République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université A. Mira de Béjaia
Faculté des Sciences Exactes
Département d'Informatique

Mémoire de Fin de Cycle

En vue de l'obtention du diplôme Master en Informatique Option : Administration et sécurité des réseaux

Thème

Implémentation d'un algorithme d'élection de leader dans les réseaux Ad-Hoc

Réaliser par :

Mr. BOUICHE Kheireddine
Mr. BOUDRAHEM Moukhtar

Devant le jury composé de :

Président : Mme YAICI Malika

Examinateur : Mr. KHENOUS Lachemi

Examinateur : SKLAB Youcef

Encadré par : Mr. SADI Mustapha

Promotion : 2016/2017

Remerciement

Notre premier remerciement va à Allah soubhanou Wa ta ala.

En terminant notre mémoire de fin d'étude, il nous est agréable d'adresser nos vifs remerciements à tous ceux qui nous ont aidé de près ou de loin à élaborer cet ouvrage.

On tiens à remercier vivement notre encadreur. Mr. SADI Mustapha, pour sa gentillesse, sa disponibilité et sa contribution générale à l'élaboration de ce modeste travail.

Nous souhaiterions également remercier l'ensemble de nos enseignants de la faculté des sciences exactes pour tous le savoir qu'ils nous ont donné toute au long de notre formation à l'université de Béjaia.

A la fin nous tenons à remercier tous nos collègues d'étude, particulièrement notre promotion.

Dédicace

C'est avec un énorme plaisir, un cœur ouvert et une immense joie, que je dédie mon travail à mes très chers, respectueux et magnifique parents ainsi que ma sœur.

Je dédie ce travail en particulier à la mémoire de mon Père Abdelaziz, qu'aucune dédicace ne serait exprimer l'amour, l'estime, le dévouement et le respect que j'ai toujours eu pour vous, rien au monde ne vaut les efforts fournis jour et nuit pour mon éducation et mon bien être. Ce travail est le fruit de tes sacrifices que tu as consentis pour mon éducation et ma formation.

Je tien en particulier à féliciter mon binôme BOUDRAHEM Moukhtar qui s'est montré à la hauteur de la tâche qu'il nous est était confier, ainsi que tous mes amis.

A toute personnes qui m'ont encouragé ou aidé au long de mes études.

Kheireddine

Dédicace

A mes très chers parents

Je vous dois ce que je suis aujourd'hui grâce à votre aide, à votre patience et vos innombrables sacrifices.

Que ce modeste travail, soit pour vous une petite compensation et reconnaissance envers ce que vous avez fait d'incroyable pour moi. Que dieu, le tout puissant, vous préserve et vous procure santé et longue vie afin que je puisse à mon tour vous combler.

A mes très chères sœurs, frères

Aucune dédicace ne serait exprimée assez profondément ce que je ressens envers vous.
Je vous dirai tout simplement, un grand merci, je vous aime.

A mes très chers ami(e)s

En témoignage de l'amitié sincère qui nous a liées et des bons moments passés ensemble. Je vous dédie ce travail en vous souhaitant un avenir radieux et plein de bonnes promesses.

En souvenir de nos éclats de rire, des bons moments et des nuits blanches.

En souvenir de tout ce qu'on a vécu ensemble. J'espère de tout mon cœur que notre amitié durera éternellement.

Moukhtar

Table des matières

Chapitre 1 : Généralités sur les réseaux et systèmes distribués dynamiques

Chapitre 2 : Le problème d'élection de leader

Liste des figures

Liste des tableaux

Liste des abréviations

DAG	Directed Acyclic Graphe
MANET	Mobile Ad hoc NETwork
PAN	Personal Area Network
IP	Internet Protocol
MST	Minimum Spanning Tree
TORA	Tomporally Ordered Routing Algorithme, TORA est un protocole de routage dans les réseaux Ad-hoc
IEEE	Institute of Electrical and Electronics Engineers
IEEE P802.11/D10	Télécommunications et échange d'informations entre systèmes
IDE	Integrated Development Environment
FIFO	First In First Out
UPD	UPDATE

Introduction générale

Dans les systèmes répartis conventionnels, le problème d'élection d'un leader a été intensivement étudié sur une grande variété de modèles et de topologies. Cependant, peu de travaux ont traité ce problème dans un réseau mobile ad hoc. Dans un tel environnement, caractérisé essentiellement par des changements fréquents de topologie induits par la mobilité des sites, l'élection d'un leader devient plus compliquée. Et la condition essentielle que l'algorithme d'élection doit satisfaire est d'élire finalement un leader unique d'un ensemble fixe de nœud avec des paramètres bien défini au préalables.

Dans ce travail, nous présentons les différentes solutions qui existent dans la littérature pour résoudre le problème d'élection dans les réseaux mobiles ad hoc, et ensuite nous présentons la phase d'implémentation d'un algorithme d'élection de leader dans les réseaux Ad-Hoc et de le comparé avec un autre algorithme d'élection en terme de message générer lors de l'élection et en tenant compte de quelques paramètres, et cela en mettons en place un simulateur qui va le permettre.

Dans le présent projet, nous présentons en détail les étapes que nous avons suivi pour arriver à bout de ce projet, illustrées en quatre chapitres organisé comme suite :

Le premier chapitre s'intitule « **Généralités sur les réseaux et systèmes distribués dynamiques** », nous présentons un aperçu général sur les réseaux Ad-Hoc et les systèmes distribués.

Dans le deuxième chapitre nommé « **Le problème d'élection de leader** », nous présentons les différents algorithme d'élection dans les réseaux Ad-Hoc.

Dans le troisième chapitre titré « **Etude de l'algorithme de Vasudevan, Kurose, Towsely** », nous étudions un algorithme d'élection de leader.

Dans le quatrième chapitre nommé « **Implémentation de l'algorithme de Vasudevan, Kusrose et Towsely** », nous présentons l'implémentation de l'algorithme d'élection qui est effectué par un simulateur réalisé par nous-même ensuite une évaluation des performances de l'algorithme est présentée en fin du chapitre. Ensuite on finit avec une conclusion générale.

CHAPITRE 1

Généralités sur les réseaux et systèmes distribués dynamiques

Introduction

L'évolution rapide de la technologie dans le domaine de la communication sans fil, a permis à des usagers munis d'unités de calcul portables d'accéder à l'information indépendamment des facteurs, temps et lieu. Ces unités, qui communiquent à travers leurs interfaces sans fil, peuvent être de divers configuration : avec ou sans disque, des capacités de sauvegarde et de traitement plus ou moins modestes et alimentées par des sources d'énergie autonomes (batteries). L'environnement de calcul résultant est appelé environnement mobile. Cet environnement n'astreint plus l'usager à une localisation fixe, mais lui permet une libre mobilité tout en assurant sa connexion avec le réseau.

Un réseau MANET est facilement installable et ainsi, très économique et un environnement populaire en informatique ; par contre c'est aussi un environnement très contraignant. Un nœud portable limite le calculateur de puissance, de petite batterie pour le système de secours, et un petit champ de communications, ils peuvent communiquer uniquement par messages qui passent par des liens sans fil, les nœuds qui ne sont pas dans le champ de communication peuvent communiquer par message de relaie. La mobilité non déterministe conduit vers des changements arbitraires de topologie, le changement devient plus fréquent à cause du dynamisme dans les liens sans fil et la limitation de la bande passante. Par conséquent, les algorithmes distribués sont développés pour les domaines statiques et non directement implémentés dans les réseaux sans fil ad hoc.

De nos jours, il existe de nombreuses applications distribuées qui sont exécutés dans MANET, par exemple : problème d'accord, échange de données, agrégation de données, distribution de clés, communication de groupe, etc.

1. MANET

Les réseaux mobiles ad hoc (MANET) sont formés dynamiquement par un système autonome de nœuds mobiles qui sont connectés via des liaison sans fil, sans utiliser l'infrastructure de réseau existant ou l'administration centralisée.

En fait, deux ou plusieurs nœuds peuvent former le réseau mobile ad hoc en étant dans leur portée de transmission.

Dans ce type de réseaux, la communication entre les nœuds mobiles est point-à-point, de sorte que chaque nœud a une communication directe avec un autre. Les nœuds agissent également comme nœuds relais pour transférer des paquets de données.

C'est une partie très importante de la technologie de communication qui soutient l'informatique omniprésente, car dans de nombreux contextes, l'échange d'informations entre les unités mobiles ne peut s'appuyer sur aucune infrastructure de réseaux fixe mais sur la configuration rapide de la connexion sans fil à la volé [1].

Les réseaux mobiles ad hoc ont du succès parce qu'ils aident à réaliser le service réseau pour les utilisateurs mobiles dans les zones sans infrastructure de communication préexistante ou lorsque l'utilisation de cette infrastructure nécessite une extension sans fil. Les nœuds ad hoc peuvent également être connectés à un réseau de base fixe via un dispositif de passerelle dédié permettant des services de routage IP dans la zone où le service internet n'est pas disponible en raison du manque d'infrastructure préinstallée.

La configuration minimale et le déploiement rapide rendent les réseaux ad hoc adaptés à des situations d'urgence comme les catastrophes naturelles ou humaines, les conflits militaires, les situations médicales d'urgence, etc. La figure suivante montre un modèle de réseau ad hoc avec infrastructure.

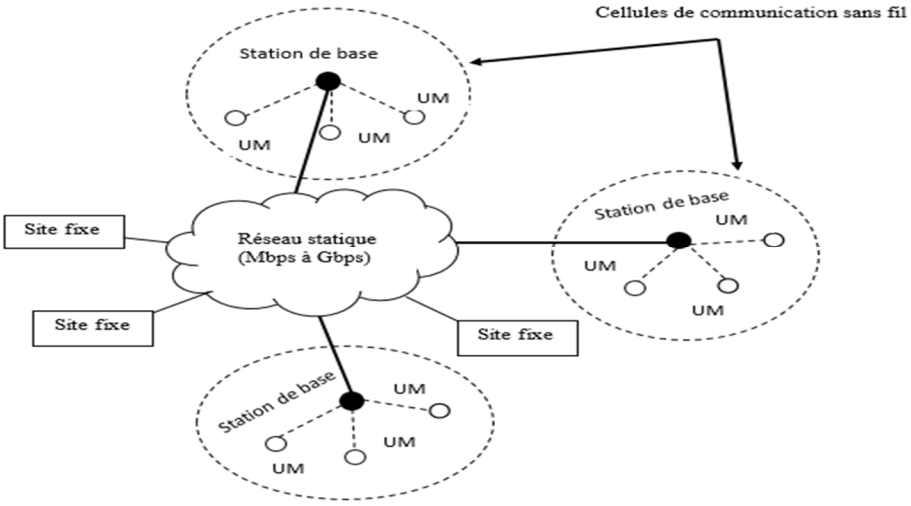

Figure 1.1 - Le modèle des réseaux mobiles avec infrastructure

1.2 Caractéristiques des réseaux mobiles ad hoc

Un réseau mobile ad hoc est constitué de plates-formes mobiles (i.e. un routeur avec plusieurs hôtes et éléments de communication sans fil). Actuellement appelés nœuds qui sont libres de se déplacer arbitrairement. Ces nœuds peuvent être dans des avions, des bateaux, voir même sur des personnes ou sur des éléments extrêmement petits, éventuellement avec plusieurs hôtes par routeur. Un MANET est un système autonome de nœuds mobiles. Ce système peut être isolé, mais il peut aussi avoir des passerelles ou des interfaces le liant à un réseaux fixe. Dans son fonctionnement futur, on le voit typiquement comme un réseaux 'final' rattaché à un réseau d'interconnexion. Un réseau final gère le trafic créé ou à destination des nœuds internes, mais ne permet pas à un trafic extérieur de transiter par lui [2].

Les nœuds des réseaux mobiles ad hoc sont équipés d'émetteurs et de récepteurs sans fil utilisant des antennes qui peuvent être omnidirectionnelles (diffusion), fortement directionnelles (point-à-point), probablement orientable, ou une combinaison de tout ça. A un instant donné, en fonction de la position des nœuds, de la configuration de leur émetteur-récepteur, des niveaux de puissance de transmission et d'interférence entre les canaux, il y a une connectivité sans fil qui existe entre les nœuds, sous forme de graphe multi-saut aléatoire ou de réseau ad hoc. Cette topologie ad hoc peut changer avec le temps en fonction du mouvement des nœuds ou de l'ajustement de leurs paramètres d'émission-réception [2]. Les réseaux mobiles ad hoc ont plusieurs caractéristiques particulières :

- Topologies dynamiques : les nœuds sont libres de se déplacer arbitrairement, ce qui fait que la topologie du réseau typiquement multi-saut peut changer aléatoirement et

rapidement n'importe quand, et peut être constituée à la fois de liaisons unidirectionnelles et bidirectionnelles. La figure1.2 montre le changement de topologie dans les réseaux ad hoc [35].

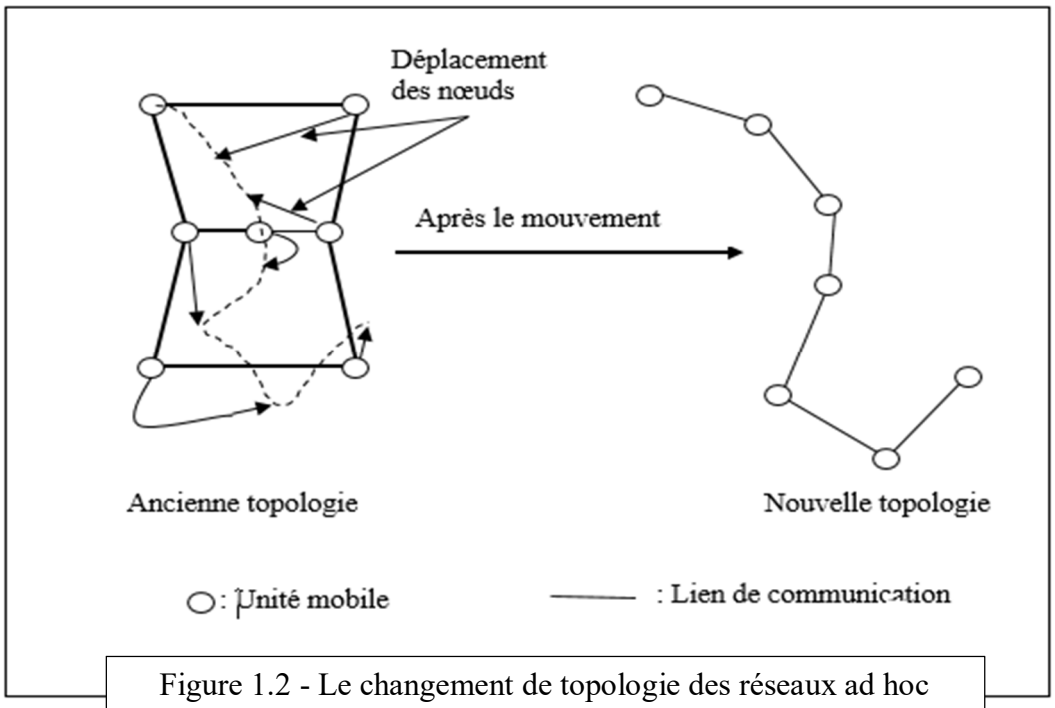

Figure 1.2 - Le changement de topologie des réseaux ad hoc

- Liaison à débit variable et à bande passante limitée : les liaisons sans fil auront toujours une capacité inférieure à leurs homologues câblés. En plus, le débit réel des communications sans fil après avoir déduit les effets des accès multiples, du bruit, des interférences, etc. est souvent inférieur aux taux de transfert maximum des ondes radios.
- Utilisation limitée de l'énergie : une partie des nœuds d'un réseau mobile ad hoc, voir l'ensemble des nœuds, peut se reposer sur des batteries ou un autre moyen limité pour puiser leur énergie. Pour ces nœuds, le plus important est sans doute de mettre en place des critères d'optimisation pour la conservation de l'énergie.
- Sécurité physique limités : les réseaux mobiles sans fil sont généralement plus sensibles aux menace physique que le sont les réseaux câblés fixes. Les possibilités accrues d'attaques par écoute passive, par usurpation d'identité et par déni de service doivent être étudiées avec attention. Les techniques pour la sécurité des liaisons sont souvent appliquées au sein des réseaux sans fil pour réduire les risques d'attaques. Notons cependant un avantage dans le fait que le contrôle des réseaux mobiles ad hoc soit décentralisé ajoute à sa robustesse. Contrairement aux problèmes pouvant survenir sur les points centraux dans des approches centralisées.

