles entre terminaux, est calculé. A partir de cet ensemble, OLSR établit ensuite le plus court chemin entre chaque paire de nœuds. Ceci est fait sur chaque terminal et les terminaux doivent divulguer la liste de leurs voisins à portée de communication à l'ensemble du réseau.

L'idée derrière OLSR est de limiter le nombre de terminaux envoyant des messages d'inondation, en élisant des pairs relais, nommés MPR *(MultiPoint Relays)* : quand un pair veut communiquer, en diffusion ou en point à point, il doit envoyer son message à travers un des MPR, seuls pairs autorisés à faire suivre des messages dans le réseau. Un MPR est par construction un voisin à 1 saut, comme nous allons le voir par la suite.

Quatre types de messages sont utilisés : HELLO, TC (Topology Control), MID (Multiple Interface Declaration) et HNA (Host and Network Association). Nous allons présenter brièvement le fonctionnement d'OLSR et les messages MID et HNA ne seront donc pas détaillés.

Les messages HELLO servent tout d'abord à détecter le voisinage d'un pair :

– périodiquement, chaque pair P diffuse à un saut un message de HELLO, avec la liste L de ses voisins P_i et en indiquant si le lien est symétrique (P_i reçoit les messages de P) ou asymétrique (à la connaissance de P, P_i ne reçoit pas ses messages). Cette liste est au départ vide.
– quand un pair P_1 reçoit un message de HELLO de P_2, il enregistre P_2, comme un voisin, asymétrique.
– quand un pair P_1 reçoit un message de HELLO de P_2 avec sa liste de voisin L_2, et que P_1 se trouve dans L_2, il enregistre P_2, comme un voisin symétrique, puisque P_2 voit les messages de P_1.

Un message HELLO émis par P contient donc entre autres informations :

– sa liste de voisins, indiquant pour chacun :
  – si la connexion est symétrique, asymétrique, ou si elle a disparu.
  – si le voisin a été sélectionné comme MPR pour le terminal.
– sa disponibilité à agir lui-même en tant que MPR, avec 5 degrés de disponibilité : *Never, Low, Default, High, Always*.

Ces messages HELLO sont ensuite utilisés pour élire les MPR. Suite à l'échange de HELLO, chaque terminal connaît l'ensemble N de ses voisins symétriques à 1 saut et l'ensemble *N2* de ses voisins symétriques à 2 sauts. Il applique ensuite l'heuristique suivante afin de sélectionner la liste des MPR :

1. Initialiser l'ensemble MPR avec les membres de N (voisins à 1 saut) acceptant de jouer le rôle de MPR avec une disponibilité *Always*.
2. Pour chaque terminal y dans N, calculer le degré $D(y)$, c'est à dire le nombre de ses voisins à un saut symétriques :
   – qui ne sont pas voisins à 1 saut du terminal local (donc pas dans N : en calculant 2 sauts, on n'emprunte pas deux fois le même lien).
   – qui sont donc par construction à deux sauts du terminal local.
3. Pour tout terminal dans *N2*, si un seul terminal t dans N permet la communication, t est ajouté à l'ensemble MPR, quelle que soit sa disponibilité.
4. Tant qu'il existe un terminal dans *N2* auquel on ne peut pas accéder via un terminal dans l'ensemble MPR :
   (a) Pour chaque terminal t de N, pas encore MPR, on calcule son accessibilité, c'est à dire le nombre de terminaux dans *N2*, non accessibles via un MPR existant, auxquels on pourrait accéder si t devenait MPR.
   (b) On sélectionne comme MPR le terminal avec la plus haute disponibilité, et qui a une accessibilité non nulle. En cas de possibilités multiples, on sélectionne le terminal qui présente la plus forte accessibilité. S'il reste encore des possibilités multiples, on choisit le terminal qui possède le degré D le plus élevé.
5. Une fois construit l'ensemble MPR, une optimisation possible est de classer les MPR par disponibilité ascendante et de vérifier tour à tour si on peut retirer un MPR tout en continuant à joindre tout N2. Le cas échéant, ce MPR peut être retiré de l'ensemble MPR.

Cet algorithme est une heuristique permettant de construire l'ensemble MPR, et d'autres heuristiques ont été proposées pour améliorer OLSR [31].

Les messages TC sont utilisés pour construire la topologie du réseau. Un message TC contient la liste des liens du terminal, et un numéro de séquence afin de pouvoir discriminer entre deux messages lequel est le plus récent (l'autre n'étant donc pas pris en compte). Ces messages sont envoyés périodiquement, avec par défaut une période de 5s. Ils sont diffusés via les MPR.
A partir des messages TC, chaque terminal peut reconstruire la topologie du réseau. Il applique ensuite un algorithme de type Dijkstra ou Bellman-Ford pour construire le plus court chemin entre lui et chacun des terminaux, et peuple ainsi les tables de routages.

5.3 Structures de données de l'algorithme proposé

Pour exécuter cet algorithme chaque terminal T maintient deux structures de données résumant la connaissance qu'on a des autres pairs :
– *stableNeighbourhood* qui contient une liste d'adresses IP représentant les pairs appartenant au groupe stable autour de T. C'est cette structure qui représente l'information utilisée par les applications des couches supérieures.
– *heardOf* qui associe à une adresse IP un compteur de présence. Elle contient la liste des pairs dont on a récemment entendu parler.
La section suivante présente la manière dont ces structures sont gérées.

Nous faisons usage des tables de routage OLSR pour déterminer la liste des destinations possibles, *Liste_destinations*.
Les tables de routages sont lues depuis la couche de routage périodiquement. L'information est récupérée sous la forme d'un ensemble de paires (IP, distance), retourné par la fonction *getRoutingTableFromRoutingLayer*.

5.4 Algorithme

La figure 5.2 présente le code de notre algorithme en python.
Rappelons les opérateurs ensemblistes en python :
– A - B : éléments appartenant à l'ensemble A et n'appartenant pas à l'ensemble B
– A & B : intersection des ensembles A et B
Cet algorithme est exécuté périodiquement et se comporte de la manière suivante, en deux phases :

1. Phase d'observation : quand un pair P arrive en vue, on lui associe un compteur, nommé *Presence Counter*, ou PC, initialisé à 1 et stocké dans *heardOf*. Chaque période on vérifie si P est présent. Si c'est le cas, PC est incrémenté (ligne 28), et quand PC dépasse le seuil de stabilité *stableThreshold*, il devient membre du groupe stable (ligne 34). Sinon, *PC* est décrémenté (ligne 25), et quand il atteint 0, on arrête d'observer P (ligne 23).
   Pour devenir un voisin stable, P doit donc être présent pendant au moins *stableThreshold* périodes + le nombre d'absences depuis le début de l'observation.
2. Quand le compteur de présence d'un pair a atteint *stableThreshold*, il devient un voisin stable, et on entre donc dans la phase de maintenance
3. Phase de maintenance : lors de cette phase, on continue d'observer P à chaque période et de modifier PC en fonction de sa présence ou de son absence. *PC* peut être incrémenté jusqu'à un second seuil, *stableThreshold + maxAbs*, suite à quoi il est plafonné à cette valeur. Si *PC* passe en dessous du premier seuil de stabilité *stableThreshold*, il est sorti du groupe stable.
   On tolère donc un certain nombre d'absences sporadiques consécutives de la part du pair avant qu'il soit retiré du groupe stable.

5.5 Exemple

Nous présentons ici un exemple de fonctionnement de l'algorithme proposé, illustré par la figure 5.3. Ici, le seuil de stabilité est fixé à 5, et le nombre maximum d'absences tolérées à 3. L'algorithme est exécuté sur le terminal A, qui examine le comportement du terminal P. Le compteur de présence associé par A à P est appelé PC.
La première ligne représente le temps en nombre de périodes de l'algorithme, la seconde ligne indique si à cette période le pair est en contact, et la troisième ligne est l'évolution de PC.

1. Avant t=0, P n'est pas connu de A.
2. De t=0 à t=2, PC est incrémenté.
3. À t=3, P est absent, donc son compteur de présence PC est decrémenté.
4. De t=4 à t=6, PC est incrémenté ; à t=6, PC atteint la valeur *stableThreshold* et P devient un voisin stable de A.

```
PERIOD_SEC=2
stableThreshold = 120
maxAbs = 60

class Peer :
    def __init__(self, group, filename):
        self.group = group
        self.heardOf =dict()
        self.stableNeighbourhood=[]

def updateHeardOf(self, routes) :
    heardOfSet= set(self.heardOf.keys())
    routesSet= set(routes.keys())
    absent = heardOfSet-routesSet
    stillpresent= routesSet & heardOfSet
    newcomers = routesSet-heardOfSet

    for p in newcomers :
        self.heardOf[p]= 1

    for p in absent :
        if self.heardOf[p] == 1 :
            del self.heardOf[p]
        else:
           self.heardOf[p]= self.heardOf[p] - 1

    for p in stillpresent :
        if self.heardOf[p]<stableThreshold+maxAbs :
            self.heardOf[p]=self.heardOf[p]+1

def buildStableGroup(self):
    self.stableNeighbourhood=[]
    for p in self.heardOf.keys():
        if self.heardOf[p]>stableThreshold :
            self.stableNeighbourhood.append(p)

def buildStableGroup(self) :
    while true:
        routes = getRoutesFromRoutingLayer()
        self.updateHeardOf(routes)
        self.buildStableGroup()
        sleep(PERIOD)