En plus, certains réseaux envisagés (i.e. réseaux militaires ou d'autoroutes) pourront être relativement étendus (i.e. des dizaines ou des centaines de nœuds par espace de routage). Le besoin de scalabilité n'est pas réservé aux réseaux mobiles ad hoc. Quoiqu'il en soit, les mécanismes dont on a besoin pour assurer la scalabilité existent.

Ces caractéristiques précitées forment un ensemble de paramètres prédominants pour la conception d'un protocole, qui va au-delà de la conception d'un protocole de routage pour la topologie semi-statique de la partie fixe d'internet [2].

1.3 Les contraintes liées aux réseaux ad hoc

Dans les réseaux MANET, les failles de liaison sans fil se produisent lorsque des nœuds de communication antérieurs se déplacent de telle sorte qu'ils ne sont plus dans la portée de transmission de l'autre. La formation de liaison sans fil se produit lorsque des nœuds qui ne sont pas dans la plage de communication se déplacent dans la plage de transmission l'un de l'autre. Les réseaux ad hoc sans fil héritent des problèmes traditionnels de la communication sans fil et de réseau sans fil (IEEE P802.11/D10, janvier 14,1999.), comme décrit ci-dessous :

- Le medium sans fil n'a ni limites absolues, ni facilement observable à l'extérieur desquelles les nœuds sont connus pour être incapable de recevoir des trames de réseau.
- Le canal n'est pas protégé par des signaux extérieurs.
- Le support sans fil est nettement moins fiable que le support filaire.
- Le canal a des propriétés de propagation temporelle et asymétrique.
- Des phénomènes terminaux cachés et des terminaux exposés peuvent se produire.
- Variation de la capacité de liaison d'un nœud.
- Modification dynamique de la topologie du réseau.

1.4 Le routage dans les réseaux mobiles ad hoc

On utilise des algorithmes de mise à jour des tables de routage spécifiques au réseau ad-hoc : la table de routage d'un nœud peut être modifiée très fréquemment si les nœuds sont très mobiles. Une station peut se trouver pendant un moment donné à la portée d'une station et ne plus l'être à un autre moment, sans aucun moyen de prévenir cette dernière. Si une route passait par l'intermédiaire de la station qui n'est plus accessible, il est nécessaire de retrouver une autre route pour accéder au destinataire [3]. On peut distinguer trois grande famille de protocole de routage : [4]

- **Routage proactifs :** un protocole proactif est un protocole qui construit les tables de routage avant que la demande en soit effectuée. Il identifie en fait à chaque instant la topologie du réseau.
- **Protocoles réactif :** un protocole réactif est un protocole qui construit une table de routage lorsqu'un nœud en effectue la demande. Il ne connait pas la topologie du réseau, il détermine le chemin à prendre pour accéder à un nœud du réseau lorsqu'on lui demande.
- **Protocoles hybrides :** les protocoles hybrides sont un mélange des deux premiers types. Un nœud agit d'une façon proactive dans une zone limitée à son voisinage jusqu'à n saut (n à préciser) et devient réactif au-delà de n sauts.

Dans la figure 1.3, le routage multi-hop dans un réseau ad-hoc est mis en évidence [35] :

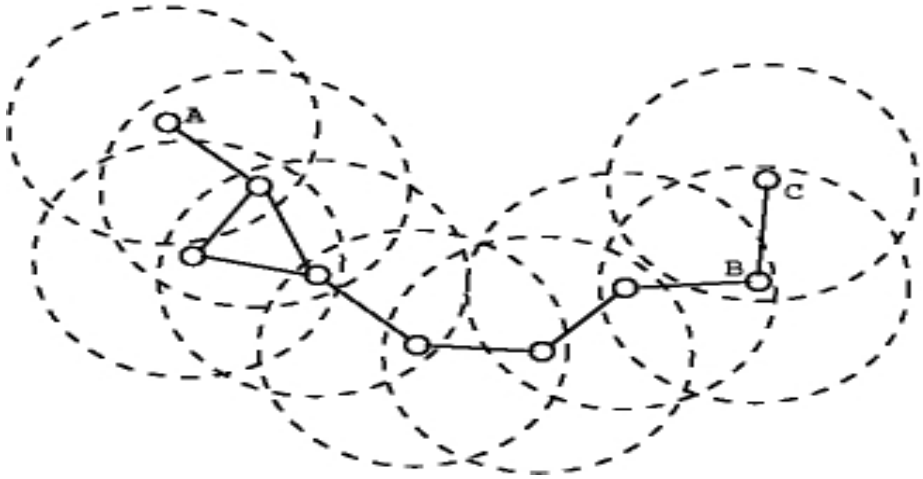

Figure 1.3 - Routage Multi-hop (hybride)

1.5 Application de MANET

MANET a de nombreuses applications, allant des réseaux mobiles de grande envergure et très dynamiques aux petits réseaux statiques qui sont contraints par les sources d'alimentation. Les domaines d'application de MANET incluent le secteur commercial, militaire, champ de bataille, environnements civils, opérations d'urgence et réseau personnel (PAN). Certaines des applications spécifiques sont mentionnées ci-dessous [5] :

- Conférence. Lorsque les utilisateurs d'ordinateurs mobiles se rassemblent au-delà de leur environnement normal, l'infrastructure du réseau d'affaires est souvent absente. Tout le point de la réunion pourrait être de faire des progrès supplémentaires sur un projet de collaboration particulier. Il s'avère que la création d'un réseau ad hoc pour les utilisateurs d'ordinateurs mobiles collaborateurs est nécessaire, même lorsque le support d'infrastructure Internet existe déjà.

- Réseau domestique. Considérez le scénario qui résultera si l'ordinateur sans fil devient populaire à la maison. Ces ordinateurs seront probablement pris pour et forment l'environnement de travail de bureau et sur les voyages d'affaires. De tels ordinateurs n'auraient pas d'adresses IP associées à chaque nœud sans fil à des fins d'identification ajouterait un fardeau administratif, et l'alternative de déploiement de réseaux ad hoc semble plus attrayante.
- Services d'urgence. Un réseau ad-hoc mobile peut également être utilisé pour fournir des applications de services de gestion de crise, comme dans le cas de reprise après sinistre, où toute l'infrastructure de communication est détruite et le recours rapide à la communication est crucial. En utilisant un réseau mobile ad-hoc, une infrastructure pourrait être configurée en heures au lieu de semaines, comme cela est requis dans le cas de la communication filaire.
- Application informatique embarquée. Certaines recherches prédisent un monde de l'informatique omniprésente [6], dans lequel les ordinateurs seront autour de nous, accomplissant constamment des tâches banales pour rendre notre vie un peu plus facile. Ces ordinateurs omniprésents réagiront souvent à l'environnement changeant dans lequel ils se situent et entraîneront eux-mêmes des changements dans l'environnement de façons qui sont, nous l'espérons, prévisibles et planifiées. Ces capacités peuvent être fournies avec ou sans l'utilisation de réseaux ad hoc, mais la mise en réseau ad hoc est susceptible d'être plus souple et plus commode que l'allocation continue et la réallocation de l'adresse IP d'extrémité chaque fois qu'une nouvelle liaison de communication sans fil est établie.

2. Systèmes répartis

2.1 Définition

Un certain nombre de définitions ont été proposées dans la littérature pour expliquer la signification des systèmes distribués. Un système distribué (réparti) est un réseau de communication, une collection d'ordinateurs indépendants qui apparaît à ses utilisateurs comme un seul système cohérent, et il peut même être un seul ordinateur multitâche [7]. Bien que les composants des systèmes distribués soient de nature autonome, ils peuvent avoir besoin de communiquer entre eux pour coordonner leurs actions et atteindre un niveau raisonnable de coopération [2]. Un programme composé d'instructions exécutables est exécuté par chaque ordinateur. Chaque exécution d'une instruction change le contenu de la mémoire locale de l'ordinateur, d'où l'état de l'ordinateur. Par conséquent, un système distribué est modélisé comme un ensemble de n machines d'état qui communiquent entre elles. Il s'agit principalement de deux

modèles de communication entre machines ; transfert de message et mémoire partagée. Dans le modèle de passage de messages, les machines communiquent entre elles en envoyant et en recevant des messages. Dans le modèle de mémoire partagée, les communications sont effectuées en écrivant et en lisant à partir de la mémoire partagée.

2.2 Différents systèmes répartis

Les systèmes répartis peuvent être distingues selon les hypothèses qu'ils admettent : nombre de sites connu ou non, topologie connue ou non, codes uniformes ou non, évolutions concertées des sites ou non. Au niveau de la communication on se demandera si le temps de délivrance d'un message est incertain ou bien fixe, si l'ordre des messages sur un lien est connu ou aléatoire. On distingue quatre paramètres essentiels : la topologie du réseau, la synchronisation des nœuds, les pannes tolérées et le type de communications. [8]

2.3 Les objectifs des systèmes répartis

- Transparences à la localisation : désignation. L'utilisateur ignore la situation géographique des ressources. Transparence à la migration.
- Transparence d'accès : l'utilisateur accède à une ressource locale ou distante d'une façon identique.
- Transparence aux pannes : les pannes et réincarnations sont cachées à l'utilisateur. Transparence à la réplication.
- Transparence à l'extension des ressources : extension ou réduction du système sans occasionner de gêne pour l'utilisateur (sauf performance).

2.4 L'architecture du réseau

La structure d'un réseau est une caractéristique importante. En effet, si nous considérons par exemple un problème *P*, plusieurs instances de solutions à ce problème *P* peuvent être proposées. Ces solutions sont en fonction de la nature du système considéré et des caractéristiques souhaitées pour la solution. Ces caractéristiques s'expriment, par exemple, par une complexité ou par un modèle de communication.

Un réseau de processus est modélisé par un graphe de connexion $G = (V, E)$. Ce graphe est défini tel que l'ensemble V des sommets du graphe est égal à $II = (p1,..,pn)$ et l'ensemble E des

arêtes le composant identifie les connexions inter processus. Les connexions entre processus n'étant pas forcément symétriques.

Un anneau est un réseau de communication ou un graphe sous-jacent qui sont des circuits simples. Les processus peuvent être unidirectionnelles ou bidirectionnelles

Dans un réseau complet, il y a un lien entre chaque deux nœuds ou processeurs. Si un des liens s'arrête, les autres liens peuvent être utilisés pour continuer la communication. Dans un réseau complet les paires de nœuds sont directement connectées.

2.5 Le modèle de défaillances

Les défaillances des processus se définissent, plus généralement, par un comportement du composant déviant de sa spécification originelle. Elles se constatent par la modification des exécutions dans le cas de défaillances logicielles, ou par un arrêt de l'exécution dans le cas de défaillances physiques.

Comme nous l'avons dit précédemment, les défaillances des composants ont des origines diverses. Nous pouvons distinguer plusieurs types de défaillances qui peuvent être persistantes ou temporaires [9].

2.5.1 Les défaillances des processus

Un site peut s'arrêter subitement de fonctionner, il peut oublier d'envoyer des messages, ou bien exhiber un comportement imprévu. Un site qui ne subit pas de défaillance est dit 'correct' sinon il est dit 'fautif'.

On désigne communément par la lettre *f* le nombre maximum de processus fautif au-delà duquel la tolérance aux défaillances n'est plus assurée. En effet, pour un problème *P* donné, *f* correspond au maximum de défaillances au-delà duquel la correction de la solution peut être mise en défaut. Une défaillance peut être un arrêt définitif, une omission, une faute de performance, ou une faute arbitraire. Une faute byzantine se caractérise par un comportement arbitraire, déviant de la spécification originelle du processus. Ce dernier type de comportement est, à l'inverse des fautes précédentes, une faute grave. Un processus byzantin peut, peut par exemple, corrompre des messages ou encore ne pas exécuter volontairement des parties entières de l'algorithme. Ce type de fautes caractérise principalement les comportements malveillants volontaires. L'indéterminisme sur les fautes possibles rend difficile, voire impossible, la résolution de certains problèmes [10].

2.5.2 Les défaillances des canaux de communication

Les défaillances sur les canaux de communication concernent principalement les messages et les liaisons correspondant aux canaux. Un canal peut arrêter le transit de message, les retarder en perdre, ou les corrompre.

Une classification des principaux types de canaux de communication que l'on retrouve dans la littérature est présentée comme suit [10] :

- **Les canaux fiables :** avec ce type de canal, si un processus *pi* envoie un message à un processus *pj* alors ce dernier reçoit ce message. Si un processus *pj* reçoit un message, il est délivré, et de plus il existe un processus *pi* qui l'a émis. Ainsi, l'intégrité du message est assurée.
- **Les canaux FIFO :** les canaux First In First Out sont des canaux fiables. Ils possèdent une propriété supplémentaire sur l'ordonnancement des messages. En effet, ce type de canal est tel que l'ordre de réception des messages suit l'ordre d'émission de ceux-ci.
- **Les canaux fiables avec fautes de performances :** ce sont des canaux fiables tels que les bornes sur les délais de transmission peuvent être mises en défaut. En effet, avec cette hypothèse, on n'assure aucunement la réception des messages en un temps borné.
- **Les canaux fiables avec pertes de messages :** ces canaux sont appelés canaux fair-lossy [30]. La propriété caractérisant ces canaux est la suivante : si un message est envoyé une infinité de fois alors, il est reçue une infinité de fois.
- **Les canaux non fiables :** ces canaux sont caractérisés par une perte non contrôlée des messages y transitant.

2.6 Les détecteurs de défaillances

Les détecteurs de défaillances de Chandra et Toueg [31] peuvent être considérés comme des briques de base des systèmes distribués tolérant les fautes. Un détecteur de défaillances peut être vu comme un ensemble de modules. A chaque processus est associé un module (voir figure 1.7). Ces modules fournissent aux processus du système une liste de processus suspectés d'être défaillant. Cette liste peut être différente d'un processus à l'autre [9].

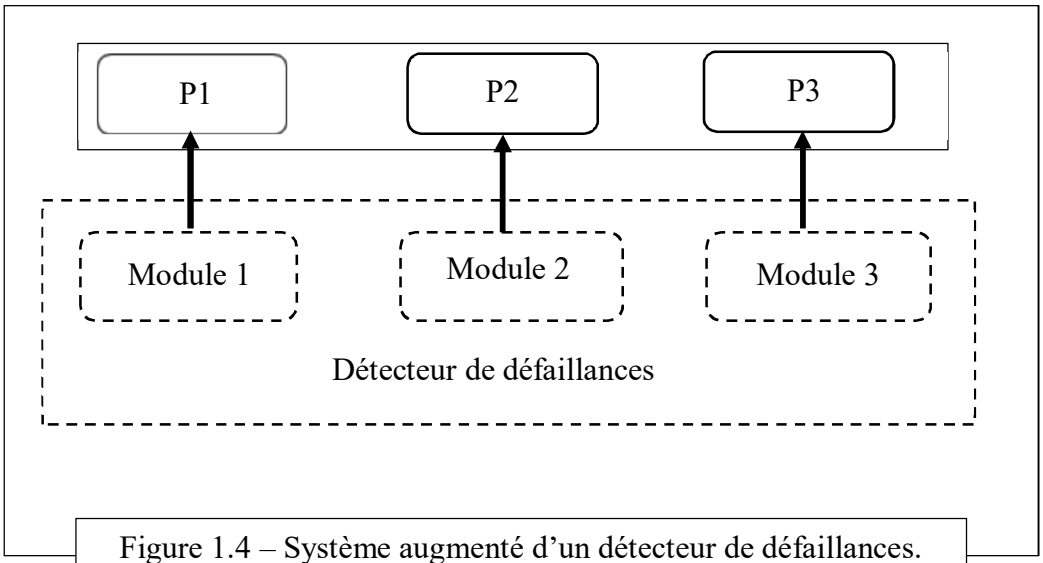

Figure 1.4 – Système augmenté d'un détecteur de défaillances.

La mise en œuvre du mécanisme de détection se fait en définissant des délais de garde : un processus est suspecté s'il ne s'est pas manifesté avant l'expiration de délai de garde. Cela signifie qu'un détecteur de défaillances peut commettre des erreurs :

- Ne pas suspecter un processus alors qu'il est défaillant.
- Suspecter un processus alors qu'il est correct.

2.7 Propriétés d'un détecteur de défaillances

Un détecteur de défaillances est défini par deux propriétés suivantes : la complétude et la précision, la complétude est un ensemble de contraintes concernant la détection des processus suspectés par tous les processus corrects ainsi on peut distinguer deux cas [11] :

- La complétude forte : les processus fautifs sont finalement et définitivement suspectés par tous les processus corrects.
- La complétude faible : pour tous processus fautif, il existe au moins un processus correct qui le suspecte.

La précision quant à elle, donne le degré de fiabilité dans le non suspicion des processus corrects. Idéalement, seuls les processus défaillants doivent être détectés comme tels. Là encore, la précision peut être forte ou faible, mais aussi ultimement forte ou ultimement faible.

- La précision forte : les processus corrects ne sont suspectés à aucun moment.
- La précision faible : il existe au moins un processus correct qui n'est jamais suspecté.
- La précision ultime forte : il existe un moment à partir duquel aucun processus correct n'est plus suspecté.

- La précision ultime faible : il existe un moment à partir duquel il existe un processus correct qui n'est plus jamais suspecté.

2.8 Exemple des systèmes répartis

Parmi plusieurs exemples de systèmes répartis, on distingue le web, un serveur web auquel se connecte un nombre de client à travers des navigateurs web, ce dernier garantie l'accès à distance d'informations, et cela de manière simple, un client envoie une requête demandant une page web, le serveur renvoie la page html statique qu'il a stocké localement ensuit il interroge une base de données pour générer dynamiquement le contenu de la page html et tous cela est retransmis au client. La figure 1.8 permet de schématiser le processus décrit ci-dessus [12] :

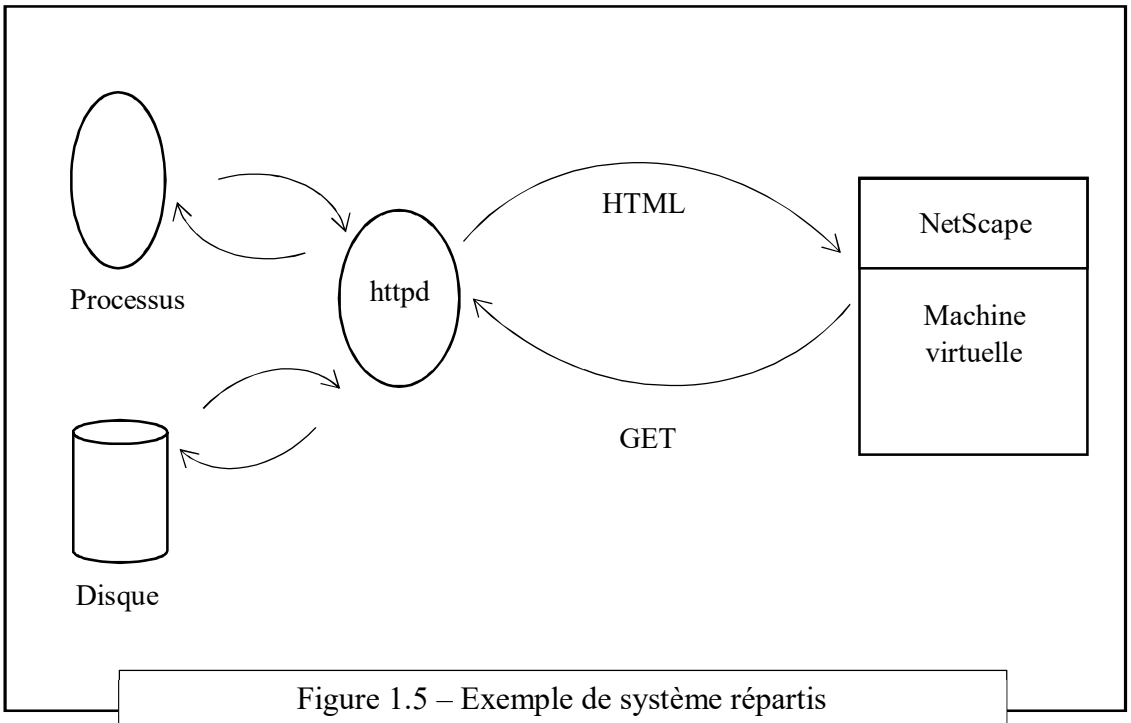

Figure 1.5 – Exemple de système répartis

Conclusion

Dans ce chapitre, nous avons donné un aperçu général sur les environnements mobiles qui sont caractérisés par de fréquentes déconnexions et des restrictions sur les ressources utilisées. Ensuite, nous avons présenté les environnements mobiles ad hoc avec leurs caractéristiques qui compliquent le problème de routage dont les stratégies (proactives ou réactives) doivent tenir compte des changements fréquents de la topologie, de la consommation de la bande passante qui est limitée, ainsi que d'autres facteurs. Les limitations imposées par l'environnement mobiles ad hoc, nous obligent à changer la vision classique des problèmes liés aux systèmes distribués. Parmi ces problèmes, celui de l'élection d'un leader qui devient plus compliquée à résoudre dans des environnements mobiles ad hoc.

Le problème d'élection de leader

Introduction

Dans un système réparti, beaucoup d'applications exigent qu'un seul processus soit désigné pour coordonner une tâche effectuée par un ensemble d'autres processus. Généralement, beaucoup de processus dans le système sont potentiellement capables d'offrir une telle coordination. Cependant, et en raison de consistance, seulement un processus est en mesure d'offrir vraiment cette fonction. Par conséquent, un processus appelé le leader, doit être choisi pour assurer cette fonction. Un leader dans un système réparti est un processus du système. En plus, si le processus élu tombe en panne, un nouveau leader doit être élu.

En informatique distribué, l'élection de leader est une primitive importante. Il est utile pour de nombreuses applications qui nécessitent la sélection d'un processeur unique parmi plusieurs processeurs, en particulier lorsque les défaillances sont fréquentes.

Il existe de nombreuses applications pour les algorithmes d'élection de leader ; Généralement utilisé comme primitif dans d'autres algorithmes distribués. Le problème d'élection de leader pour les réseaux statiques est : chaque réseau doit éventuellement avoir un leader unique.

Cependant, dans les réseaux dynamiques en raison de la mobilité des nœuds et des échecs de liaisons, les partitions peuvent se produire et le leader ne sera pas élu avant qu'un partitionnement ne soit détecté. Et parfois, deux composants de réseau peuvent fusionner et temporairement il peut y avoir deux leaders dans le réseau nouvellement formé ou il peut y avoir une période où il n'y a pas de leader dans le composant. Ainsi, la définition du problème d'élection d'un leader dans les réseaux ad hoc mobiles devrait être légèrement modifiée : chaque composant connecté aura éventuellement un leader unique, même après tout nombre de défauts simultanés.

1. Définition

Le problème d'élection constitue une brique de base utile dans les systèmes distribués, particulièrement ceux sujets à des défaillances. A titre d'exemple, si la défaillance d'un nœud provoque la perte d'un jeton dans un algorithme d'exclusion mutuelle, les autres nœuds doivent élire un nouveau 'leader' qui sera chargé de régénérer le jeton. L'élection d'un leader est aussi utile dans les protocoles de communication de groupe afin de choisir un nouveau coordinateur lorsque la composition du groupe change.

Un algorithme d'élection de leader valide doit remplir les conditions suivantes [13] :

- **Terminaison :** l'algorithme devrait se terminer dans un temps fini une fois le leader est sélectionner. Dans les approches randomisées cette condition est parfois affaibli (par exemple, ce qui nécessite une terminaison avec probabilité).
- **Unicité :** il y a exactement un processeur qui se considère comme leader.
- **Accord :** tous les autres processeurs savent qui est le leader.

Un algorithme pour l'élection de leader peut varier dans les aspects suivants [14] :

- **Mécanisme de communication :** les processeurs sont soit synchrones dans lequel les processus sont synchronisés par un signal d'horloge ou asynchrone où les processus fonctionnent à des vitesses arbitraires.
- **Les noms des processus :** si les processus ont une identité unique ou sont impossibles à distinguer (anonyme).
- **La topologie du réseau**
- **Taille de réseau :** l'algorithme ne peut pas utiliser la connaissance du nombre de processus dans le système.

2. Algorithmes d'élection de leader dans les systèmes répartis

2.1 Election de leader en anneaux

Un réseau en anneau est une topologie-graphe connexe dans lequel chaque nœud est connecté à exactement deux autres nœuds, soit pour un graphe avec n nœuds, il y a exactement n arêtes reliant les nœuds. Un anneau peut être unidirectionnel, ce qui signifie que les processeurs communiquent uniquement dans une direction (un nœud ne peut envoyer des messages vers la

gauche ou envoyer uniquement des messages vers la droite), ou peut être un anneau bidirectionnel, ce qui signifie qu'un nœud peut transmettre et recevoir des messages dans les deux sens (un nœud pourrait envoyer des messages à gauche et à droite). Pour chaque type d'anneau (unidirectionnel/bidirectionnel), il existe des algorithmes d'élection adéquat :

2.1.1 Anneaux unidirectionnels

Le premier algorithme d'élection pour les anneaux unidirectionnels a été proposé par LeLann [15]. Il exige que chaque processus doit avoir une identité unique, avec un ordre total sur les identités ; le processus avec la plus grande identité devient le leader. L'idée fondamental de l'algorithme de LeLann est que chaque processus envoie autour de l'anneau un message portant son identité. Ainsi il nécessite n^2 messages pour une élection, n étant le nombre de processus dans l'anneau. La figure 2.1 présente un anneau unidectionnel.

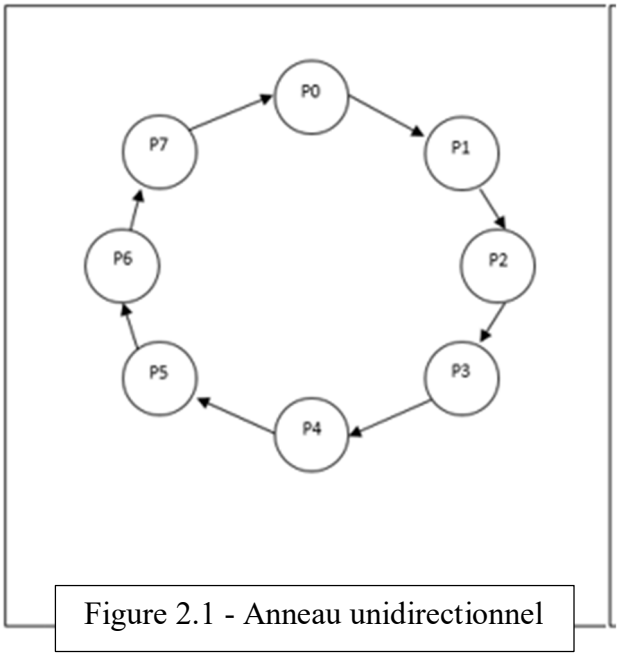

Figure 2.1 - Anneau unidirectionnel

2.1.2 Anneaux bidirectionnels

L'algorithme de Hirschberg et Sinclair [16] échange *O (n log n)* message dans le pire des cas, mais utilise une communication bidirectionnelle. L'algorithme bidirectionnel de Burns [17] nécessitant *O (n log n)* messages est légèrement meilleur. En plus, Burns a établi une borne inferieure de Ω *(n log n)* sur le nombre des messages exigés uniquement si la communication est asynchrone. Franklin [18], a développé un algorithme avec une complexité de *O (n log n)* message dans les pires des cas, pour les anneaux bidirectionnels. Peterson [19] et Dolev, Klawe, et Rodeh [20], ont indépendamment adapté l'algorithme de Franklin de telle sorte qu'il fonctionne

également pour les anneaux unidirectionnels. Tous les algorithmes précédents fonctionnent pour les deux modes de communication, asynchrone et synchrone. Et n'exigent pas de connaitre, à priori, le nombre de processus. La figure 2.2 présente anneau bidirectionnel.

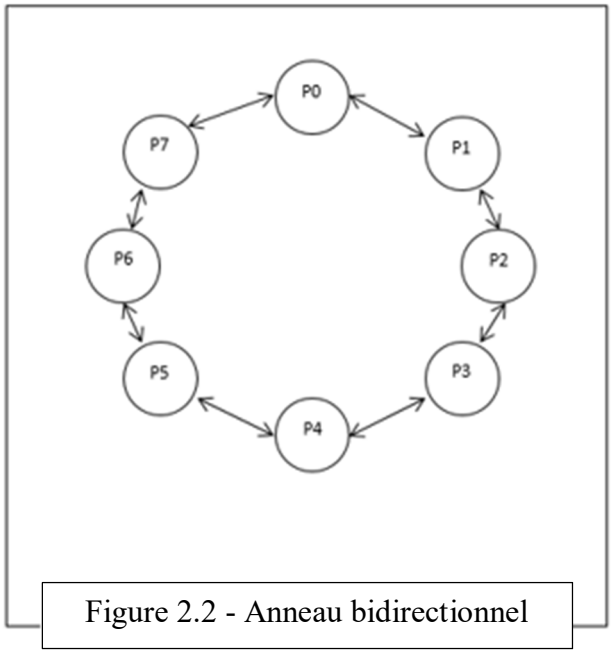

Figure 2.2 - Anneau bidirectionnel

2.2 Election de leader dans les réseaux complets

Ce sont des structures dans lesquelles tous les processus sont reliés les uns les autres, à savoir, le degré de chaque nœud est un groupe *n-1*, n étant la taille du réseau. Une solution optimale avec le message d'*O(n)* et de la complexité de l'espace est connu [21]. Dans cet algorithme, les processus ont les états suivants :

- Dummy : les nœuds qui ne participent pas l'algorithme d'élection de leader.
- Passif : l'état initial des processus avant le début.
- Candidat : l'état des nœuds après le réveil. Les nœuds candidats seront considérés pour devenir le leader.

Pour élire un leader, un anneau virtuel est considéré dans le réseau. Tous les processeurs commencent d'abord par un état passif jusqu'à ce qu'ils soient réveillés. Une fois que les nœuds sont éveillés, ils sont candidats pour devenir le leader. Sur la base d'un schéma de priorité, les nœuds candidat collaborent dans l'anneau virtuel. A un certain moment, les candidats prennent conscience de l'identité des candidats qui les précèdent dans l'anneau. Les candidats de priorité plus élevée demandent les plus bas au sujet de leurs prédécesseurs. Les candidats ayant une priorité inferieur deviennent nul après avoir répondu aux candidats avec une priorité plus élevée. Sur la

base de cet algorithme, le candidat avec la priorité la plus élevée sait finalement que tous les nœuds du système sont nul sauf lui-même, ce qui lui indique qu'il est le leader.

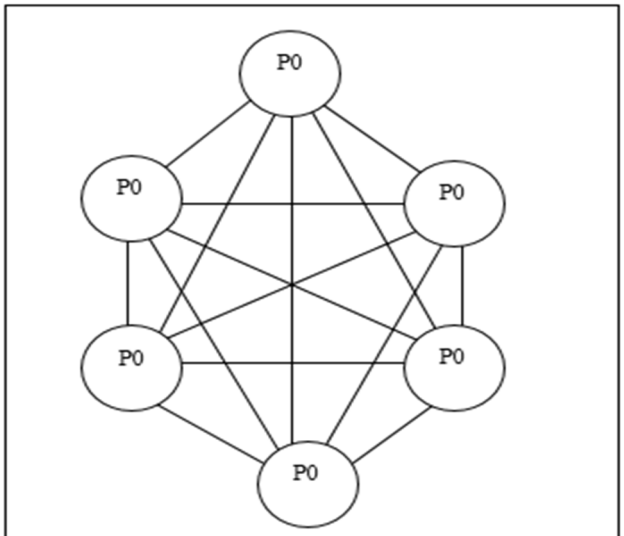

Figure 2.3 – Réseau complet de 6 processus

2.3 Algorithmes universel d'élection de leader

Comme leurs noms l'indique, ces algorithmes sont conçus pour être utilisés dans toute les formes de réseau de processus sans aucune connaissance préalable de la topologie d'un réseau ou ses propriétés telle que sa taille [22].