```

Figure 5.2 – Algorithme de construction de groupe stable

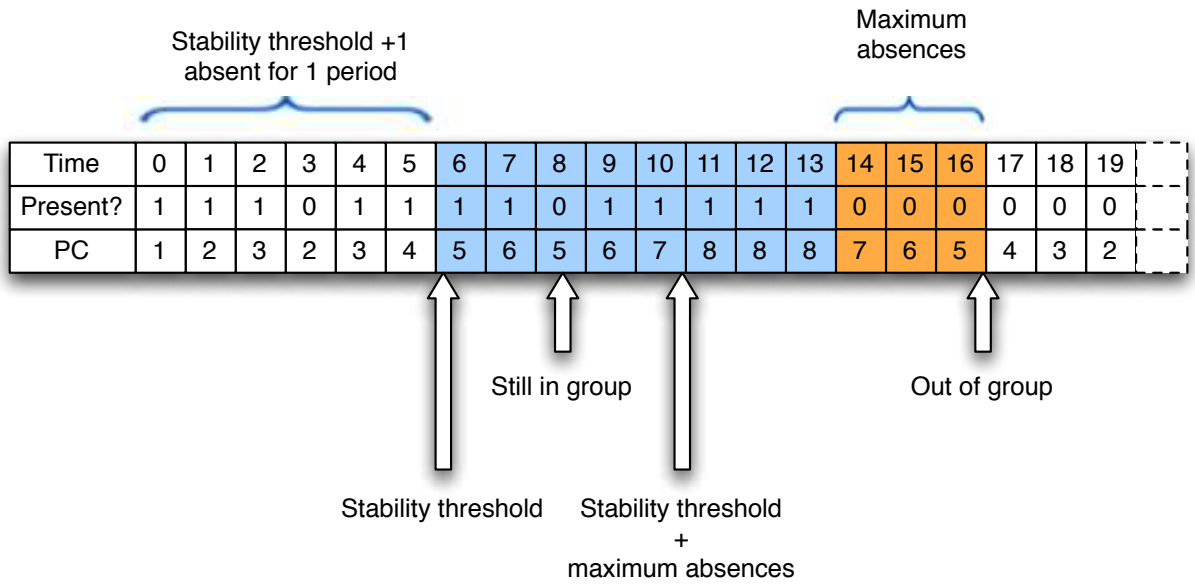

FIGURE 5.3 – 1 group

5. De t=6 à t=7, PC est incrémenté.
6. À t=8, P n'est pas accessible, PC est donc decrémenté ; P reste dans le groupe : une absence sporadique est tolérée.
7. De t=9 à t= 13, PC est incrémenté jusqu'à atteindre la valeur *stableThreshold+maxAbs* ; il se stabilise.
8. À t=15, P disparaît ; PC est décrémenté, mais P reste dans le groupe.
9. de t=14 à t=16, PC est décrémenté.
10. À t=18, PC passe en dessous de *stableThreshold* : le nombre maximum d'absences *maxAbs* tolérées est dépassé ; P est donc sorti du groupe.

5.6 Paramètres de l'algorithme

On peut voir que différents paramètres influencent le comportement de l'algorithme proposé :
– le taux de rafraîchissement, représenté par la période *PERIOD* (ligne 42),
– le seuil de stabilité que le pair doit atteindre pour être considéré comme un voisin stable, représenté par *stableThreshold* (ligne 28 et ligne 34),
– le nombre maximum d'absences tolérées avant qu'un pair ne soit sorti du groupe stable, représenté par *maxAbs* (ligne 28).
Dans cette section, nous présentons la signification physique de chaque paramètre et nous voyons comment choisir leurs valeurs.

5.6.1 Période de rafraîchissement du voisinage

L' algorithme proposé est périodique et cherche à prédire quels pairs seront présents à la période suivante, en se basant sur les périodes passées. On veut donc connaître la probabilité

que le statut d'un pair change entre deux échantillonnages.
La période d'envoi de messages HELLO permettant de rafraîchir les tables de routage est, par défaut, de 2 secondes. Notre algorithme se basant sur le contenu des tables de routage pour déterminer qui se trouve à portée, une période inférieure à 2s est donc inutile.
Nous avons donc fixé *la période de rafraîchissement* du voisinage à **PERIOD=2s**.
Soit un pair A du groupe stable au temps t, moment de l'échantillonnage. Nous voulons évaluer la probabilité que ce pair reste joignable durant l'intervalle [t ; t+PERIOD].
Voyons tout d'abord ce qui se passe pour un terminal à portée de communication, comme illustré par la figure 5.4. On considère une portée de communication R comprise entre $r_{min} = 30m$ et $r_{max} = 100m$. La vitesse du pair est v=1m/s. Dans le pire des cas, si deux pairs se déplacent dans des directions opposées, la distance entre eux va donc grandir de 4m.
Si un pair est à portée de communication, deux cas sont possibles :
– le cas A : il est à une distance inférieure à R-4, auquel cas il ne perdra pas contact
– le cas B : il est à une distance supérieure à R-4, et se trouve donc dans la zone bleue de la figure 5.4. Dans ce cas, selon son déplacement, il risque de se retrouver dans la zone orange où la communication sera perdue.

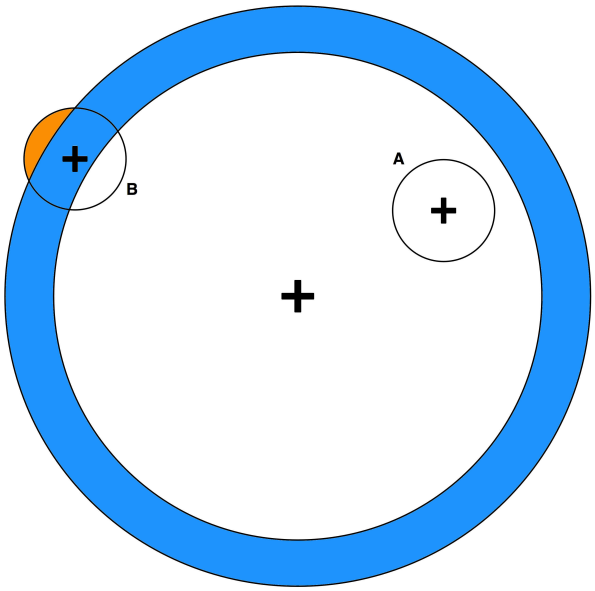

Figure 5.4 – Période de rafraîchissement du voisinage

Nous avons évalué la probabilité de garder le contact par la méthode de Monte Carlo [92]. Pour chaque portée R de communication entre 10m et 100m (pas de 10m), on effectue 10^9 fois cette expérience :

1. Soit un repère orthogonal, avec le terminal C positionné à l'origine.
2. On place aléatoirement un autre pair D à une position P, telle que la distribution des positions sur le disque de rayon R et de centre (0,0) soit uniforme. D est donc à portée de communication de C. Pour cela, on génère aléatoirement des coordonnées polaires, avec les lois de probabilité suivantes :
   – l'azimut ρ suit une distribution uniforme continue sur l'intervalle [0 ;2π]

- la coordonnée radiale θ suit une distribution de loi $R * \sqrt{(X)}$, avec X une v.a. suivant une loi uniforme continue sur l'intervalle [0 ;1].

3. Si θ est inférieur à R-4, quels que soient les déplacements effectués par C et D, ils resteront à portée de communication.
4. Sinon, on génère le déplacement de D pendant le temps PERIOD. Rappelons qu'on travaille dans un référentiel de centre C, et que D peut donc se déplacer d'au plus 4m :
   (a) comme à l'étape 2, on génère une coordonnée polaire distribuée uniformément dans le disque de centre P et de rayon 4.
   (b) on calcule la distance [CD] suite au déplacement : si celle-ci est supérieure à R, le contact est perdu.

Pour chaque valeur de R, nous avons comptabilisé le nombre de fois où le terminal reste à portée, puis avons calculé la probabilité recherchée. Les résultats sont présentés dans le tableau 5.5.

Portée, en m	10	20	30	40	50	60	70	80	90	100
Probabilité, en%	92.88	96.64	97.80	98.36	98.69	98.91	99.07	99.19	99.28	99.35

FIGURE 5.5 – Probabilité de contact au temps T+1, selon la portée de communication, pair à portée de communication

Cette probabilité concerne les terminaux à portée de communication. Nous faisons l'hypothèse que les terminaux se déplacent suivant le modèle RPGM, et nous avons vu que, dans ce cas, le réseau fait 2 portées de communication de large. Les routes peuvent cependant être plus ou moins longues selon la densité du réseau.
Dans le pire cas, deux terminaux A et C ne peuvent communiquer qu'à travers B. La probabilité que A et C gardent contact est donc la probabilité que A et B restent en contact et que B et C restent en contact. On a donc résumé dans le tableau 5.6 les probabilités de perdre le contact avec un membre du groupe.

5.6.2 Limite de stabilité

Le seuil de stabilité *stability threshold* indique pendant combien de périodes de l'algorithme un pair doit être vu avant de pouvoir être considéré comme faisant parti du voisinage stable. C'est donc une approximation de δ_p, le nombre de secondes pendant lesquelles deux pairs doivent être capables de communiquer avant de devenir voisins stables.
Dans notre algorithme, deux pairs deviennent voisins après *stableThreshold + le nombre de périodes durant laquelle la communication était perdue**PERIOD secondes. La durée entre le premier contact et le moment où deux pairs deviennent voisins stables est donc supérieur ou égal à *stableThreshold**PERIOD.
L'algorithme proposé doit différencier deux groupes qui se croisent de deux groupes qui fusionnent. Le seuil *stabilityThreshold* doit donc être suffisamment élevé pour ce faire, mais

Portée, en m	10	20	30	40	50	60	70	80	90	100
Probabilité, en %	86.26	93.39	95.64	96.74	97.39	97.83	98.14	98.38	98.56	98.70

FIGURE 5.6 – Probabilité de contact au temps T+1, selon la portée de communication, cas général

assez bas pour que la collaboration puisse commencer le plus rapidement possible en cas de fusion.
Le temps de contact entre deux groupes mobiles varie en fonction de l'angle α de leurs trajectoires. Dans le meilleur des cas, les deux groupes vont dans des directions opposées. Dans le pire des cas, ils empruntent des trajectoires quasi parallèles et reste donc en contact longtemps.
On veut pouvoir distinguer deux groupes se croisant avec des trajectoires orthogonales ($\frac{\pi}{2} rad$) ou un angle plus grand.
Comme nous supposons un modèle de mobilité de type RPGM, et une portée de communication maximale R= 100m, le diamètre de l'aire maximale de couverture d'un groupe est donc de 4R = 400m.