Dans ce qui suit nous avons présenté quelques algorithmes universels [23] :

- Shout : ce protocole construit un arbre couvrant sur un graphique générique et choisit sa racine comme leader.
- Méga-fusion : cette technique est essentiellement similaire à la recherche d'un arbre d'extension minimum (MST) dans lequel la racine de l'arbre devient le leader. L'idée de base de cette méthode et des nœuds individuels se confondent les uns des autres pour former des structures plus grandes. Le résultat de cet algorithme est un arbre (un graphe sans cycle) dont la racine est le leader du système. Le coût de la méthode méga-fusion est $O(m + n \log n)$ où m est le nombre d'arêtes et n le nombre de nœuds.

3. Les protocoles d'élection dans les réseaux ad hoc

3.1 Protocole de HATZIS et al.

Hatzis et al [24] ont traité le problème d'élection dans les réseaux ad hoc. Leurs protocoles sont divisés en deux classes, les protocoles forcés, qui déterminent le mouvement de certains ou de tous les hôtes en exigeant qu'ils effectuent certain mouvement afin d'assurer l'exécution correcte du protocole. Les protocoles non forcés, n'affectent pas le mouvement des hôtes.

3.1.1 Le protocole non forcé

Le protocole non forcé tire profit du mouvement des hôtes mobiles en échangeant l'information à chaque fois qu'ils se rencontrent incidemment. Dans le cas contraire, le protocole peut ne jamais élire un leader unique, ce protocole n'inclut aucun mécanisme de détection de terminaison.

3.1.2 Protocole forcé de Las Vegas sans sens d'orientation

Les protocoles forcés, forcent les hôtes mobiles à se déplacer selon un procédé bien défini afin de satisfaire les demandes du protocole. On peut distinguer trois cas selon de l'orientation des hôtes mobiles :

- Les hôtes mobiles n'ont aucun sens d'orientation.
- Chaque hôte mobile a un sens individuel d'orientation.
- Les hôtes mobiles ont un sens d'orientation commun.

Vue la faiblesse importante du protocole non forcé qui est le fait de ne pas pouvoir élire un unique leader, tous nous a mené au protocole forcé de Las Vegas qui permet d'élire un leader unique dans les réseaux mobiles ad hoc dans un temps linaire à n.

Ce protocole se comporte comme le précèdent, et ce qui le rend plus efficace, c'est qu'il oblige les hôtes mobiles à exécuter un mouvement aléatoire, et inclut aussi un mécanisme de détection de terminaison. L'hôte peut décider sur une probabilité de défaillance pf et l'employer afin de trouver l'instant t nécessaire pour exécuter le protocole. Si l'hôte après l'instant t est toujours dans l'état participant, il est l'unique leader.

Le protocole forcé de Las Vegas, peut être appliqué même dans le cas où le réseau mobile est anonyme, ce qui veut dire que les hôtes mobiles n'ont aucune identité distincte.

Pour permettre l'utilisation du protocole dans les réseaux mobiles anonyme, les variantes suivantes ont été proposés comme suit :

A chaque fois que deux hôtes mobiles se rencontrent, ils choisissent un identifiant aléatoire à partir d'un ensemble prédéfini et l'échange aura lieu. Si un conflit survient, le procédé doit être répété. Le gagnant est automatiquement celui qui a l'identité la plus élevée.

3.2 L'algorithme de Vasudevan, Kurose, et Towsely

L'objectif de cet algorithme d'élection est d'assurer qu'après un nombre fini de changements topologiques, chaque nœud *i* a un leader qui est le nœud ayant la valeur la plus élevé, et cela parmi tous les nœuds dans la composante connexe. [25]

Cet algorithme d'élection de leader est basé sur l'algorithme de détection de terminaison pour les calculs de diffusion de Dijkstra et Scholten. Son idée est de 'grandir' et 'rétrécir' un arbre couvrant enraciné au nœud qui lance l'algorithme d'élection. Ce dernier est appelé nœud source.

Après le rétrécissement de l'arbre couvrant, le nœud source obtient l'information nécessaire pour déterminer le nœud ayant la valeur la plus grande, puis diffuse son identité aux autres nœuds qui sont appelé fils. Il est supposé que la valeur du nœud est identique à son identificateur.

Pour le bon fonctionnement de l'algorithme, il doit y avoir impérativement cinq types de messages : *Election, ACK, leader, Prob et Replay*.

3.3 Protocoles basé sur TORA

Les algorithmes proposés par Malpani, Welch, Vaidya [26], sont des algorithmes qui ont comme idée de base le protocole de TORA, mais avec des modifications qui sont apportées à l'algorithme :

- Chaque composante du réseau doit former un DAG, au lieu d'avoir un DAG orienté.
- En utilisant le mécanisme de TORA, un nouveau leader est élu et son identificateur est propagé à travers toute la nouvelle composante et cela lors de la détection d'une partition.
- Quand deux composantes se fusionnent, une comparaison aura lieu entre les deux leaders de tel sorte que l'identificateur du gagnant soit diffusée et écrase l'identificateur du perdant.
- Des complications surgissent quand plusieurs changements de topologie se produisent. Et cela est dû au fait que durant la diffusion de l'identificateur du nouveau leader, des changements de topologie pourraient se produire dans la composante et le processus

d'élection sera peut-être répété. Pour cela Malpani, Welch, Vaidya ont proposé un algorithme pour chaqu'un des cas suivant :

3.3.1 Algorithme d'élection pour un seul changement de topologie

Malpani, walch, et Vaidya ont apportés les changements suivant au protocole TORA. La structure de chaque nœud i dans l'algorithme est 6-tuple ; *(lidi, ti, oidi, ri, δi, i)*. La première composante est l'identificateur du nœud supposé être leader de la composante de i. les cinq composantes restantes sont les même que dans TORA. Le niveau de référence *(-1,-1,-1)* est employé par le leader d'une composante pour assurer qu'il soit le seul nœud sans successeur.

Dans l'algorithme de TORA, le premier nœud qui détecte une partition envoie des signalisations aux autres nœuds dans sa composante pour qu'ils cessent d'effectuer des changements de taille et d'envoyer des messages inutiles. Dans l'algorithme de MWV (Malpani, Welch et Vaidya), le nœud qui a détecté la partition s'élit en tant que leader de la nouvelle composante. Il communique alors cette information à ces voisins, qui propagent à leur tour cette information à leurs voisins et ainsi de suite. Par la suite, tous les nœuds dans la nouvelle composante se rendent compte du changement du leader. Quand deux composantes fusionnent à cause de la formation de nouveau liens, le leader de la composante dont l'identificateur est le plus petit devient par la suite l'unique leader de la nouvelle composante entière.

On va maintenant décrire le code exécuté par le nœud *i*. Chaque étape est déclenchée par la signalisation d'une défaillance ou de la formation d'un lien incident (incident link) ou par la réception d'un message envoyé par un voisin. Le nœud *i* stocke les identificateurs de ses voisins dans sa variable locale *Ni.* Quand un lien incident devient défaillant, *i* met à jour *Ni*. Quand un lien incident est formé, *i* met à jour une structure de données locale qui maintient la taille courante rapportée pour chacun de ses voisins.

A la fin de chaque étape, si la structure de *i* a été changée, i envoie un message UPD avec la nouvelle taille à tous ses voisins. Le pseudo code ci-dessous explique comment et quand la taille du nœud *i* va être changée. Les parties de *B* et *D* sont exécutées seulement si l'identificateur de leader dans le message UPD reçu est identique à *lidi*.

A. Quand un nœud *i* n'a aucun lien sortant à cause d'une défaillance de lien :

```
1. Si le nœud i n'a aucun lien entrant alors :
2.     Lidi := i
3.    (ti, oidi, ri) := (-1,-1,-1)
4.     Tetai : = 0
5. Sinon
6.     (ti,oidi,ri) := (t,i,0) // t est le temps courant
7.     Tetai := 0
```

B. Quand le nœud *i* n'a aucun lien sortant à cause d'une inversion de lien suite à la réception d'un message UPD, et les niveaux de références *(tj, iodj, rj)* ne sont pas égaux pour tout *j* appartenant à *Ni* :

```
1. (ti, oidi, ri) := max {(tj, oidj, rj) | j appartient à Ni}
2. Tetai := min {tetaj | j appartient à Ni et (tj, oidj, rj) = (ti,
   oidi, ri)}-1
```

C. Quand le nœud i n'a aucun lien sortant à cause d'une inversion de lien suite à la réception d'un message UPD, et les niveaux de références (*tj, oidj, rj)* sont égaux avec *rj=0* pour tout *j* appartenant à *Ni* :

```
1. (ti, oidi, ri) := (tj, oidj, 1) pour tout j appartenant à Ni
2. Tetai := 0
```

D. Quand le nœud *i* n'a aucun lien sortant à cause d'une inversion de lien suite à la réception d'un message UPD, et les niveaux de références *(tj, oidj, rj)* sont égaux avec *rj = 1* pour tout *j* appartenant à *Ni* :

```
1. Lidi := i
2. (ti, oidi, ri) := (-1,-1,-1)
3. Tetai := 0
```

E. Quand le nœud *i* reçoit un message UPD d'un nœud voisin *j* tel que *lidi* n'est pas égale à *lidj* :

```
1. Si lidi > lidj ou (oidi = lidj et ri = 1) alors
2.    Lidi := lidj
3.    (ti, oidi, ri) := (0,0,0)
4.    Tetai := tetaj + 1
```

Dans le point *E*, si le nouvel identificateur est plus petit que l'autre, il sera adopté. Si le nouvel identificateur est plus grand que le vôtre, alors il sera adopté, mais seulement si c'est le cas où le créateur d'un nouveau niveau de référence a détecté une partition et il s'est élu.

3.3.2 Algorithme d'élection pour plusieurs changements topologiques simultanés

Malpani, Welch et Vaidya ont proposé un algorithme d'élection pour plusieurs changements topologiques simultanés. Les changements de topologie simultanés signifient qu'un nouveau changement (défaillance ou formation de lien) apparait avant la reconfiguration du réseau. Le dernier point de l'algorithme précèdent est remplacé par le code ci-dessous et un autre point sera introduit.

F. Quand le nœud *i* reçoit un message UPD (UPADTE) d'un nœud voisin *j* tel que lidi n'est pas égale à *lidj* :

```
1.  si lidi > lidj ou (oidi = lidj et ri = 1) alors
2.      lidi := lidj
3.    si lidj = j then
4.      (ti, oidi, ri) := (0,0,0)
5.    Sinon
6.      (ti,oidi,ri) := (tj,oidj,rj)
7.       Tetai := tetaj + 1
8.  Sinon si lidi < lidj et ce n'est pas la première mise à jour reçue à partir de j
    depuis la formation du lien :
9.  Si lidi = i alors
10.   Heighti[j] := (lidi ,0,0,0,tetaj + 1,i)
11. Sinon
12.   Heighti[j] := (lidi, ti,oidi,ri,tetai+1,i)
13.   Envoyer à j un message UPD qui contient la taille de i. // vérifier si i devient
    commencer un nouveau niveau de référence
14. Si le nœud i n'a aucun lien sortant et lidi n'est pas égale à i et lidk = lidi pour
    tout k appartenant à Ni alors
15. (ti,oidi,ri) := (t,i,0) // t est le temps courant
16. Tetai := 0
```

G. Quand le nœud *i* n'a aucun lien sortant à cause d'une inversion de lien suite à la réception d'un message UPD (UPDATE), et les niveaux de références *(tj, oidj, rj)* sont égaux avec *rj = 1* et *oidj* n'est pas égale à *i*, et pour tout j appartenant à *Ni* :

```
1.  Lidi := i
2.  (ti,oidi,ri) := (t,i,0) // t est le temps courant
3.  Tetai := 0
```

Les lignes 3-6 du point *E*, sont nécessaire pour manipuler le cas ou deux composantes *A* et *B* fusionnent. Un exemple pour ce cas est montré par la figure 2.1 supposons que le nœud *i* de la composante *A* n'a pas encore reçu le message de mise à jour concernant le nouveau leader de la composante *A*, mais il a un niveau référence reflété. Le nœud *i* reçoit un message de mise à jour du nœud *j* de la composante *B*. Si *lidj* > *lidi* et *lidj* n'est pas égale à *j*, alors le niveau de référence de *j* est remplacé par le niveau de référence de *i*. Ceci permet au nœud *i* de propager le message de mise à jour concernant le nouveau leader de la composante A dans la composante de B entière.

La figure 2.1 est un exemple de fonctionnement d'algorithme proposé par Malpani, Welch, et Vaidya.

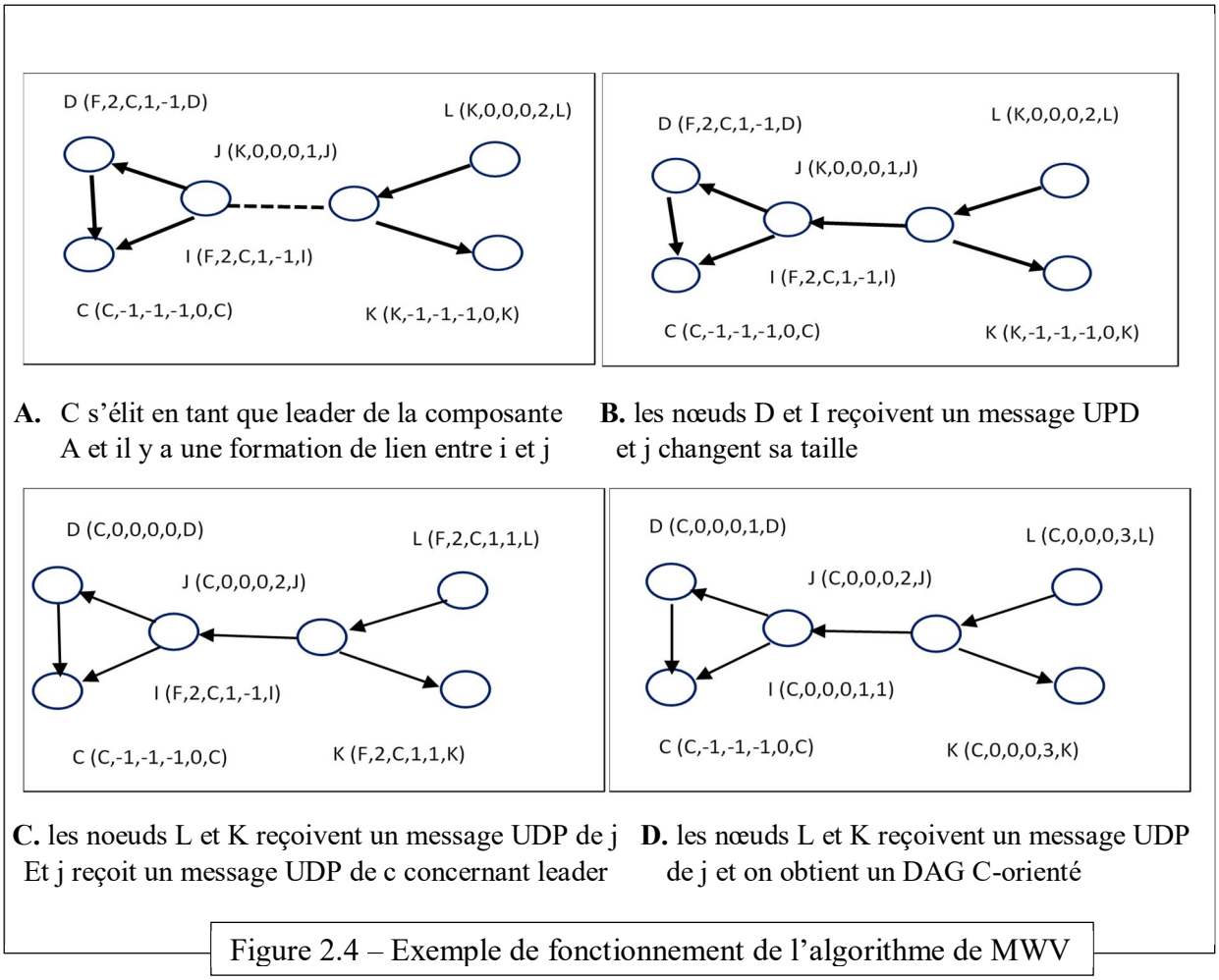

Figure 2.4 – Exemple de fonctionnement de l'algorithme de MWV

3.3.3 Un protocole d'élection pour les changements topologiques uniques et simultanés

Velayutham et Chaudhuri [27] ont proposé une version simple et modifiée des algorithmes de MWV et ils ont fourni une preuve de bon fonctionnement dans le cas des changements

topologiques simultanés. Leur algorithme couvre les deux types de changements, unique et simultanés.

Les trois premiers points *A*, *B* et *C* des algorithmes de MWV restent le mêmes dans cet algorithme, mais le quatrième et le cinquième point sont modifiés. Dans le point *D*, quand un nœud détecte une partition, il s'élit comme étant le leader de la composante. Ceci est différent de l'algorithme MWV où seulement l'initiateur peut s'élire comme étant le leader et les autres nœuds peuvent uniquement lancer un nouveau niveau de référence.

D. Quand le nœud *i* n'a aucun lien sortant à cause d'une inversion de lien suite à la réception d'un message UPD (UPDATE), et les niveaux de références *(tj,oidj,rj)* sont égaux avec *rj = 1* pour tout *j* appartenant à *Ni*, le nœud *i* s'élit en tant que leader comme suit :

```
1. Lidi := i
2. (ti,oidi,ri) := (-1,-1,-1)
3. Tetai := 0
```

E. Quand le nœud *i* reçoit un message UPD (UPDATE) d'un nœud voisin *j* tel que *lidi* n'est pas égale à *lidj* :

```
1. Si lidi > lidj ou (oidi = lidj) alors
2.    Lidi := lidj
3.   (ti,oidi,ri) := (0,0,0)
4.    Tetai := tetaj + 1
```

L'initiateur de ces trois algorithmes peut être réalisé en commençant chaque nœud comme étant le leader de sa propre composante, c.à.d., chaque nœud *i* commence avec sa taille égale à *(i, -1,-1,-1,0, i)* et sa liste de voisins *Ni* vide.

Pour que ces algorithmes soient tolérants aux défaillances de nœud, il est supposé que lorsqu'un nœud se remet d'une défaillance de nœud, il recommence en se déclarant en tant que leader, c.-à-d., plaçant sa taille à *(i,-1,-1,-1,0,i)*.

Conclusion

Le problème d'élection qui constitue une brique de base utile dans les systèmes distribués, a été intensivement étudié sur une grande variété de modèles et de topologies. En fait, les solutions proposées différent selon l'identité des processus, la topologie du réseau (anneau, graphe complet, topologie arbitraire, arbre, etc.), et même par la manière de résoudre le problème.

Dans ce chapitre nous avons présenté différents algorithmes d'élection dans les réseaux mobiles ad-hoc. Certains protocoles comme ceux proposés par Malpani, Welch et Vaidya sont une modification du protocole de routage TORA, L'algorithme proposé par Vasudevan, Kurose, et Towsely, qui repose sur la détection de terminaison pour les calculs de diffusion de Dijkstra et Scholten s'avère le plus intéressant à implémenter et cela en raison du nombre de message faible qu'il peut générer lors du processus d'élection.

Dans le chapitre suivant, nous détaillons l'algorithme de Vasudevan, Kurose, Towsley, c'est celui que nous avons choisi pour l'étude détaillé dans notre mémoire et une petite comparaison sera faite dans la partie implémentation avec les protocoles de Hatzis et al.

Etude de l'algorithme de Vasudevan, Kurose, Towsely

Introduction

L'algorithme présenté ci-dessous utilise le concept de diffusion, pour effectuer l'élection de leader. Il procède comme suit : lorsqu'une élection est déclenchée sur un nœud, le nœud commence à diffuser le calcul pour déterminer son nouveau leader. Plusieurs nœuds peuvent commencer à diffuser des calculs en réponse au départ d'un leader et, par conséquent, plusieurs calculs diffusants peuvent être en cours simultanément ; cependant, un nœud participe à un seul calcul de diffusion à la fois. Eventuellement, lorsqu'un calcul diffusant se termine, le nœud qui lance le calcul informe d'autre nœuds de l'identité du leader élu. Une élection peut être déclenchée par un nœud à cause de la déconnexion de leader ou à cause de la valeur du leader en dessous d'un seuil défini par l'application.

1. Construction d'un arbre couvrant (diffusion)

Le site *I* ne reçoit pas de message

If *leader* ***and*** *parent =* ***null then***
Envoyer *m aux voisin ; parent = I ;*

Le site *I* reçoit le message *m* depuis le site *j*

If *parent =* ***null then***
Parent = j
Envoyer *<parent> à j*
Envoyer *M aux voisin k, tel que k ≠ j*
Else envoyer *<rejet>* ***to*** *j*

Le site *I* reçoit <parent> depuis site j

Children = children ∪ {j}
If *children ∪ others = voisins \ {parent}* ***then fin***

Le site *I* reçoit <rejet> depuis le site *j*

Other = other ∪ {j}
If *children ∪ other = voisins \ {parent}* ***then fin***

La figur3.1 nous montre une construction d'un arbre couvrant sur une topologie.

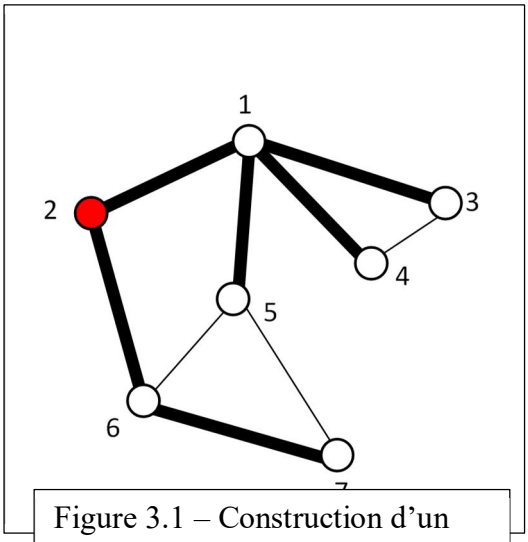

Figure 3.1 – Construction d'un arbre couvrant sur une topologie

1.1 Definition d'un arbre

Un graphe non orienté est un arbre s'il est connexe sans cycle et si un sommet particulier, la racine, a été distinguer [28].

Un arbre enraciné est souvent muni d'une orientation naturelle :

On oriente chaque arrête de telle sorte qu'il existe un chemin de la racine à tous les autres sommets. Le graphe orienté résultant est aussi appelé arbre enraciné ou plus simplement arbre.

Un arbre couvrant, est un sous-graphe de G (soit G un graphe orienté non connexe), connexe et sans-cycle. Son poids est la somme des évaluations de ses arêtes.

2. Terminaison pour le calcul d'arbre

2.1 Principe généraux

- Le graphe de calcul est en forme d'arbre et l'initiateur c'est la racine,
- L'algorithme de contrôle s'exécute en parallèle avec l'algorithme de calcul (base) mais sans interférence (messages de contrôle),
- On considère que les liens réseaux sont bidirectionnels (ils sont indépendants du graph de l'application).

2.2 Algorithme de Dijkstra et Scholten

Dans ce qui suit nous allons présenter les différentes étapes de fonctionnement que constitue l'algorithme de Djikstra et Scholten [29] :

2.3 Principe de fonctionnement

- Solution centralisée : il y a un seul initiateur, pour l'algorithme de base et pour l'algorithme de contrôle,
- L'algorithme gère un arbre dynamique *T (Vt, Et)*
  - Les nœuds sont des processus actifs ou des messages en transit.
  - Les arrêtes reliant un nœud et celui qui lui a donné naissance (activation d'un processus suite à la réception d'un message ou le message lui-même).
  - L'arbre possède les propriétés suivantes :

  -$T = \emptyset$ ou bien T est un arbre orienté de racine p_0.

  -V_T contient tous les processus actifs et les messages en transit.

- p_0 appelle l'algorithme de l'annonce() de la terminaison dès que $p_0 \notin V_T$ ou $T = \emptyset$.

2.3.1 Ajout des éléments dans l'arbre

- Les évènements qui conduisent à créer un nœud sont :
  - *P* envoie le message <*msg*>, le nœud <*msg*> est créé dans l'arbre, le père de <*msg*> est *p*.
  - Si $p \notin T$, p devient actif à la réception d'un message de q, si le nœud p est créé avec *q* le père de *p*.
- On représente un message avec l'émetteur par le couple <*msg, q*> qui est l'émetteur du message <*msg*>.
- Les messages sont des feuilles alors que les processus sont soit des feuilles soit des nœuds internes de *T*.
- Les nœuds de l'arbre maintiennent le compte de leurs fils, soit des messages, soit des autres processus.

La procédure suivante montre le pseudo code d'ajout des éléments dans l'arbre :

```
Sp : {etatp = actif}
Debut
        Send <msg, p>
        SCp++
Fin
```

2.3.2 Suppression des éléments dans l'arbre

- Les événements qui conduisent à supprimer des éléments sont :
  - Acquittement d'un message, le message en transit était dans l'arbre. Il est supprimé dès lors qu'il est consommé par son destinataire : envoi d'un message de contrôle,
  - Passage d'un processus de l'état **actif** à l'état **passif** : envoi d'un message de contrôle,
- Chaque processus compte le nombre de messages et le nombre d'instances dans le sous-arbre dont il est la racine.
- L'arbre se résorbe en un nombre fini d'étapes après la terminaison.

La procédure suivante montre le pseudo code de suppression des éléments dans l'arbre :

```
Ip : {etatp = actif}
debut
        etatp := passif
        si (SCp = 0) alors
                si perep = p alors annonce()
                sinon send <sig, perep> à perep
                perep := nDef
fin
```

2.3.3 Effacement d'un fils du nœud p

Par défaut on considère que le premier message créé le processus fils et les suivants sont nécessaire au calcul. Tous les messages ne conduisent donc pas à l'activation d'un processus, d'où les 2 cas suivants :

- Un message pour prévenir l'initiateur de l'arrivée de son message lorsque le processus destinataire est déjà actif.
- Un message pour prévenir l'initiateur de l'arrêt d'un processus fils actif.

Ces deux cas conduisent à la même conclusion, la suppression d'un fils pour le nœud courant dans *T*.

Le message de contrôle <*sig, p*> permet d'informer le processus père *p*. Ce message est effacé dès sa réception dans *p* et le compteur des fils de *p* est décrémenté.

La procédure suivante montre le pseudo code d'effacement des éléments dans l'arbre :

```
Ap : {le signal <sig,p> arrive en p}
debut
        receive <sig,p>
        SCp--
        si (SCP = 0) || (etatp = passif)
        alors
                si perep = p alors annonce()
                sinon send <sig, perep> à perep
                perep := nDef
fin
```

La figure 3.2 nous montre l'ensemble des procédures que constitue l'algorithme :

```
Etatp : (actif, passif)
        Init à actif si p = p0 sinon passif
SCp : entier (nombre de fils dans T)
Perep ∈ P init à p si p = p0 sinon nDef
Sp : {etatp = actif}
Debut
        Send <msg, p>;SCp++
Fin
Rp : {le message <msg, p> arrive en p}
Debut
        Receive<msg,q> ;Etat := actif
        Si perep = nDef alors perep := q ;Sinon send <sig, q> à q
Fin
Ip : {etatp = actif}
debut
        etatp := passif
        si (SCp = 0) alors
                si perep = p alors annonce()
                sinon send <sig, perep> à perep
                perep := nDef
fin
Ap : {le signal <sig,p> arrive en p}
debut
        receive <sig,p> ;SCp--
        si (SCP = 0) || (etatp = passif)
        alors
                si perep = p alors annonce() ;sinon send <sig, perep> à perep
                perep := nDef
fin
```

Figure 3.2 – Algorithme de Dijkstra-Scholten

La figure 3.3 est un exemple de l'exécution de l'algorithme de Dijkstra et Scholten : [30]

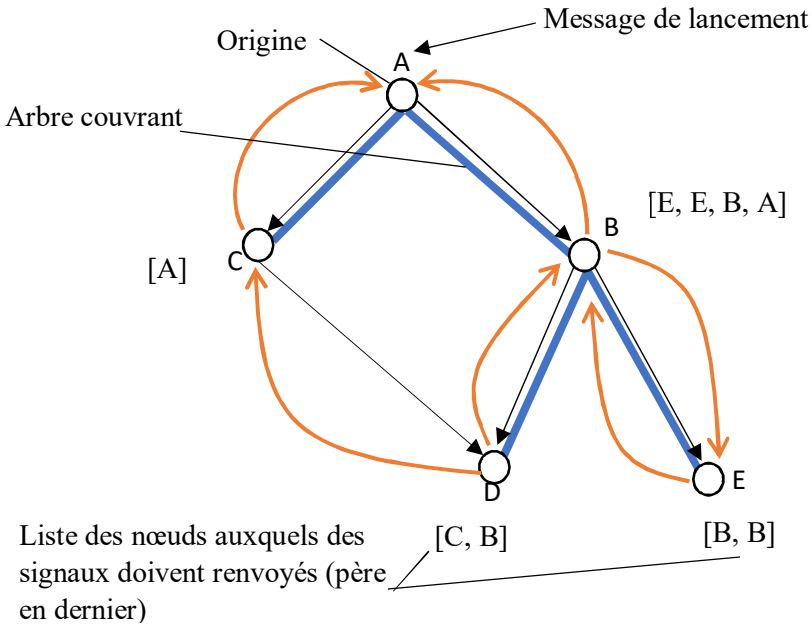

Figure 3.3 – Exemple d'exécution de l'algorithme de Dijkstra et Scholten

Calcul diffusant : lancé à partir d'un nœud origine

Un nœud qui reçoit un message effectue un calcul et peut soit passer à l'état passif soit envoyer un message à chacun de ses successeurs (le nombre total de messages envoyés est fini).

Un arbre couvrant implicite est construit (le premier message reçu par un nœud définit son père dans l'arbre).

Quand un nœud a terminé (feuille) ou reçu un signal de chaque arc sortant, il renvoie un signal sur un de ses arcs entrants (sauf celui du père), pour annuler le 'déficit' (nombre de message reçus – nombre de signaux renvoyés).

Un nœud renvoie un signal à son père quand son déficit a été annulé sur chaque arc sortant (via les signaux sur cet arc) et qu'il a renvoyé un signal sur chaque arc entrant autre que celui issu de père.

Le calcul est terminé quand le déficit du nœud origine est nul sur tous ses arcs sortants.

3. Processus d'élection de leader dans les réseaux Ad hoc

3.1 Introduction

Nous décrivons maintenant l'opération de cet algorithme de choix de leader dans le cadre d'un réseau mobile ad hoc. Avec l'introduction de la mobilité des nœuds, des pannes de nœuds, des pannes de liens, des partitions de réseau et une fusion de partitions, l'algorithme ordinaire est inadéquat. De plus de nombreux nœuds peuvent déclencher des élections leaders de manière concrète, chacun d'eux commençant indépendamment un calcul de diffusion en raison du manque de connaissances d'autres calculs lancés par d'autres nœuds.