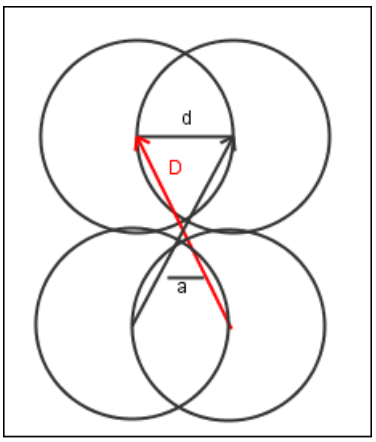

Figure 5.7 – Deux groupes se croisent à angle a

Nous devons donc calculer le temps T durant lesquels deux groupes, dont les trajectoires, qu'on assimile à des droites, se croisent avec un angle α restent en contact, connaissant d la couverture de communication maximale d'un groupe. Dans le schéma 5.7, on peut voir que T est le temps nécessaire pour parcourir la distance D. Connaissant la vitesse v du groupe, calculer D nous permet donc de trouver T.
On a donc :

$$\frac{0.5 * d}{0.5 * D} = sin(0,5 * \alpha) \implies D = \frac{d}{sin(0,5 * \alpha)}$$

Comme v=1m/s, on a :

$$T = \frac{D}{v} = \frac{d}{sin(0,5 * \alpha)}$$

On veut que les groupes restent distincts pour un angle $\alpha = \frac{\pi}{2}$. Dans ce cas :

$$T = \frac{200}{sin(\frac{\pi}{4})} \approx 235$$

Comme PERIOD=2s, nous avons fixé *stableThreshold* à 140 : un pair devient un voisin stable s'il est vu pendant au moins 4 minutes 40s.
Notre algorithme peut alors distinguer s'il y a fusion ou s'il s'agit de deux groupes se croisant avec un angle $\alpha \geq \frac{\pi}{2}$. Pour un angle α des trajectoires plus faible, il existe donc

Angle de croisement (en degré)	80	70	60	50
Temps (en minutes)	1mn2s	2mn17s	4mn40s	5mn53s
Angle de croisement (en degré)	40	30	20	10
Temps (en minutes)	7mn44s	10mn52s	17mn11 s	36mn14s

FIGURE 5.8 – Temps nécessaire à l'algorithme pour corriger une détection de groupe fallacieuse

une période *error* durant laquelle l'algorithme détectera de manière erronée un seul groupe au lieu de deux. Nous avons donc calculé *error* pour différentes valeurs de α.
A t_{deb}, supposons deux groupes qui entrent en contact, et le compteur de présence croît. On suppose qu'il n'y a pas d'absence sporadique. A $t_{grp} = 280$, les deux groupes ont fusionnés. A t_{end} =T, PC décroît. Dans ce cas :
- Soit T > 2*(stableThreshold+maxAbs)
  - PC = stableThreshold+maxAbs
  - l'erreur est corrigée au bout de 2*maxAbs secondes
  - l'algorithme était donc incorrect pour :
    $error = t_{end} + 2maxAbs - t_{grps} = T + 2 * maxAbs - 280$
- Sinon
  - $PC = \frac{T}{2}$
  - l'erreur est corrigée au bout de $2 * (\frac{T}{2} - stableThreshold)$
  - l'algorithme était donc incorrect pour :
    $error = t_{end} + 2 * (\frac{T}{2} - stableThreshold) - t_{grp} = 2 * T - 560$

La table 5.8 présente quelques résultats en fonction de l'angle selon lequel les deux groupes se croisent.
Comme on peut le voir, plus l'angle de croisement est grand, plus la durée de l'erreur est élevée. Un observateur humain ferait cependant sans doute la même erreur.
Notons que le temps de correction est en fait lié à deux paramètres : le temps d'attente avant de construire un groupe *stability threshold*, fixé ici à 280s, et le temps d'attente avant de sortir quelqu'un du groupe, fixé ici par *maxAbs* à 2 minutes. La raison pour fixer ce paramètre est présentée dans la section suivante.

5.6.3 Tolérance aux absences transitoires

Le paramètre *maxAbs* représente le nombre maximal de périodes durant lesquelles on tolère qu'un voisin stable soit absent avant de le sortir du voisinage stable.
Dans la définition du groupe stable donnée en 5.1.3, *maxAbs* est donc une approximation de δ_f, la durée durant laquelle deux pairs stables sont supposés être capables de communiquer. Quand deux voisins stables perdent contact, leur compteur de présence PC est compris entre *stableThreshold* et *stableThreshold+maxAbs*.
Ce paramètre a donc deux usages :
- Quand un voisin stable disparaît, on veut le sortir du groupe stable le plus rapidement possible. Il en va de même quand un groupe se sépare en deux.
- Quand un voisin stable est absent pour peu de temps, on ne peut pas être certain qu'il a vraiment disparu. On veut donc que notre algorithme tolère ces absences sporadiques et garde le voisin dans le groupe stable.

La fréquence de départ des pairs n'a pas d'influence sur *maxAbs*, à ceci près que *maxAbs* doit être assez bas pour permettre de les sortir rapidement du groupe.

Par contre, si on connaissait la fréquence et la durée des absences temporaires dans le pire cas, on pourrait fixer *maxAbs* de manière à les supporter. Ces absences peuvent avoir des causes diverses : un obstacle temporaire entre deux terminaux , comme par exemple un mur, ou une surcharge temporaire du réseau qui crée une perte de paquets
Comme nous n'avons pas de contrôle sur ces situations, ni de possibilité de les prédire, nous avons choisi *maxAbs* en nous basant sur la définition du groupe : si un pair faisant parti de notre groupe stable est absent pendant plus de 2 minutes, il doit être sorti du groupe.
On a donc : $maxAbs = \frac{60*2}{PERIOD} = 60$.

5.7 Critères d'évaluation classiques : Messages, Calcul, Passage à l'échelle

Cet algorithme a été tout d'abord évalué en terme de propriétés calculables a priori, à savoir le nombre de messages échangés et la complexité du calcul, et ce que l'on peut en déduire sur sa capacité à passer à l'échelle. L'algorithme proposé est avantageux sur ce point car du fait de l'utilisation d'informations inter-couches, aucun message n'est échangé pour la mise à jour du voisinage, et on ne fait donc pas de calcul complexe.
Tout d'abord, notre méthode ne nécessite *aucun échange* de message entre les pairs car elle profite des informations contenues dans les tables de routage. Bien que nous nous soyons ici basés sur OLSR, cet algorithme peut s'appuyer sur n'importe algorithme de routage proactif. Il peut aussi être exécuté au dessus d'un algorithme réactif, mais ne construira qu'une vue partielle du voisinage.
Par ailleurs, le rafraîchissement du voisinage est calculé toutes les 2 secondes, qui est la période d'envoi des messages HELLO d'OLSR. Ceci est fait en calculant les différences entre les listes *hearfOf*, et *table de routage*. Si les deux listes, de taille M et N sont triées, la complexité de cet algorithme est max(M,N). On travaille, par ailleurs, dans un réseau d'une centaine de terminaux, ce qui borne le temps de calcul.
Cette technique passe donc à l'échelle dans la même mesure que le protocole de routage sur lequel elle s'appuie.
Sur ces critères, notre algorithme est satisfaisant. Cependant, cette évaluation n'est pas suffisante. On propose un algorithme prédictif. Afin d'en montrer la validité, on doit vérifier que ses prédictions sont correctes.

5.8 Simulation

La validation de protocoles et d'applications pour MANets par des expériences réelles est complexe à mettre en œuvre. En effet, la mise en oeuvre de MANets pour la validation est problématique pour plusieurs raisons :
- les terminaux étant mobiles, on a besoin d'un opérateur, humain ou robot, par terminal, capable de reproduire des schémas de mobilité et des actions.
- les expérimentations sont difficilement reproductibles. Les communications étant radio, elles peuvent être perturbées par des signaux extérieurs sur lesquels l'expérimentateur n'a pas de contrôle.

Il existe des environnements de validation pour applications mobiles, comme par exemple Emulab [107], mais ils sont rares et peu accessibles, ce qui ne permet pas de reproduire plusieurs fois nos expériences.

Valider notre algorithme sur un MANet déployé nécessiterait donc un groupe d'opérateurs mobiles et de terminaux, ainsi qu'une salle isolée. N'ayant pas de telles ressources, nous avons donc validé notre algorithme par simulation sur le simulateur NS-3 [105].

5.8.1 NS-3

NS-3 (Network Simulator 3) est un simulateur de réseaux à événements discrets. Il se veut successeur de NS-2, et a été développé initialement par la National Science Foundation, et l'INRIA. C'est un projet open source sous licence GNU GPLv2, et tout le monde peut y contribuer.
NS-3 est implémenté en C++ pour le moteur de simulation et les modules (applications, protocoles et modèles physique), avec une sur-couche python pour créer des simulations.
L'implémentation de ce type de simulateur se décompose en un moteur de simulation et des modules. Le moteur de simulation comprend une horloge logique et une liste d'événements triés par date logique. Quand on crée une simulation, on la configure avec des modules, comme par exemple des protocoles réseaux, ou des générateurs de trafic, qui vont réagir à certains événements et en générer de nouveaux. Quand tous les événements à une date sont exécutés, l'horloge logique est avancée, ce qui active de nouveaux événements.
NS-3 est open source et gratuit. Le projet étant récent, NS-3 est plus simple à prendre en main que son ancêtre NS-2. Par ailleurs, le protocole OLSR est implanté dans NS-3, ainsi que la norme WIFI 802.11b avec différents modèles physiques, et les protocoles internet.
Nous avons donc pu nous concentrer sur le développement des modules spécifiques à l'algorithme proposé.

5.8.2 Méthodologie

Nous avons effectué une série d'expérimentations avec NS-3 pour valider l'algorithme de création de groupes proposé.
Chaque expérimentation est paramètrée par les valeurs suivantes :
- la taille de l'aire simulée.
- le nombre de groupes mobiles.
- le nombre de terminaux par groupe.
- le type d'événement de mobilité qu'on souhaite générer (fusion, séparation, disparition), sachant que le modèle de mobilité est RPGM.
- la volatilité des terminaux.

Le processus de simulation se déroule ensuite, comme illustré par la figure 5.9, ainsi :

1. Un script python génère :
   - des traces de mobilité (liste de coordonnées cartésiennes dans l'aire) pour chaque terminal.
   - des traces de volatilité pour chaque terminal. La période entre deux absences et le temps d'absence sont générés aléatoirement et suivent des lois normales gaussiennes.
2. NS-3 est ensuite configuré avec les modules suivants :
   - couche physique wifi et liaison de donnée 802.11b.
   - couche réseau IPV4 et OLSR.
   - *ReadFromTracesApplication* : modèle de mobilité que nous avons développé, lisant les traces générées à l'étape 1.
   - deux applications, que nous avons développées pour cette simulation :

- *OffOnDeviceApplication* : éteint et allume la carte wifi pour simuler des *erreurs transitoires*, en fonction des valeurs générées auparavant.
- *DumpRoutingTableApplication* : récupère le contenu des tables de routage depuis le module OLSR toutes les deux secondes, les transforme sous la forme d'une liste de paire IP/Distance, et à la fin de la simulation, les sauvegarde sur disque.

3. Un script python utilise ces routes pour exécuter ensuite plusieurs fois l'algorithme avec différentes valeurs pour les paramètres *maxAbs* et *stableThreshold*.

Découpler ces différentes étapes du simulateur nous permet d'optimiser le temps de validation :
- Générer la mobilité en dehors de NS-3 par un script python et n'utiliser qu'un lecteur de traces dans NS3 est beaucoup plus flexible et rapide que développer plusieurs modules NS-3.
- Découpler l'exécution de l'algorithme de la simulation physique est beaucoup plus rapide : en effet, on réutilise les même schémas de mobilité pour examiner le comportement de l'algorithme pour différentes valeurs des paramètres, sans avoir à simuler plusieurs fois les couches basses.
- On peut ainsi utiliser directement le code de l'algorithme, sans avoir à le réimplémenter dans NS-3.

5.8.3 Critère d'évaluation : une métrique de précision

Nous avons vu dans la section précédente comment l'algorithme proposé s'évaluait selon les critères de complexité classique.
Cependant, l'algorithme proposé étant de nature prédictive, nous voulons pouvoir l'évaluer en mesurant l'acuité de ses prédictions en supposant l'existence d'un oracle qui puisse déterminer des groupes idéaux en se basant sur la connaissance de l'ensemble des positions et des mouvements des terminaux.
Pour cela, nous allons simuler des terminaux mobiles se déplaçant en groupes, suivant le modèle RPGM. L'oracle connaît ces groupes simulés, et idéalement, ceux-ci devraient être détectés par notre algorithme.
Pour déterminer la précision de l'algorithme proposé, nous comparons donc la composition du groupe idéal *ideal*, ici le groupe simulé, au groupe déterminé par l'algorithme *computed* (indice de Jaccard) :

$$accuracy(t) = \frac{|ideal \cap computed(t)|}{|ideal \cup computed(t)|}$$

On notera que $|ideal \cap computed(t)|$ n'est jamais nul, puisque le pair lui même est toujours contenu dans les deux ensembles eux-mêmes.
Le résultat de cette métrique a valeur dans $]0; 1]$. Plus un algorithme est précis, plus cette valeur tend vers 1.

5.8.4 Sensibilité à la densité : calibrer la simulation

Si les terminaux évoluent dans une aire trop petite, ils vont interagir du fait de la densité de nœuds même s'ils appartiennent à des groupes différents. Dans les simulations où l'on essaye de distinguer plusieurs groupes, il faut donc utiliser une aire simulée assez grande pour éviter ces interactions.

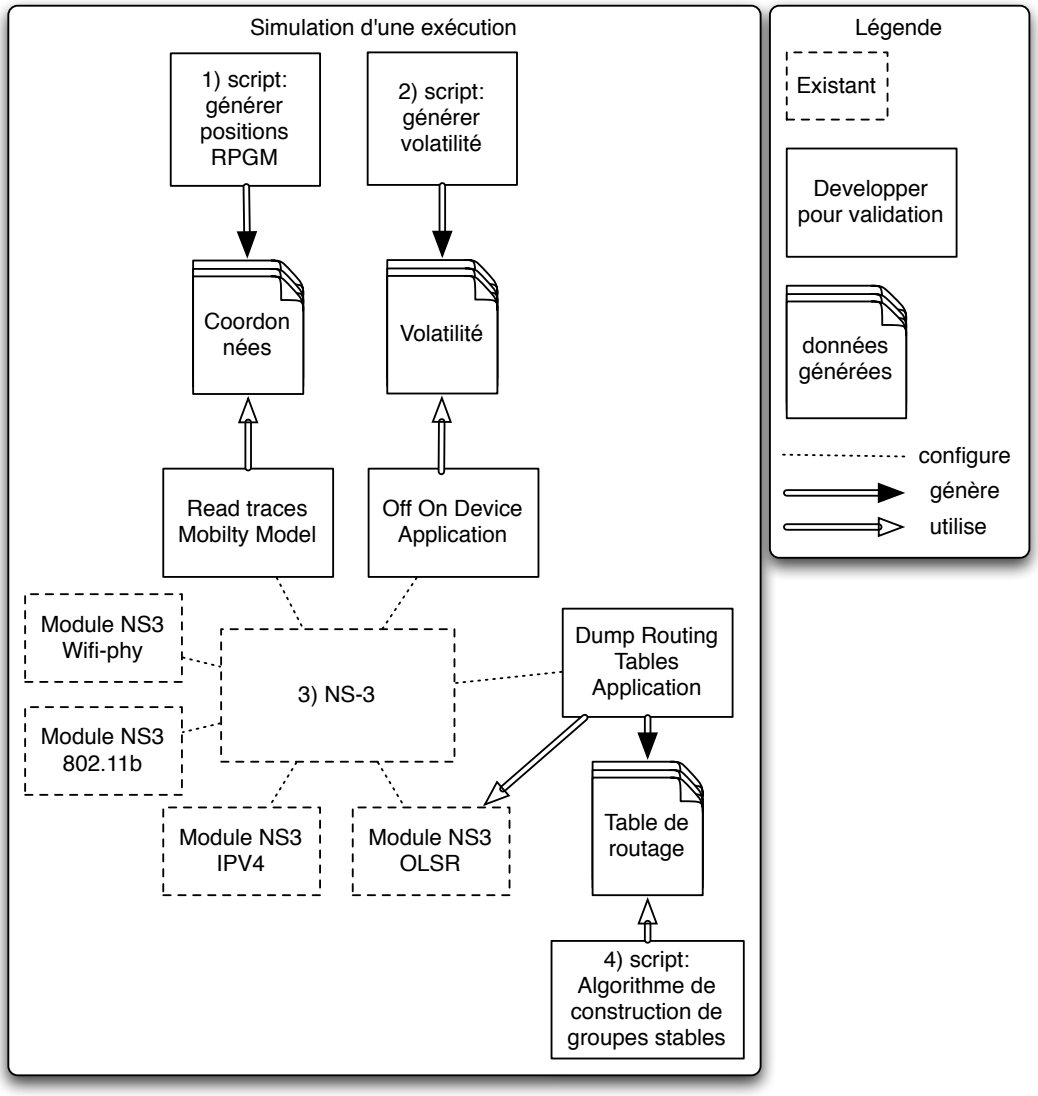

FIGURE 5.9 – Processus de validation

Ce problème est aussi lié à l'aire de couverture d'un groupe, qui dépend elle-même de comment la mobilité est simulée. La figure 5.10 représente différentes organisations possibles du groupe simulé.

Dans notre simulation, nous utilisons le modèle RPGM, correspondant à la figure 3 dans la figure 5.10, que nous avons présenté dans nos hypothèses. La portée de communication maximum de 802.11b en extérieur étant de 100m, l'aire de couverture maximale d'un groupe est donc un disque de diamètre 200m.

Par la suite, nous avons choisi la taille de l'aire de simulation de manière à pouvoir placer les groupes hors de portée de communication.

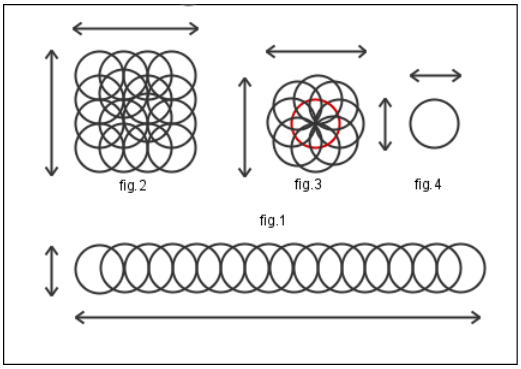

Figure 5.10 – Plusieurs configurations possibles pour le groupe

5.8.5 Scenarii

Afin de valider notre algorithme, nous avons déterminé différentes situations particulières à tester.
Tout d'abord, nous simulons les interactions entre groupes, sans absence transitoire, afin de tester le paramètre du seuil de stabilité *stableThreshold* ainsi que de *maxAbs*, permettant de détecter la séparation d'un groupe.
– le cas par défaut : déplacement de groupes sans interactions.
– deux groupes fusionnent.
– un groupe se sépare en deux.
Nous introduisons ensuite des fautes transitoires, afin de vérifier l'impact de *maxAbs*.
Enfin, nous exécutons un scénario général, avec une mobilité de type RPGM, sans trajectoire déterminée, et des absences transitoires des pairs.

5.8.5.1 Un groupe mobile

Dans ce scénario, illustré par la figure 5.11 nous voulons tester la validité de notre algorithme dans le cas le plus simple : un groupe de pairs se déplace sans qu'il y ait de fautes ou d'interaction avec d'autres groupes.

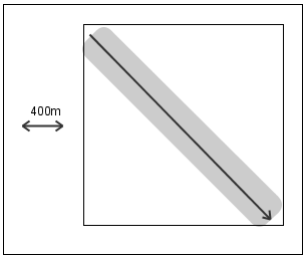

Figure 5.11 – Un groupe

Scénario : un groupe de pairs se déplace en diagonale depuis le coin haut gauche de l'aire simulée. On veut savoir quelle valeur de *stableThreshold* maximise l'exactitude de notre algorithme. Celle-ci varie entre 100 et 180 dans notre test.
Résultats attendus : on s'attend à ce que le groupe simulé ne soit pas correct entre 0 et 2**stableThreshold* secondes ; l'exactitude est ensuite de 1 pour le reste de la simulation. La plus petite valeur de *stableThreshold* devrait donc maximiser l'exactitude de l'algorithme.