3.2 Objectifs, contraintes et suppositions

Durant le développement de cet algorithme d'élection, nous devons en premier définir notre modèle de système, supposition et objectifs, le modèle du réseau ad hoc est un graphe non orienté qui change de forme durant le temps par le mouvement des nœuds. Les sommets dans le graphe correspondent aux nœuds mobiles et un bord entre deux nœuds qui représente le fait qu'ils sont dans le champs de transmission l'un de l'autre, et par conséquent peuvent directement communiquer l'un à l'autre. Le graphe peut être déconnecté, si le réseau est partitionné en raison du mouvement des nœuds. Les suppositions suivantes concernent les nœuds et l'architecture du système.

- La valeur du nœud : chaque nœud possède une valeur. Cette valeur du nœud indique si le nœud est potentiellement désiré en tant que leader du réseau, et peut-être un attribut lié à la performance tel que la batterie des nœuds, ..., etc.
- Identifiant de nœud unique et ordonné : tous les nœuds ont un unique identificateur. Ils sont utilisés pour identifier les participants au processus de l'élection. Les ID's sont utilisés pour briser les liens entre les nœuds qui ont la même valeur.
- Les liens : les liens sont bidirectionnels et FIFO, c'est-à-dire les messages sont délivrés en ordre sur un lien entre deux voisins.
- Comportement des nœuds : la mobilité des nœuds peut entrainer des changements arbitraires de topologie, incluent le partitionnement. En outre, les nœuds peuvent tomber en panne à n'importe qu'elle moment et revenir dans le réseau n'importe quand.
- Communication nœud à nœud : une livraison de messages est garantie uniquement lorsque l'expéditeur et le récepteur, restent connecter durant la transmission du message.

- La taille tampon : chaque nœud a un tampon de réception suffisamment grand pour éviter le débordement de la mémoire tampon à n'importe quel point de sa durée de vie.

L'objectif de cet algorithme d'élection de leader est d'assurer après un nombre fini de changement de topologie, éventuellement chaque nœud i possède un leader ayant doté de la valeur la plus grande parmi tous les nœuds connectés.

3.3 Explication d'algorithme

1. Variables et types de message :

   Les types de messages et les variables utilisées dans l'algorithme sont indiqués respectivement dans la Tableau 3.1 et la Tableau 3.2. L'algorithme comporte cinq types de messages : election, ack, leader, Probe et Reply. Dans ce qui suit la description en détail des messages :

   - Election : les messages d'élection sont utilisés pour agrandir l'arbre couvrant. Quand une élection est déclenchée à partir d'un nœud source s, le nœud commence un calcul de diffusion en envoyant un message d'élection a tous ces voisins les plus proches. Chaque nœud, i, autre que le nœud source, désigne le voisin à partir duquel il a reçu en premier un message d'élection tel que le nœud parent dans l'arbre couvrant. Le nœud i diffuse le message d'élection à tous ses voisins sauf le nœud source.
   - Ack : quand un nœud i reçois un message d'élection à partir d'un nœud voisin qui n'est pas le parent, il répond immédiatement avec un message Ack. Cependant le nœud i ne renvoie pas immédiatement un message Ack. Au lieu de cela, il attend jusqu'à ce qu'il ait reçu des message d'Ack de tous les autres nœuds (children), avant d'envoyer un message Ack au nœud parent (source). Le message Ack envoyé par i au nœud source contient l'information de l'élection de leader qui est basé sur le message Ack qu'a reçu i des autres nœuds (children). Une fois que l'arbre couvrant est totalement agrandi, ce dernier rétrécie vers la source spécifiquement, une fois que tous les message d'élection sortant de i, sont acquitter.
   - Leader : une fois le nœud source a reçu les messages Ack, de la part de tous les autre nœuds (children), il diffuse ensuite un message d'élection à tous les autres nœuds en annonçant le nœud qui possède le plus grand identificateur.

Chaque nœud i maintient une variable booléenne δi, dont la valeur est 0 si le nœud i a un leader et 1 si il est en train d'en choisir un. La variable srci contient l'indice de calcul du calcul

diffusant dans lequel le nœud *i* participe actuellement. Comme nous le verrons dans le point 3 plus tard, cet indice de calcul identifie de manière unique un calcul et est nécessaire pour gérer des calculs multiples et simultanés. Au cours d'un calcul diffusant, le nœud *i* conserve la trace de son parent, pi. La variable Δi est définie sur 0 si le nœud *i* a envoyé son message Ack en attente à son parent et 1 s'il n'a pas. Chaque nœud *i* maintient son leader *lidi*. *Ni* est la liste des voisins actuels de *i* et, *Si* représente l'ensemble des nœuds qui n'ont pas encore entendu un message Ack. Il est mis à jour chaque fois qu'il reçois un message Ack.

Message	Objectif
Election	Pour agrandir l'arbre couvrant
Ack	Accuser de réception d'un message d'élection
Leader	Pour annoncer le nouveau leader
Probe	Pour déterminer si un nœud est toujours connecté
Reply	Envoie d'une réponse à un message Probe

Tableau 3.1 – Type de message dans l'algorithme d'élection

variables	Explication
δ_i	Une variable binaire indiquant si i est dans une élection ou non
pi	Les nœuds parent de i dans l'arbre couvrant
Δ_i	Une variable binaire indiquant si i à envoyer un message Ack à pi ou non
lidi	Le leader de i
Ni	Les nœud voisins de i
Si	Ensemble de nœuds dont i doit recevoir un Ack depuis le nœud source
Src i	L'index de calcule de i

Tableau 3.2 – Variables maintenues par un nœud i pendant le processus d'élection

1. Début de l'élection

Chaque nœud commence l'exécution en initialisant les différentes variables de l'algorithme de sélection de leader. Après l'initialisation, l'algorithme en chaque nœud boucle à jamais, et chaque itération, vérifie si l'une des actions de la spécification de l'algorithme est activée, en exécutant au moins une action activée sur chaque itération en boucle.

2. Gestion multiple, calcul simultané

L'élection du leader d'un composant connecté émet périodiquement des messages de battement de cœur vers d'autres nœuds. L'absence de ces messages de son leader pour une période de temporisation prédéfinie déclenche un nouveau processus d'élection de leader sur un nœud. Il convient de noter que plus d'un nœud peut détecter simultanément le départ du leader et que chaque nœud peut déclencher des calculs diffusants. Cette algorithme s'occupe de multiples calculs diffusants simultanés en exigeant que chaque nœud participe à un seul calcul diffusé à la fois. Pour ce faire, chaque calcul diffusant est identifié par un indice de calcul. Cet indice de calcul est une paire, à savoir. <Num, id>, où id représente l'identifiant du noeud qui a initié ce calcul et num est un nombre entier, qui est décrit ci-dessous :

<num1, id1> $\succ$ <num2, id2> $\leftrightarrow$ ((num1 > num2) $\vee$ ((num1 = num2) $\wedge$ (id1> id2)))

Un calcul diffusant A est considéré comme ayant une priorité supérieure à un autre calcul diffusant B si l'indice de calcul A $\succ$ l'indice de calcul B.

Une source donnée commence toujours un calcul diffusant avec un nombre supérieur à celui de tout autre calcul qu'il a précédemment initié, tandis que le champ source-id est utilisé pour briser les liens entre les calculs diffusants simultanés avec différentes sources mais la même valeur numérique. En conséquence, il existe une commande totale sur les indices de calcul. La variable num est incrémentée chaque fois qu'un nœud commence un nouveau calcul diffusant. Lorsqu'un nœud participant à un calcul diffusant «entend» un autre calcul avec un indice de calcul plus élevé, le nœud cesse de participer à son calcul actuel en faveur de l'indice de calcul le plus élevé. Par exemple, dans la Figure 3.6, le nœud G envoie un message d'élection avec l'indice de calcul, <3, D>, au nœud A dont l'indice de calcul actuel est <3, B>. Lors de la réception de ce message d'élection, le nœud A cesse de participer à son calcul actuel, définit son indice de calcul à <3, D>, comme le montre la Figure 3.7, et propage le message d'élection reçu aux nœuds B et C.

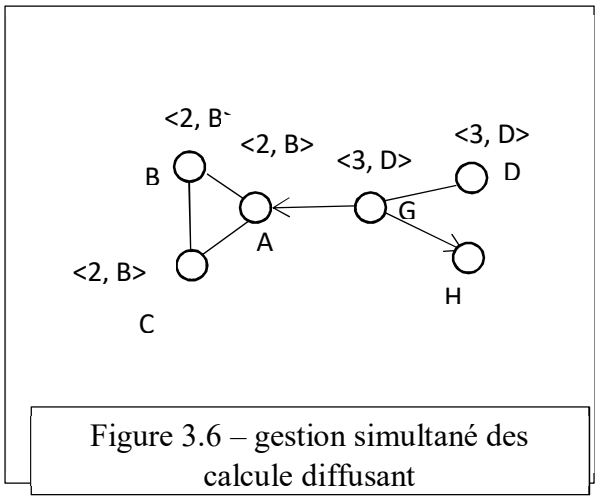

Figure 3.6 – gestion simultané des calcule diffusant

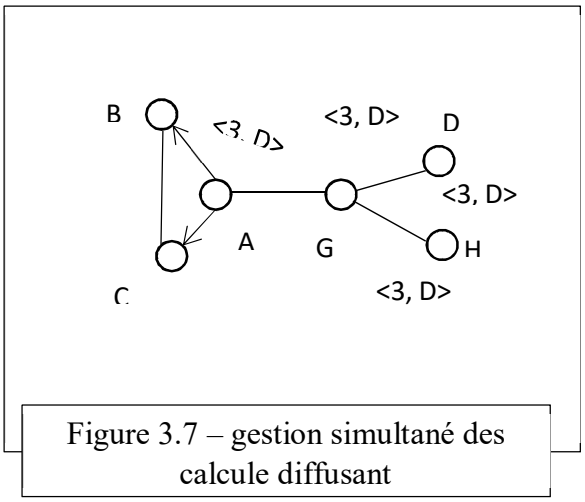

Figure 3.7 – gestion simultané des calcule diffusant

3.4 Algorithme effectué par les nœuds

L'idée principale de cet algorithme, comme précisé dans le chapitre précédent, c'est d'agrandir et de rétrécir un arbre couvrant pendant le processus d'élection et d'annoncer le leader après que l'arbre se rétrécit complètement. Cependant, si le mouvement du nœud effectue un changement dans cette arborescence, les nœuds détectent ces changements et prennent les mesures appropriées. Dans cette section, nous décrivons à travers des exemples, comment l'algorithme de l'élection accepte des modifications arbitraires de la topologie induite par la mobilité des nœuds.

3.4.1 Initier l'élection

Le nœud *i* commence le processus d'élection en réponse au départ de son leader actuel, le nœud *i* commence le processus d'agrandissement d'un arbre couvrant par des messages d'élection de propagation à ses voisins, les informant du début d'une élection d'un nouveau leader. En déclenchant une nouvelle élection, le nœud *i* définit sa variable δ_i à *1* pour indiquer qu'elle est actuellement impliquée dans une élection, le nœud *i* lance un message leader qu'après avoir reçu les messages Ack de tous les nœuds auxquels il envoie un message Election. La liste *Si* est donc initialisée a *Ni*.

3.4.2 Construction de l'arbre

Le nœud *j*, après avoir reçu un message Election du nœud *i*, rejoint l'arborescence en définissant son pointeur parent, *pj* = *i* et, à son tour, propage les messages d'élection vers ses

propres voisins dans l'ensemble *Nj*, ces messages d'élection sont propagés vers tous les nœuds et, éventuellement, l'arbre couvrant de nœuds construit.

3.4.3 Manipulation des partitions de nœuds

Une fois que le nœud *i* rejoint une élection, il doit recevoir un message Ack de tous les nœuds dans la liste *Si* avant qu'il puisse signaler un message Ack à son nœud père. Cependant, en raison de la mobilité des nœuds, il se peut que le nœud *j*, qui doit encore signaler un message Ack, se déconnecte du nœud *i*. Le nœud i doit détecter cet événement, car sinon, il ne signalera aucun message Ack à son parent et, par conséquent, aucun leader ne sera annoncé.

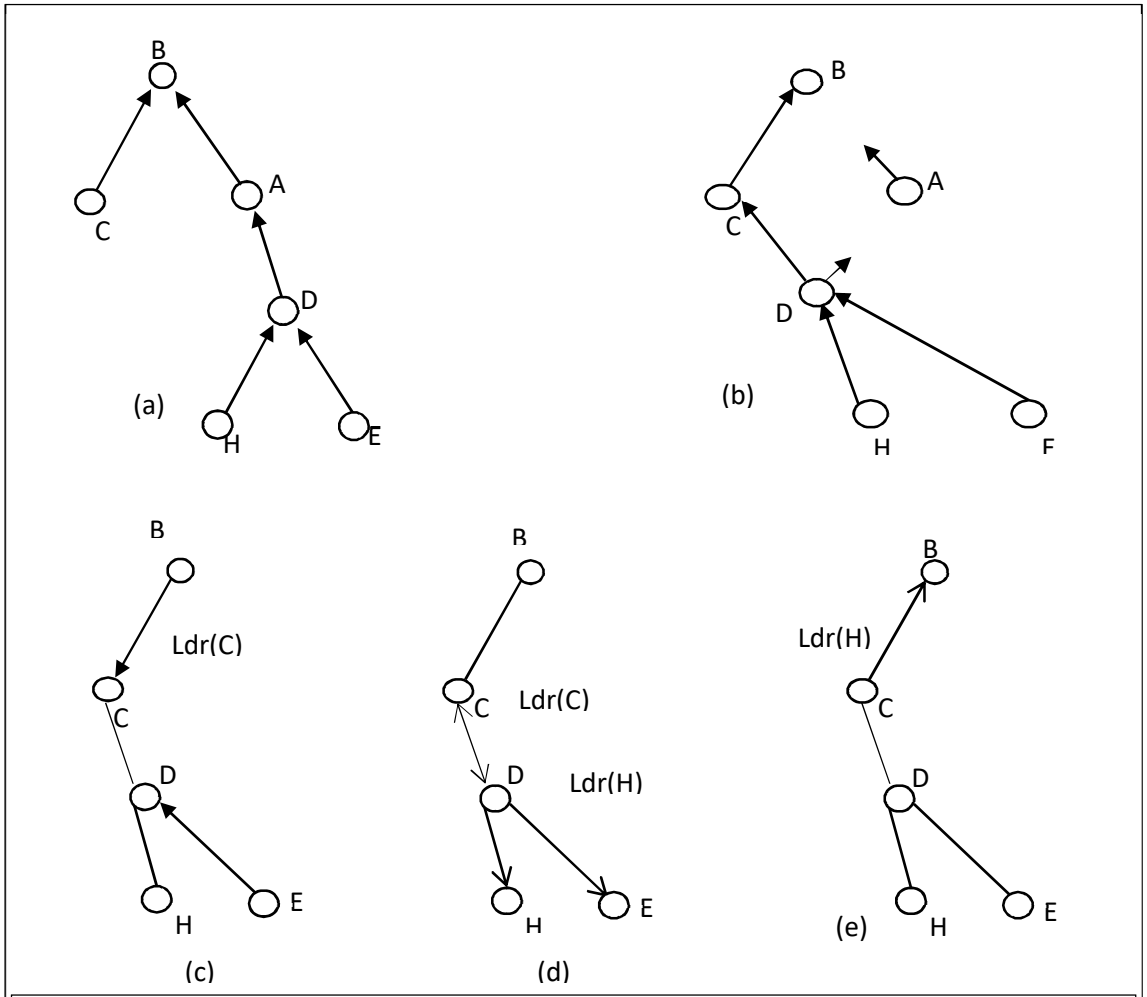

Figure 3.8 - Fonctionnement de l'algorithme d'élection de Leader face aux partitions

Considérons un scénario dans lequel un père-enfant se déconnecte pendant le processus d'élection, c'est-à-dire que la condition *di*, *j* = infini est vraie pour certains *Si*, illustrée à la Figure

3.8. La Figure 3.8 (a) montre un exemple de topologie où le pointeur parent représente L'arbre construit. En raison de la mobilité des nœuds, le nœud *A* se déconnecte du reste des nœuds et la topologie change à celle illustrée à la Figure 3.8 (b). Pour détecter de tels événements, chaque nœud dans l'arbre couvrant envoie périodiquement des messages «Probe» à chaque nœud *j* dans sa liste *Si*. Un nœud qui reçoit un message "Porbe" répond avec un message "Reply". L'absence d'un message "Reply" à partir d'un nœud j pour une période de délai d'expiration fait que le nœud *i* supprime j de la liste Si et n'attend plus qu'un message Ack du nœud j. Comme le montre la Figure 3.8 le nœud *B*, qui a déjà reçu un message Ack du nœud *C*, n'a pas encore entendu un message Ack du nœud A, finit par inférer, en utilisant le message 'Probe', que le nœud *A* est parti. Le nœud *B* supprime *A* de la liste *Sb*. Le nœud *B* n'a plus de messages Ack pour attendre et diffuser un message 'Leader' annonçant *C* comme leader comme illustré à la Figure 3.8 (c).