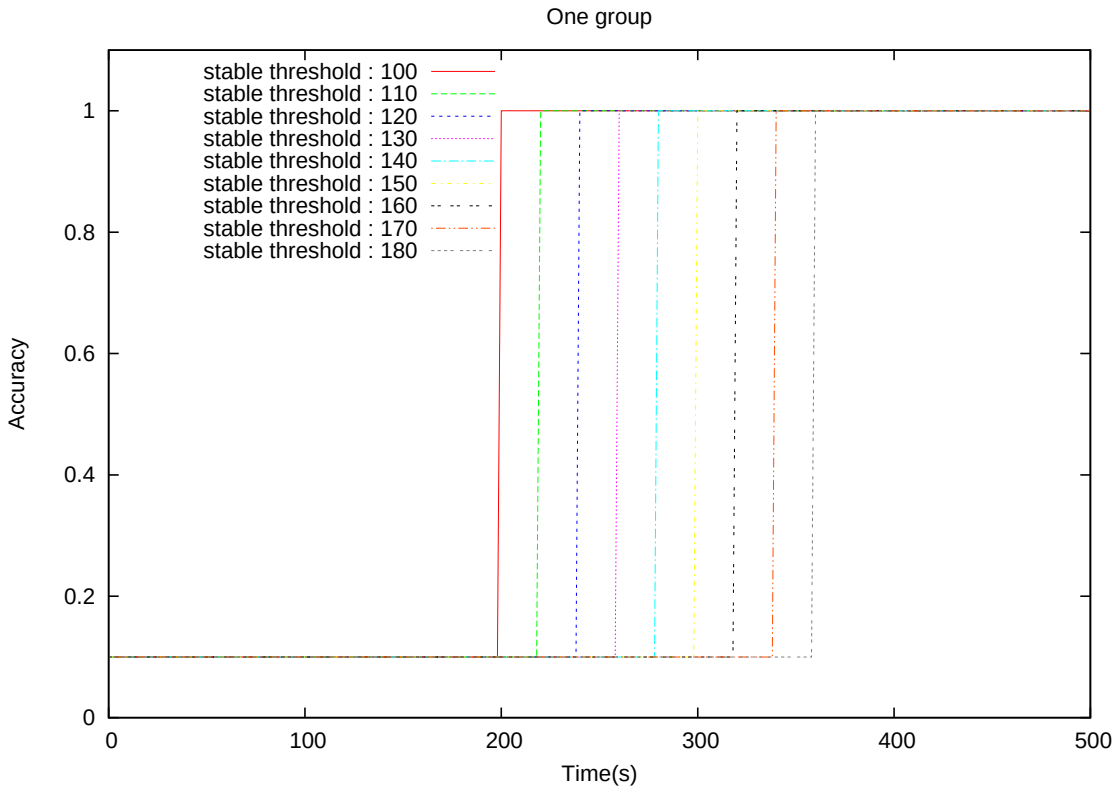

FIGURE 5.12 – Un groupe, $\neq$ seuils de stabilité

Résultats observés : le graphe 5.12 montre les résultats de la simulation. Chaque courbe représente l'exactitude de l'algorithme en fonction du temps pour une valeur donnée de *stableThreshold*. On peut voir que la simulation valide les résultats attendus : pour une valeur de *stableThreshold* plus grande, la formation du groupe prend plus longtemps.
Notons que l'algorithme est exécuté toutes les 2 secondes, et en l'absence d'erreurs, détecte tous les terminaux comme faisant parti de son groupe stable, au bout de 2**stableThreshold*.
Le passage de la précision de 0 à 1 apparait comme une impulsion, mais cela est du à l'échelle du graphe.

5.8.5.2 Deux groupes sans interaction

Dans le scénario illustré par 5.13, nous testons si les simulations sont correctement calibrées : étant données la taille de l'aire simulée et la portée de communication, deux groupes se déplaçant dans des zones non recouvrantes ne devraient pas interagir.
Scénario : deux groupes de terminaux suivent les bords opposés de l'aire. Ils ne sont jamais en contact. Le paramètre *stableThreshold* varie entre 100 et 180
Résultats attendus : les deux groupes n'interagissant pas, le graphe obtenu devrait être semblable à 5.12

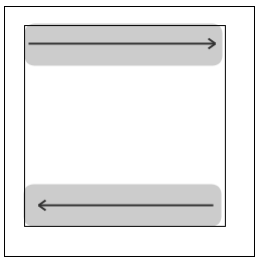

Figure 5.13 – Deux groupes sans interactions

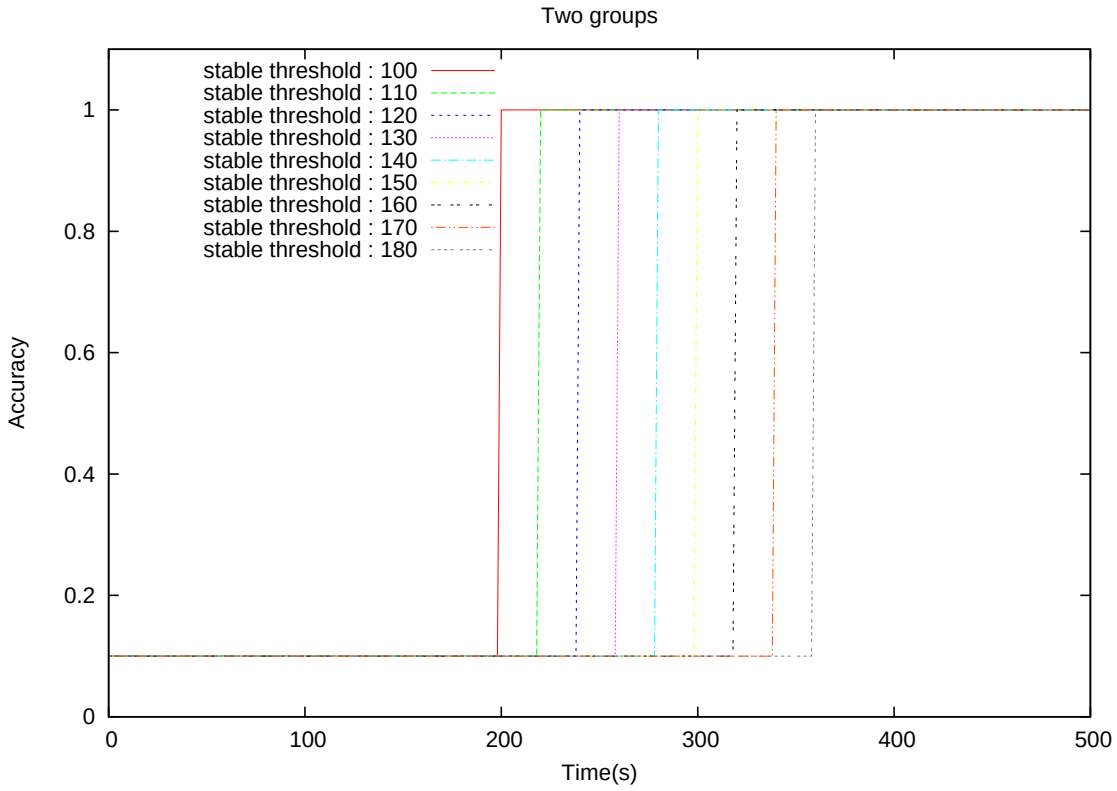

Figure 5.14 – Deux groupes, $\neq$ seuils de stabilités

Résultats observés : le graphe 5.14 montre que la précision de l'algorithme varie comme dans l'expérience précédente. La simulation est donc bien calibrée.

5.8.5.3 Deux groupes dont les chemins se croisent

On cherche ici à valider le comportement de l'algorithme dans le cas où deux groupes se croisent.
Tout d'abord, nous vérifions pour un seuil de stabilité fixe, la précision de l'algorithme de

grappes en fonction de l'angle des trajectoires des deux groupes. Nous vérifions ensuite pour différentes valeurs du seuil de stabilité.

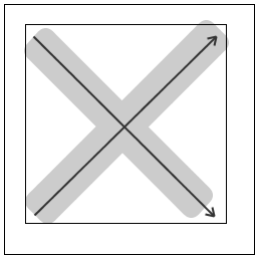

FIGURE 5.15 – Deux groupes dont les chemins se croisent

Scénario : dans ce scénario, on veut vérifier le comportement de l'algorithme en fonction de la valeur du seuil de stabilité *stableThreshold*, qui varie entre 100 et 180. Les deux groupes partent chacun en diagonale des coins droits de l'aire simulée et leurs trajectoires se croisent à angle $\frac{\pi}{2}$rad au centre de l'aire à t=1415. On observe la précision de l'algorithme autour de cette date.
Résultats attendus : pour les valeurs de *stableThrehold* supérieures à 140, la précision de l'algorithme devrait rester à 1. En dessous de 140, la précision devrait se détériorer comme la valeur de *stableThreshold* décroît.
Résultats observés : la figure 5.16 présente le résultat, chaque courbe représentant une simulation pour une valeur différente de *stable threshold*. On peut voir que les résultats sont même meilleurs que ceux attendus, et les deux groupes restent distincts pour un seuil de stabilité de 120 : seuls des seuils de stabilité de 110 et 100 créent une confusion entre les deux groupes.
Scénario : dans ce jeu de scénarios, deux groupes sont placés hors de portée de communication, et leurs trajectoires se croisent au centre de l'aire. L'angle α formé par les trajectoires varie entre 0 et π rad. Leurs positions de départ sont choisies de manière à ce que les groupes se croisent à t=1000, comme illustré dans la figure 5.17. Nous voulons vérifier que pour le paramètre *stableThreshold* fixé à 140, l'algorithme sépare bien les groupes dont les trajectoires ont des angles supérieurs ou égaux à $\frac{\pi}{2}$ rad.
Résultats attendus : si les deux groupes ont des trajectoires d'angle 0, celles ci sont parallèles. Comme on sait qu'elles se croisent par construction à t=1000, elles sont donc confondues. Les deux groupes évoluent en fait comme un et la courbe de précision devrait valoir 0,5. Pour les valeurs de α dans $[0\,;\,\frac{\pi}{2}]$, l'erreur diminue comme l'angle grandit. Pour les valeurs supérieures à $\frac{\pi}{2}$, la précision de l'algorithme reste à 1.
Résultats observés : dans la figure 5.18, chaque courbe représente l'évolution de la précision de l'algorithme pour un angle α donné. On peut voir que les résultats attendus pour α=0 et $\alpha > \frac{\pi}{2}$ sont ceux obtenus. La précision de l'algorithme est bien meilleure quand α grandit et l'on voit même que le temps *error* durant lequel l'algorithme est confus est inférieur à celui calculé théoriquement. Pour $\alpha = 45°$ par exemple, *error* vaut 4mn40, une valeur inférieure même à celle attendue pour $\alpha = 50°$ de 5mn53.

5.8.5.4 Deux groupes fusionnent

Scénario : dans ce scénario, nous voulons observer le comportement de l'algorithme quand deux groupes fusionnent. Comme illustré par la figure 5.19 deux groupes partant des coins bas opposés de l'aire simulée se rejoignent au centre de l'air pour fusionner, à t=1415, en

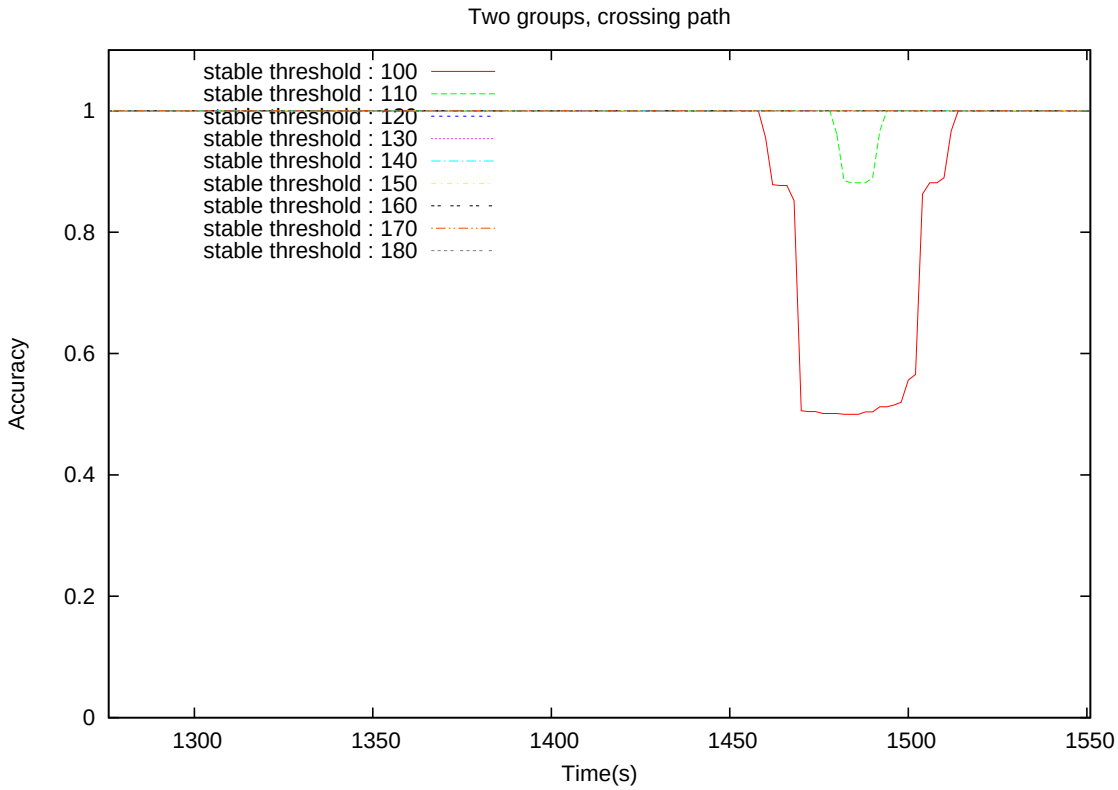

Figure 5.16 – Deux groupes dont les chemins se croisent, $\neq$ seuils de stabilité

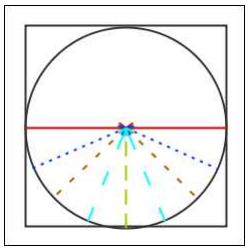

Figure 5.17 – Même distance avant rencontre, angles différents

un seul groupe qui se dirige vers le haut. Nous voulons vérifier que l'algorithme détecte la fusion, et le temps nécessaire à cela, en fonction de différentes valeurs de *stableThreshold* qui varie donc entre 100 et 180.

Résultats attendus : plus le seuil de stabilité *stableThreshold* est bas, plus la fusion des groupes sera rapidement détectée.

Résultats observés : la figure 5.20 présente le résultat de l'expérimentation sur la fusion de groupes. On peut voir que, comme on l'attendait, un seuil de stabilité de 100 permet de fusionner les deux groupes plus rapidement qu'un seuil de 180.

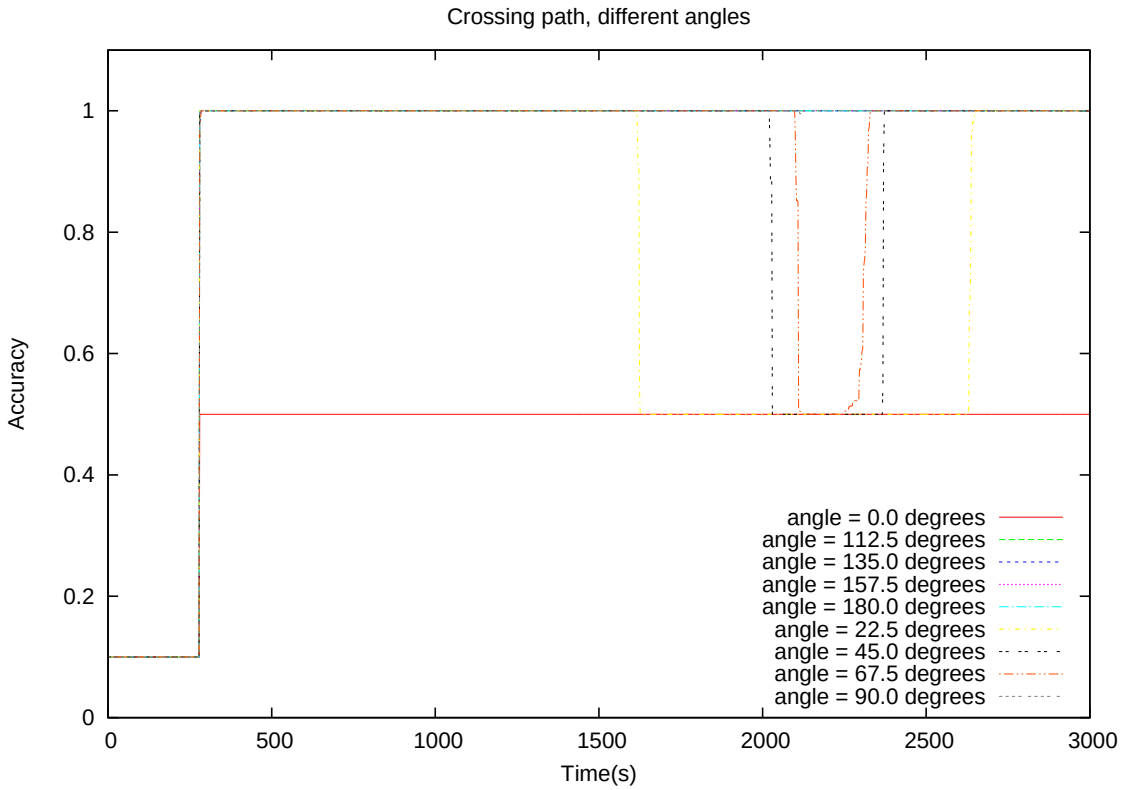

FIGURE 5.18 – 2 groups crossing, $\neq$ angles

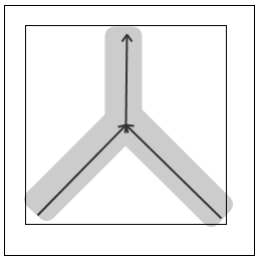

FIGURE 5.19 – Fusion de deux groupes

Notons que cette expérience est une généralisation du cas d'un pair rejoignant le groupe, ce qui valide donc notre cas de test.

5.8.5.5 Un groupe se sépare en deux

Dans ce scénario, illustré par la figure 5.21 un groupe de pairs se déplace assez longtemps pour que le groupe soit détecté par l'algorithme proposé, avant de se séparer en deux. On examine le comportement de notre algorithme au moment de la séparation.