Lorsqu'un nœud se déconnecte de son père, il ne peut plus signaler un message Ack à son père. Par conséquent, il met fin au calcul diffusant en annonçant son nœud descendant maximal en tant que leader. Dans notre exemple, le nœud *D*, qui a *A* comme parent, reçoit finalement des messages Ack de tous ses fils immédiats. Comme le montre la Figure 3.8 (d), le nœud *D* détecte par la suite le départ du nœud *A* et met fin au calcul en diffusant un message 'Leader', en annonçant *H* comme leader. Essentiellement, le nœud *D*, en l'absence du nœud parent, rapporte son nœud descendant maximal ses voisins actuels.

Enfin, le nœud *C*, dont le leader actuel est lui-même, propage le nouveau leader, H, en amont du nœud *B*. Ainsi, tous les nœuds finissent par avoir le nœud *H* en tant que leader. Le nœud *A* détecte également le départ des nœuds *D* et de son parent, nœud *B*. Dans ce cas, le nœud *A* se préconise d'être le leader.

3.4.4 Manipulation de fusion de partition

La mobilité des nœuds peut aussi provoquer Les partitions à fusionner. Il existe plusieurs possibilités. Le plus simple cas, comme le montre la Figure 3.9 (a), implique deux connexions Composants, chacun avec un leader unique, fusionnant par La formation d'un nouveau lien entre les nœuds *A* et *U*. Les nœuds *A* et *U* échangent ensuite les identités du leader sur le lien nouvellement formé. Puisque le nœud U a un Leader d'identité supérieur (W) que A (C), *A* adopte W comme son propre Leader, puis diffuse le nouveau leader vers le reste de la Nœuds dans son composant.

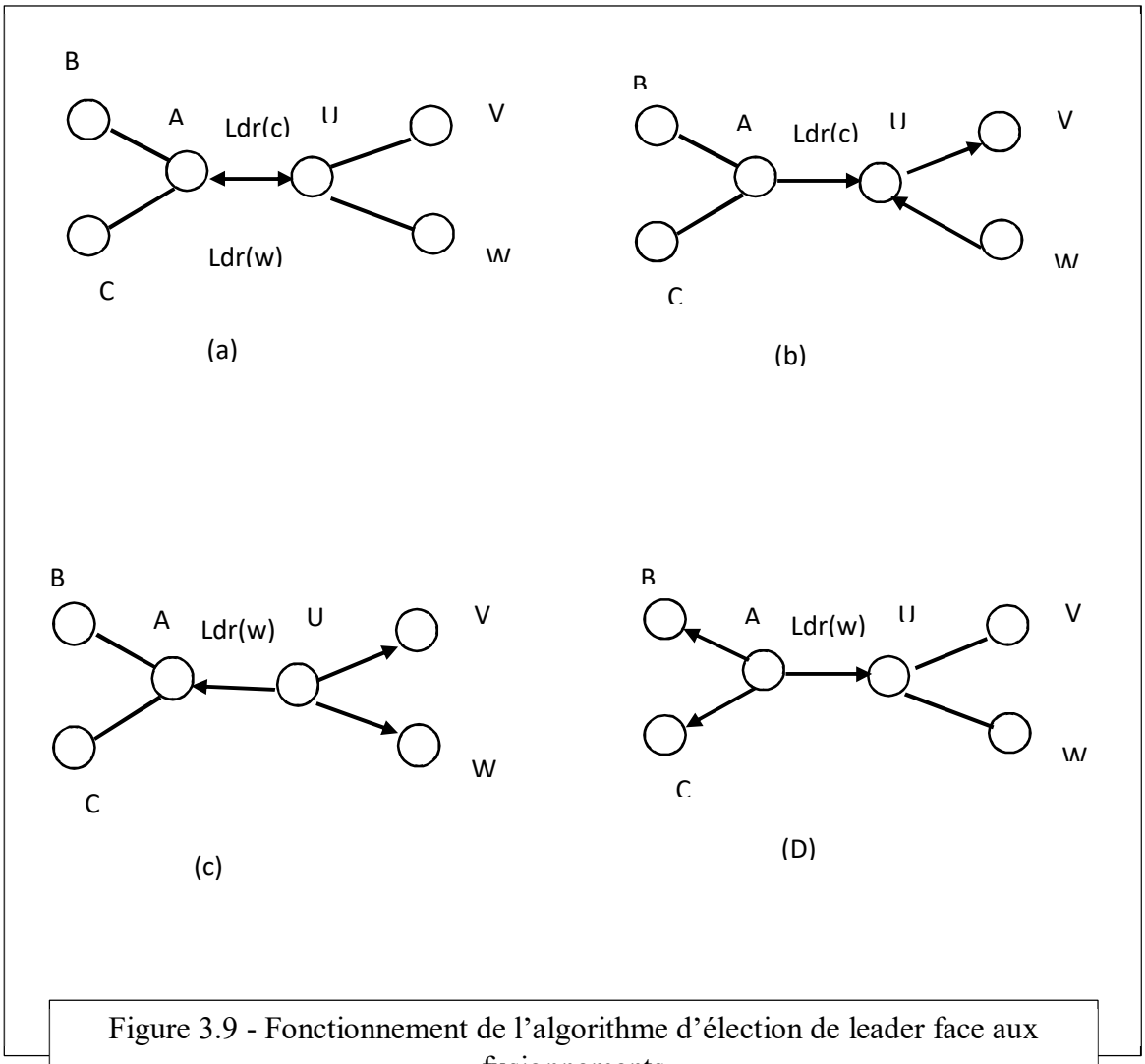

Figure 3.9 - Fonctionnement de l'algorithme d'élection de leader face aux fusionnements

Une autre possibilité est que l'un ou l'autre des composants Fusionner sont sans leader et impliqués dans un calcul. Comme le montre la figure 3.9(b), les nœuds U, V, W sont Impliqué dans un calcul et fusion avec les nœuds A, B et C qui ont C comme leur leader. L'algorithme gère ce Cas en permettant au calcul en cours de se terminer avant L'échange des identités de leader a lieu. Sur la figure 3.9(b), Le nœud A, lors de la détection d'une nouvelle formation de lien, annonce son Identité de leader au nœud U. À la fin de la phase en cours Calcul, le nœud U annonce son leader (nœud W) au nœud A, qui adopte W comme son nouveau leader et propage cette information aux nœuds B et C. L'affaire lors de la fusion les composants ont un calcul continu est également géré de même.

3.4.5 Manipulation de nœuds et de redémarrages

Cet algorithme tolère également des collisions et des récupérations de nœuds arbitraires. L'échec d'un nœud est traité comme une instance de partitionnement réseau et des actions appropriées sont prises, comme décrit précédemment. Pour cet algorithme pour tolérer les récupérations de nœuds, il est supposé que lorsqu'un nœud se rétablit d'un accident, il lance d'abord l'élection. À la fin du lancement d'élection, le nœud récupéré est sans leader et commence donc une nouvelle élection pour trouver son leader. En substance, les blocages de nœuds sont traités comme des occurrences de partitions tandis que l'événement d'un nœud récupérant d'une défaillance est traité comme la fusion de deux composants.

Conclusion

Dans ce chapitre, nous avons présenté un algorithme qui permet d'élire un seul et unique leader dans un réseau, cette algorithme se base sur le principe de la construction d'un arbre couvrant et la détection de terminaison de Djikstra et Scholten.

Dans le chapitre suivant nous allons décrire les différentes étapes de la conception de notre simulateur de réseau ad hoc qui permet entre autre d'utiliser l'algorithme de Vasudevan, Kurose et Towsely.

Implémentation de l'algorithme de Vasudevan, Kurose et Towsely

Introduction

Après l'étape de l'étude détailler de l'algorithme d'élection dans les réseaux Ad hoc, vient l'étape de développement qui sert en pratique tout ce que nous avons réalisé dans le précédant chapitre.

Dans ce chapitre, nous allons présenter notre simulateur de réseaux Ad-hoc qui permet de simuler l'algorithme présenté dans le chapitre précédant. Par suite, nous allons donner un aperçu des interfaces utilisées, et à la fin les résultats de comparaisons seront mis en évidence.

1. Réalisation de l'application

1.1 Eclipse

C'est un projet, décliné et organisé en un ensemble de sous-projet de développement logiciels, de la fondation Eclips visant à développer un environnement de production de logiciels libre qui soit extensible, universelle et polyvalent, en s'appuyant principalement sur Java. Son objectif est de produire et fournir des outils pour la réalisation de logiciels, englobant les activités de programmation (notamment environnement de développement intégré et Framework) mais aussi d'AGL recouvrant modélisation, conception, test, gestion de configuration, réporting, Son IDE, partie intégrante du projet, vise notamment à supporter tous langage de programmation à l'instar de Microsoft Visual Studio. [31]

1.2 Java

Le langage Java est un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy, présenté officiellement le 23 mai 1995 au SunWorld. La particularité et l'objectif central de Java est que les logiciels écrits dans ce langage doivent être très facilement portables sur plusieurs systèmes d'exploitation tel que Unix, Windows, Mac OS, avec peu ou pas de modifications. Pour cela divers

plateformes et Framework associés visent à guider, sinon garantir, cette portabilité des applications développées en Java [32].

Java reprend en grande partie la syntaxe du langage C++, très utilisé par les informaticiens. Java a été épuré des concepts les plus subtils de C++ et à la fois les plus déroutants, tels que les pointeurs et références, ou l'héritage multiple contourné par l'implémentation des interfaces. Les concepteurs ont privilégié l'approche orientée objet de sorte qu'en Java, tout est objet à l'exception des types primitifs (nombre entiers, nombre à virgules flottante, etc.). Java permet de développer des application client-serveur. Cotés client, les applets sont à l'origine de la notoriété du langage. C'est surtout côté serveur que Java s'est imposé dans le milieu de l'entreprise grâce aux servlets. [33]

1.3 JBotsim

JBotsim est une bibliothèque de simulation pour les algorithmes distribués dans les réseaux dynamiques. Le style de programmation est principalement axé sur les évènements : le code peut réagir à divers événements (impulsion d'une horloge, apparition / disparition d'un lien, arrivé d'un message, mouvement du nœud, etc.). Les mouvements du nœud peuvent être contrôlé par un programme. Ou au moyen d'interaction basée sur la souris pendant l'exécution. Au-delà de ses caractéristiques, le principal atout de JBotsim est sa simplicité d'utilisation. [34]

2. Présentation des différentes classes

Au début des simulations, une topologie des nœuds mobiles doit être défini durant cette étape, en spécifiant un certain nombre de paramètres comme : le nombre de nœuds, leurs coordonnées au début aléatoires, la portée et la surface. Ces paramètres seront affichés sur l'espace fenêtre d'exécution montré sur la figure 4.2. On suppose que la topologie du réseau et connexe pour éviter les partitions. Notre simulateur est constitué de plusieurs classes qui seront décrites ci-dessous :

2.1 La classe Main

Pour qu'un programme en Java puisse être exécuter, la méthode *main* est indispensable, cette méthode permet de faire l'appel à d'autres méthodes. La figure suivante nous montre le code source de la méthode main :

En exécutant tous cette classe, on obtient la figure suivante :

Figure 4.1 - fenêtre d'accueil du simulateur

2.2 La classe Fenetre

La classe 'Fenetre' : elle contient diverses méthodes et constructeur, qui sont :

1. Le constructeur Fenetre() ;
2. La méthode actionPerfomed() ;

2.2.1 Variable utilisées dans la classe Fenetre

Types des variables	Variables	Explication
Tableau	choix	Permet le choix entre les différentes options présente dans l'application choix de définir la portée ou le champ de communication.
Entier	k1	Sert a fixé le champs de communication.
	k2	Sert a fixé la portée d'un nœud.
	i	Variable de type entier, cette variable nous sert de compteur pour l'ajout de nœud.
	nbr	Permet de stocker le résultat de la conversion du nombre de nœud à ajouter de caractères vers un entier.
Bouton	Inserer le nombre de nœud	Permet à l'utilisateur d'ajouter des nœud dans l'interface du simulateur.

Boutons	demarer	Permet à l'utilisateur de lancer la simulation.
	pause	Permet à l'utilisateur de faire une pause à la simulation.
	reset	Permet de réinitialiser les paramètres par défaut du simulateur.
	parametre	permet de faire apparaitre une boite de dialogue qui permet à l'utilisateur d'y insérer les valeurs de champs de communication et la portée.
	Enregistrer les resultats	Permet de sauvgarder les données générées pendant la simulation dans un tableau.
JPanel	left	permet de contenir la zone d'affichage de la console.
	south	permet de contenir les boutons de gestion du simulateur.
JTextField	nbr_noeud	permet de contenir le nombre de nœuds que l'utilisateur souhaite ajouter au simulateur.
JTextArea	affichage	Permet de récupérer les entrées/sorties de la console d'exécution de Eclips.
JMenuBar	menuBar	Permet de regrouper les items du menu, test1, test2, test3
JMenu	test1, test2, test3	Ces variables constitue la barre menu.
JMenuItem	gra	Variable qui permet de lancer la fenêtre du graphe pour la simulation
	tab	Variable qui permet de générer un tableau qui stock les valeurs de simulation.
	rea	Variable qui permet de lancer une fenêtre qui fournit des instruction pour utiliser le simulateur
JOptionPane	jop2	Permet d'afficher la boite de dialogue invitant l'utilisateur à choisir un élément parmi ceux disponible dans le menu déroulant.
	jop1	Permet d'afficher une boite de dialogue qui informe l'utilisateur qu'il doit d'abord initialiser le nombre de nœud.

Tableau 4.1 – Variables utilisés dans la classe Fenetre

2.2.2 Le constructeur Fenetre

Cette méthode est responsable de tout ce qui est visuel, c.-à-d., ce que l'utilisateur va voir en exécutant le code source. A partir de la fenêtre en peux définir les paramètres de simulation.

2.2.3 La méthode actionPerformed

Cette méthode se charge de tout ce qui est interaction IHM (Interface Homme Machine). Elle est utilisée pour l'écoute des différentes actions que l'utilisateur effectue sur l'interface. Et en voici un exemple de son utilisations.