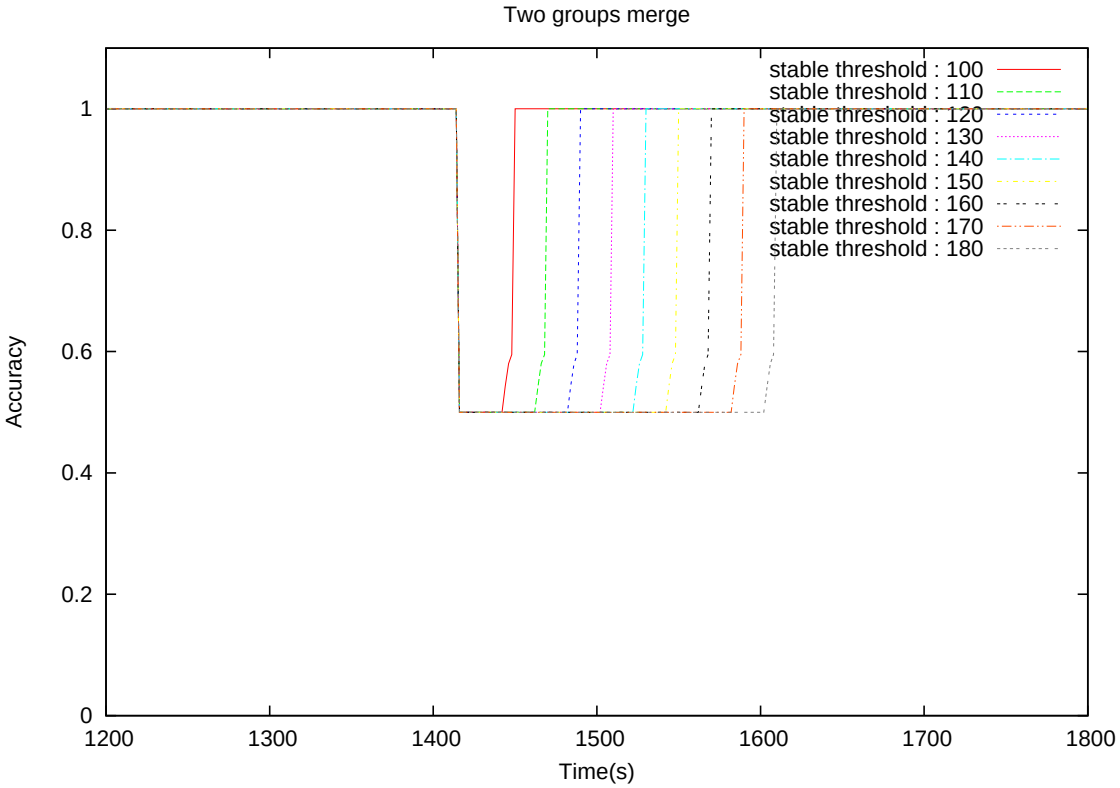

Figure 5.20 – Deux groupes fusionnent, $\neq$ seuils de stabilité

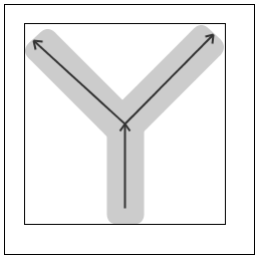

Figure 5.21 – Un groupe se sépare en deux groupes

Scénario : dans ce scénario, un groupe se déplace du centre bas de l'aire simulée au centre. Il se sépare alors en deux groupes, chaque groupe partant en diagonale vers les coins opposés de l'aire. On veut vérifier que notre algorithme détecte bien la séparation, et mesurer le temps nécessaire en fonction du nombre maximum d'absences tolérées. Le paramètre *stableThreshold* est fixé à 140, et le paramètre maxAbs varie entre 20 et 90.
Résultats attendus : on examine le comportement de notre algorithme autour de t=1000 sec, moment de la séparation en deux groupes qui partent dans des direction distinctes. L'algorithme ne peut pas être absolument exact car après la séparation les 2 groupes restent

encore en communication pour, au plus, 224 sec. Cependant, nous attendons de ce test que plus le paramètre *maxAbs* est bas, plus notre algorithme se corrige rapidement.

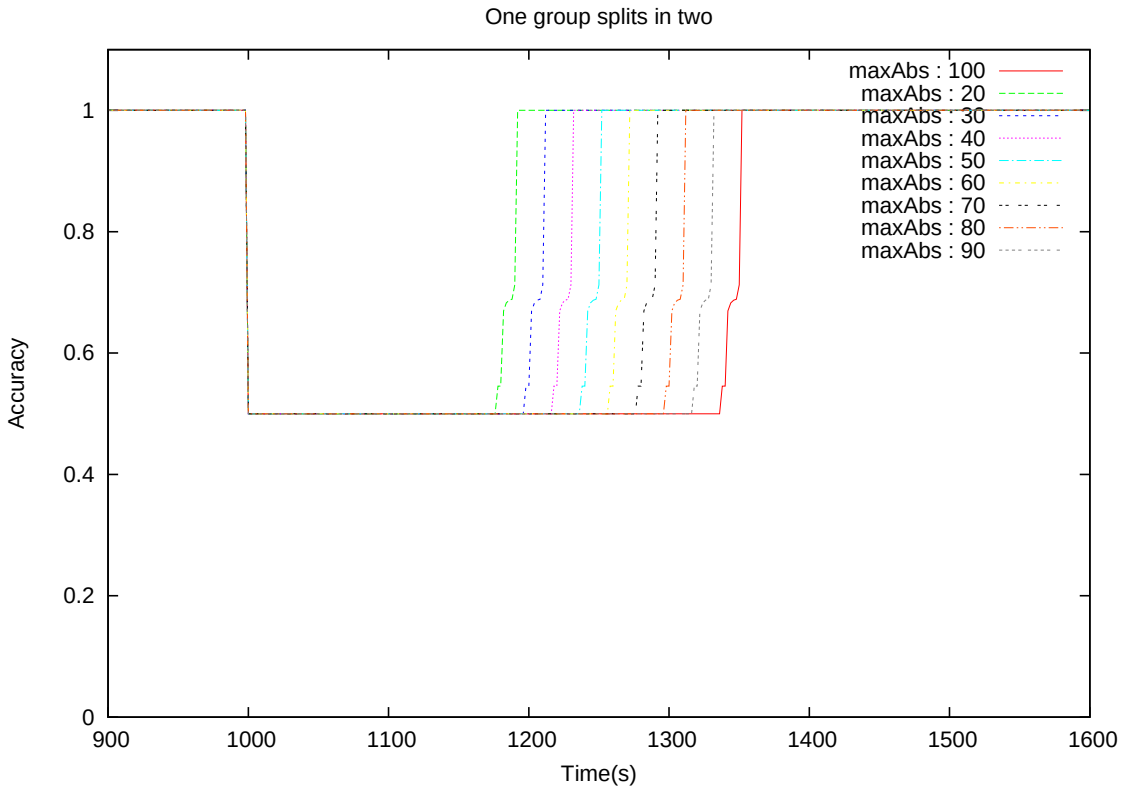

FIGURE 5.22 – Un groupe se sépare en 2, $\neq$ valeurs pour le nombre maximum d'absences tolérées

Résultats observés : dans la figure 5.22, chaque courbe représente l'évolution de la précision de l'algorithme au cours du temps en fonction d'une valeur donnée de *maxAbs*. On voit que la séparation des deux groupes est plus rapidement effective quand *maxAbs* est bas, ce qui valide nos attentes.
Notons que cette expérience est une généralisation du cas d'un pair quittant un groupe, et valide donc ce cas de test.

5.8.5.6 Un groupe mobile, absence transitoire d'un pair

Dans ce cas de test nous voulons évaluer l'impact de la durée d'une absence sporadique d'un terminal sur la précision de l'algorithme. Ces absences sont absorbées dans une certaine mesure puisque *maxAbs* absences sont normalement tolérées.
Scénario : un groupe de 10 terminaux se déplace selon un modèle RPGM et un pair subit des absences transitoires. Ces absences sont de plus en plus longues, de 1 à 7 minutes.

L'intervalle entre deux absences est assez long pour que le groupe retrouve sa stabilité. Le paramètre *stableThreshold* est fixé à 140 et *maxAbs* varie entre 20 et 100.
Résultats attendus : le pair fautif disparaît ainsi : à t=560 pour 1min, à t=1020 pour 2min, à t=1540 pour 3min, à t=2120, pour 4mn, à t=2760 pour 5min, à t= 3460 pou 6 min et à t=4220 pour 7min. Pour une valeur de *maxAbs* donnée, seules les absences de durée inférieure à 2**maxAbs* n'impactent pas la précision de l'algorithme. Par exemple, pour une absence de 2 minutes, seules les courbes 20, 30,40 et 50 devraient subir un changement.

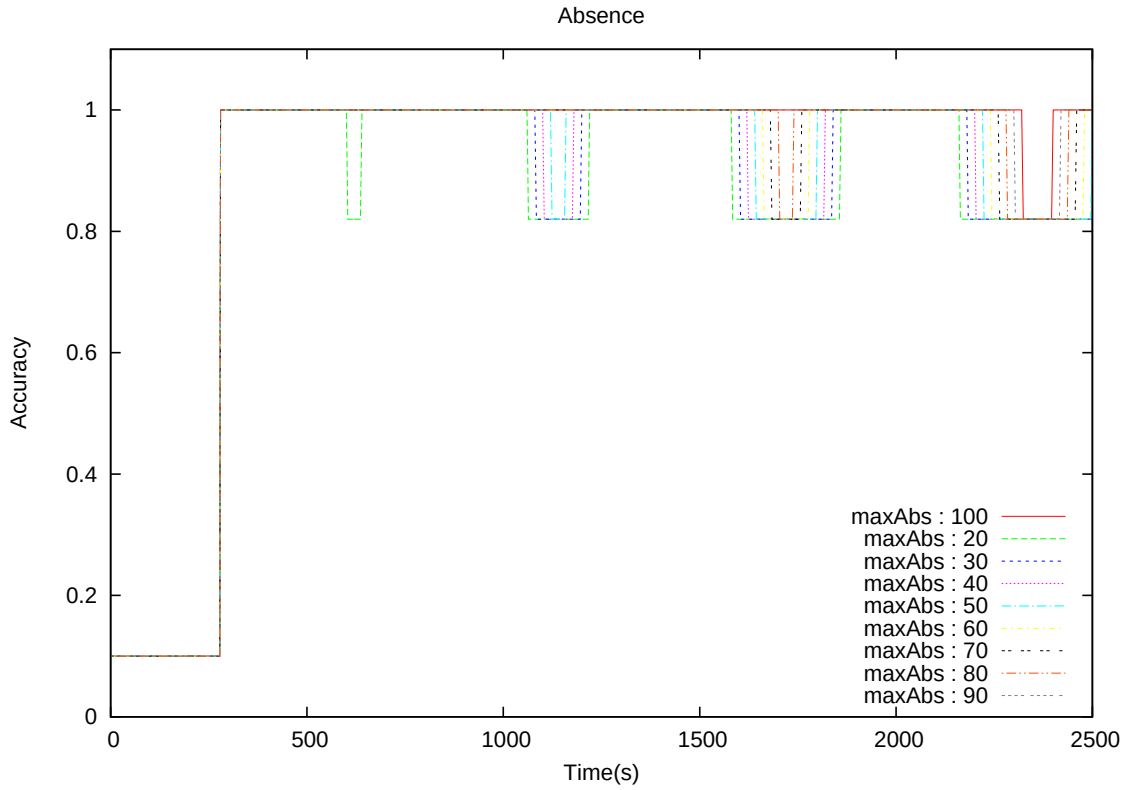

Figure 5.23 – Un groupe se sépare en deux groupes, absence de 1,2,3,4 minutes

Résultats observés : les figures 5.23 et 5.24 présente l'évolution de la précision de l'algorithme de groupe en fonction du temps. On a bien le résultat attendu : une absence d'une minute est considérée comme une sortie de groupe seulement pour *maxAbs* inférieur à 30, une absence de deux minutes pour *maxAbs*<60, etc.

5.8.5.7 Plusieurs groupes aux déplacements aléatoires absence transitoire

Pour finir, nous avons évalué notre algorithme dans un cadre plus général.
Dans ce scénario, 4 groupes de 30 terminaux évoluent dans une aire carrée de 5000mx5000m pendant 2 heures. Chaque groupe suit un modèle de mobilité *Reference Point Group Mobility*, où le point de référence suit un modèle de mobilité *Random Waypoint*.

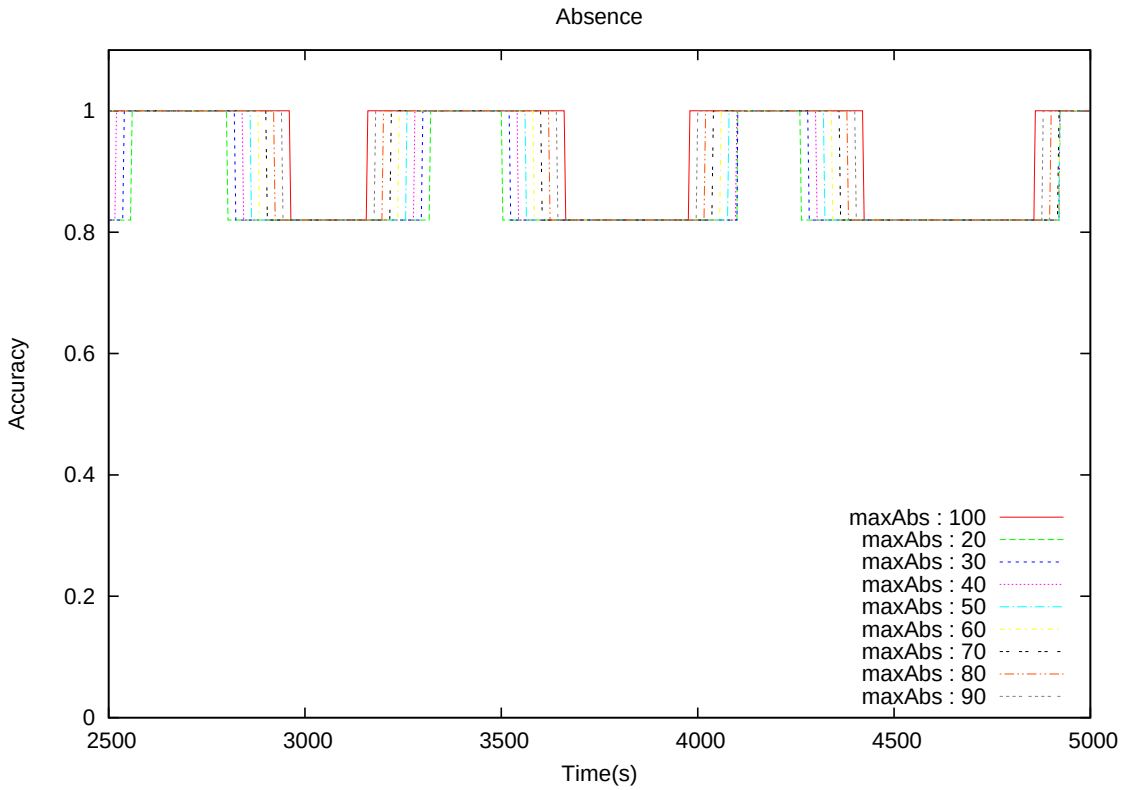

FIGURE 5.24 – Un groupe se sépare en deux groupes, absence de 5, 6, 7 minutes

Chaque terminal est susceptible d'absences transitoires, modélisées de la façon suivante :
- le temps d'absence est modélisé par une loi Normale de paramètre μ=60 et σ=30 : en moyenne une absence dure 1 minute, avec une dispersion de 30s.
- le temps d'intervalle entre deux absences est modélisé par une loi Normale de paramètre μ=5*60 et σ=60 : en moyenne, le temps entre deux absences est de 5 minutes, avec une dispersion d'1 minute.

Le temps moyen d'absence étant inférieur à 2 minutes, la précision de l'algorithme devrait donc rester à 1.
Comme les figures 5.25 et 5.26 le montrent, l'algorithme proposé produit un bon résultat dans le cas général : bien que subissant de faibles variations, la précision de l'algorithme reste proche de 1.

5.8.6 Synthèse

Dans cette section nous avons validé notre algorithme par simulation dans des cas précis afin de tester les paramètres de notre algorithme, *maxAbs* et *stableThreshold*. Nous avons ensuite vu un cas général avec des déplacements de groupes aléatoires.
Ceci nous a permis de valider que :

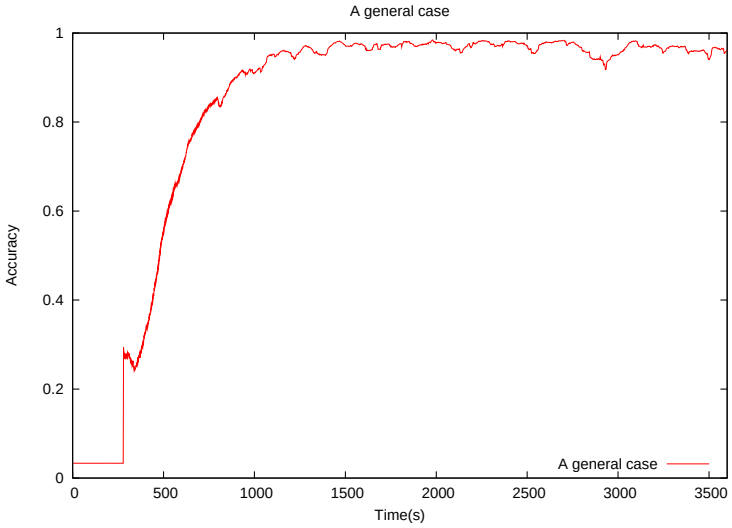

Figure 5.25 – 1ère heure

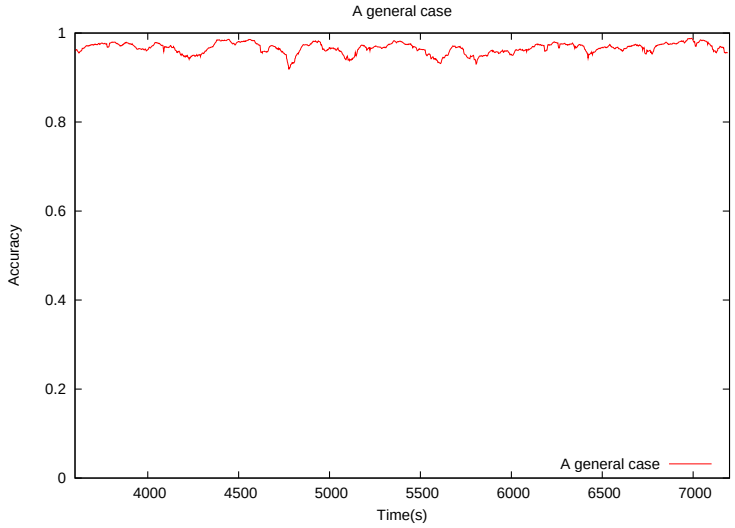

Figure 5.26 – 2ème heure

– Quand l'angle d'approche est supérieur ou égal à 90°, notre algorithme peut différencier deux groupes se croisant de deux groupes fusionnant ; dans le cas contraire, il répare son erreur dans un temps inférieur à la valeur théorique calculée.
– Notre algorithme peut supporter les absences transitoires des terminaux, de durée inférieure à 2 minutes.

5.9 Comparaison à l'existant

Le point fort de cet algorithme est *l'absence de surcoût réseau*. On notera d'ailleurs que la plupart des propositions existantes n'évaluent pas leur algorithme en terme de messages. En effet, tous les autres algorithmes nécessitent un échange périodique d'informations (du moins coûteux au plus coûteux) :
– à la tête de grappe une fois que celle ci est élue : [43] et [113]
– à un saut : [11], [97] et [114]
– à l'intégralité du réseau : [7] et [103]
Notre algorithme ne crée pas de surcoût réseau car il s'appuie sur des informations inter-couche issues des couches de routage. Il nécessite donc un protocole de routage pro-actif, et passe donc à l'échelle dans les limites de celui-ci.
Par ailleurs, notre proposition est totalement distribuée, et ne s'appuie pas sur un serveur ([103] et [24]), ni sur la nécessité d'élire une de tête de grappe pour lui déléguer des décisions.
Nous n'utilisons pas de système de position, comme GPS, ni de synchronisation d'horloge, comme [7] [97] [103], [43] et probablement [104] et [114].
Nous l'avons validé sur NS-3, un simulateur reconnu par la communauté des chercheurs en réseaux.
Enfin, nous en proposons un prototype concret et ne faisons pas uniquement une évaluation mathématique théorique, contrairement à [104] , [38] et [25].

5.10 Synthèse et conclusion

Dans un environnement où les utilisateurs sont mobiles, pour améliorer l'efficacité de certains services, comme la diffusion de messages (choisir des relais stables pour la multi-diffusion), ou le stockage collaboratif de données (stocker pour des terminaux susceptibles de rester présents assez longtemps pour utiliser la donnée), nous voulons pouvoir déterminer parmi un essaim de terminaux, quels sont les sous-groupes ayant une mobilité similaire. On a donc besoin d'un algorithme de création de grappes.
Le temps d'une session de travail étant limité par la durée de vie de la batterie, et la carte wifi étant une des sources majeures de consommation, il est important, quand on crée un algorithme, de limiter les communications.
Dans cette section nous avons donc proposé un algorithme de construction de groupe stable dans le temps qui se base sur des informations inter-couches (les tables de routages d'un protocole pro-actif, OLSR) à usage pour un groupe de piétons.
Etant donné les problèmes pour effectuer une validation dans un environnement réel, nous l'avons validé sur le simulateur NS-3. Notons que c'est le cas de la plupart des algorithmes de gestion de la mobilité, car la simulation est beaucoup moins coûteuse en ressources et permet la reproduction des expérimentations à volonté.
Notre algorithme a l'avantage de ne pas créer de surcharge réseau. On notera que les algorithmes existants sont validés de manière théorique, et par simulation, mais que peu d'entres eux examinent le surcoût en communications réseau. Il est totalement distribué, et ne dépend pas d'un matériel spécifique pour obtenir la position des terminaux. Il est cependant dépendant de l'utilisation d'un protocole de routage pro-actif.
Dans les chapitres suivants, nous allons voir comment ces groupes stables sont utilisés pour implémenter un algorithme collaboratif de réplication de données.