Le bouton paramètre une fois cliquer dessus, une boite de dialogue va inviter l'utilisateur à choisir l'une des options présentes, comme nous le montre cette figure :

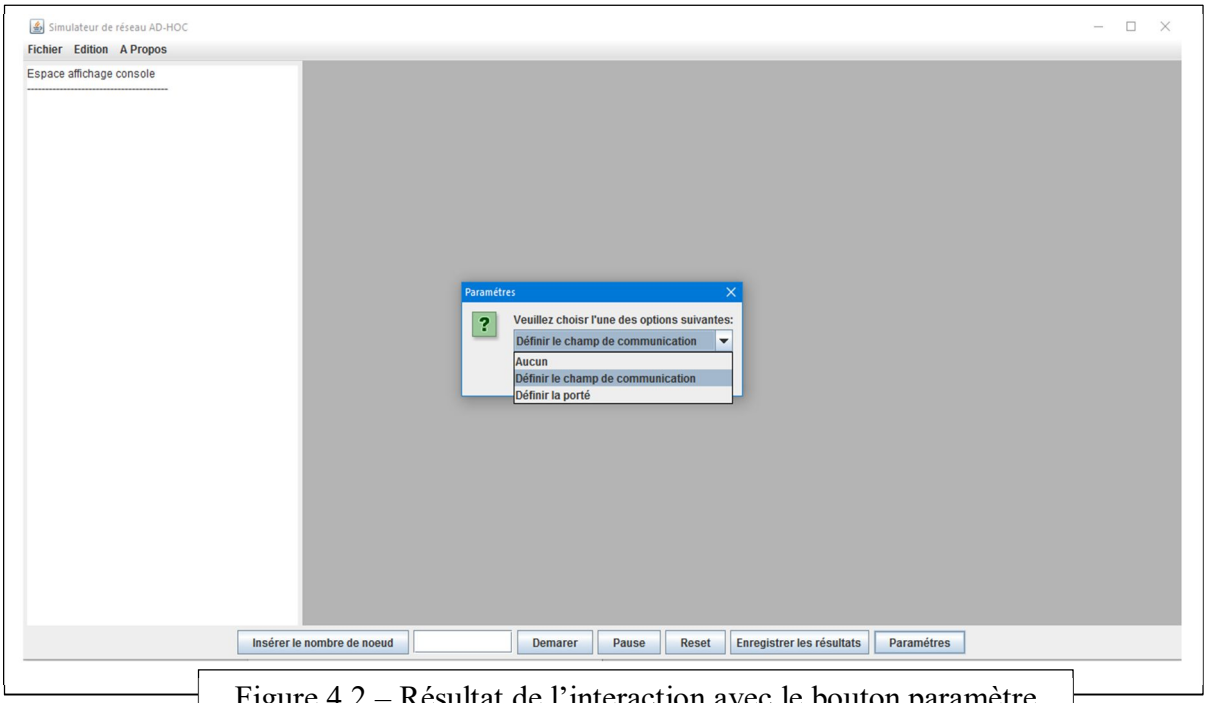

Figure 4.2 – Résultat de l'interaction avec le bouton paramètre

La sélection d'une option dans le menu déroulant nous offre une possibilité d'y insérer des valeurs, une pour définir le champ de communication et l'autre pour définir la portée, la figure suivante nous montre la boite de dialogue ou l'application invite l'utilisateur à insérer une valeur, la figure suivante nous montre le cas de l'initialisation de la valeur de champ de communication, l'autre étant similaire à celle-ci il n'est pas nécessaire de la montré :

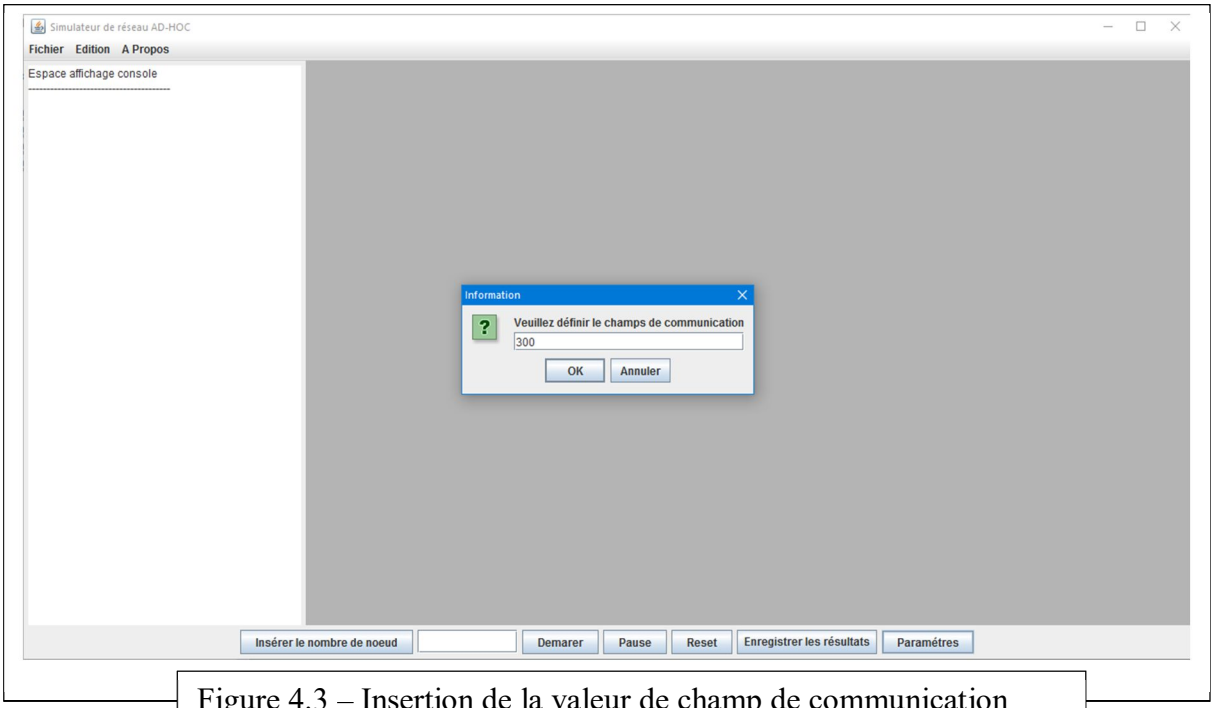

Figure 4.3 – Insertion de la valeur de champ de communication

3. La classe Algorithme

La création de ce code est basée sur le simulateur Jbotsim, qui est expliqués un peu plus haut dans ce chapitre ; dans ce qui suit la description des principales méthodes qu'offre ce Framework, et que nous exploitons pour notre travail :

- La méthode onSelection() ;
- La méthode onStart() ;
- La méthode onClock() ;
- La méthode onMessage() ;
- La méthode onSensingNode() ;

3.1 Variables utilisées dans la classe Algorithme

Types des variables	Variables	Explication
Node	source	Variable de type Node, utiliser pour affecter le nœud
	Leader	Variable de type Node, utiliser pour définir un nœud leader.
List	Children	Liste de nœud représentant les nœud fils du nœud source lors de la création de l'arbre couvrant

double	speed	Représente la vitesse de mouvement des nœuds.
	direction	Représente la direction que chaque nœud va prendre.
Chaine de caractères	racine	Variable de type chaine de caractères, indiquent à l'utilisateur que le nœud sélectionner est le nœud racine.
	elected	Variable de type chaine de caractères, indiquent à l'utilisateur que le nœud leader de la topologie.

Tableau 4.2 – Variables utilisés dans la classe algorithme

3.2 Description des méthodes

A. La méthode OnSelection()

Cette méthode a pour but de permettre à l'utilisateur de sélectionner un nœud dans la topologie parmi tous les nœuds présents, et cela en appuyant sur le bouton central de la sourie.

Cette méthode dans notre cas permet de sélectionner un nœud source pour former un arbre couvrant, la figure 4.9 nous montre le nœud sélectionner qui est coloré en noire

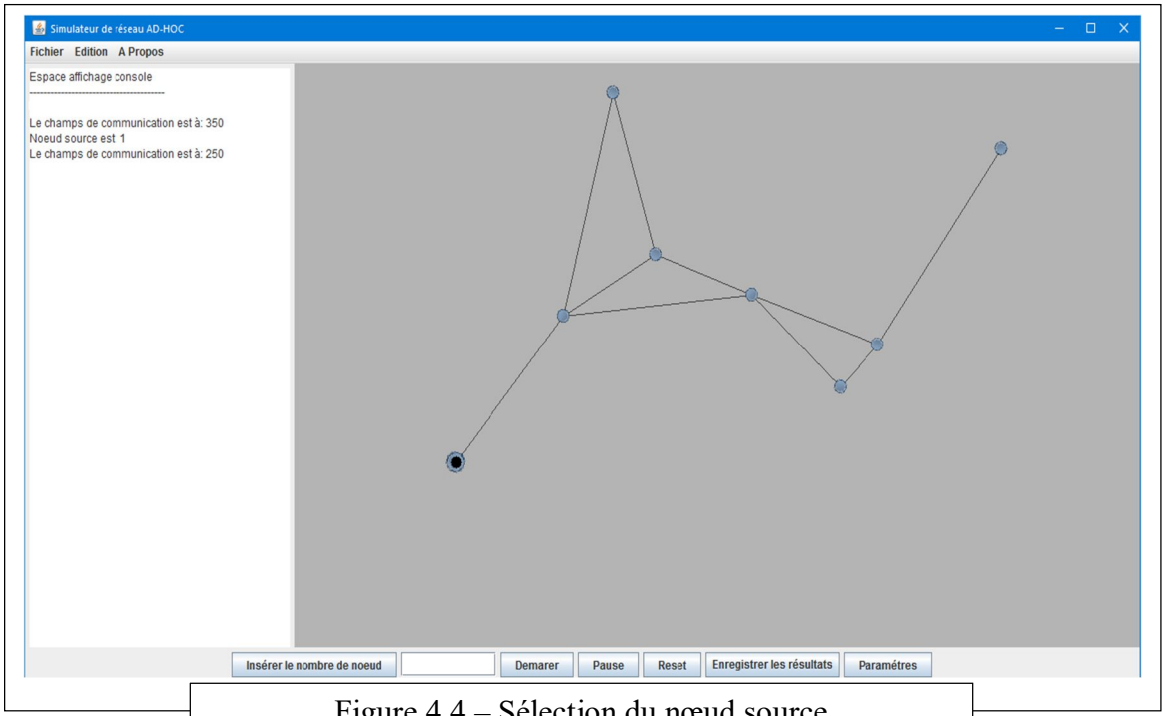

Figure 4.4 – Sélection du nœud source

B. La méthode onStart()

Cette méthode est appelée au début du lancement de la simulation, elle a pour but de donner une direction aléatoire pour chaque nœud présent dans la topologie.

C. La méthode onClock()

En modifiant cette méthode prédéfinie une action sera effectuée lors de l'impulsion d'horloge. La figure suivante nous montre le code de cette méthode :

D. La méthode onMessage()

Cette méthode s'exécute à chaque fois qu'un nœud reçoit un message. Celle-ci est responsable de la génération de l'arbre couvrant une fois cliquer sur le nœud source.

La figure suivante nous montre le résultat de l'éxecution de la méthode onMessage sur une topologie.

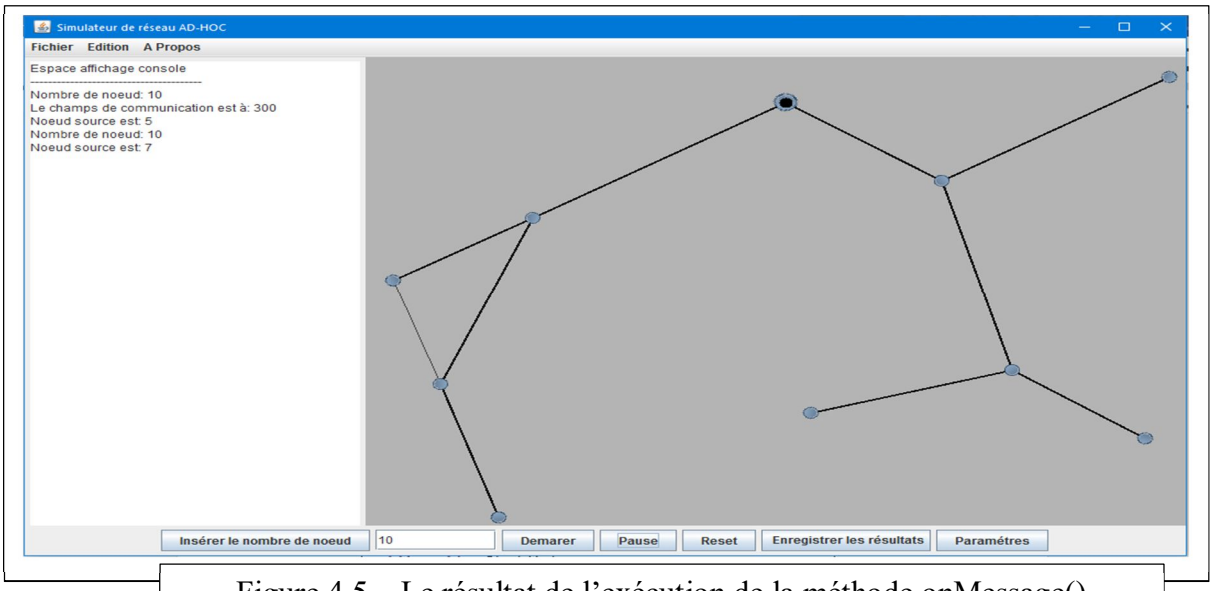

Figure 4.5 – Le résultat de l'exécution de la méthode onMessage()

E. La méthode onSensingIn()

Cette méthode est appelée quand un autre nœud est détecté pour la première fois. Vue la caractéristique de la topologie qui est dynamique certains nœuds vont se croisés plusieurs fois durant la simulation, cette méthode prend en considération cela, chaque fois

que deux nœud se croise, elle vérifie s'il ne s'était pas déjà croisé donc, cette méthode ne va pas s'exécuter pour ces deux nœuds.

La figure 4.15 nous montre de résultat de l'exécution du protocole d'élection de leader :

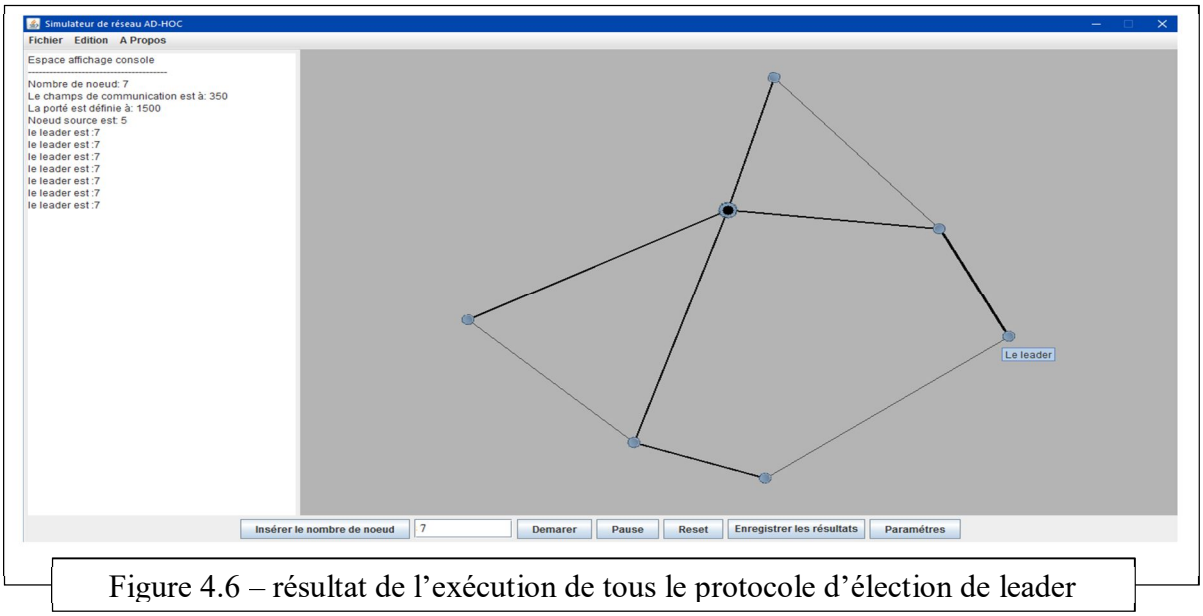

Figure 4.6 – résultat de l'exécution de tous le protocole d'élection de leader

Le leader ayant été choisie est représenter dans le graphe par un point rouge, avec un label indiquent sont état dans ce cas le leader.

4. La classe graphe

Cette classe a pour rôle de récolter les différentes données des simulations et ainsi les affichées sous formes de courbe, cette classe est appelée à partir de la classe fenêtre en sélectionnant courbe de simulation dans le menu Edition, la figure suivante nous montre le graphe en question :

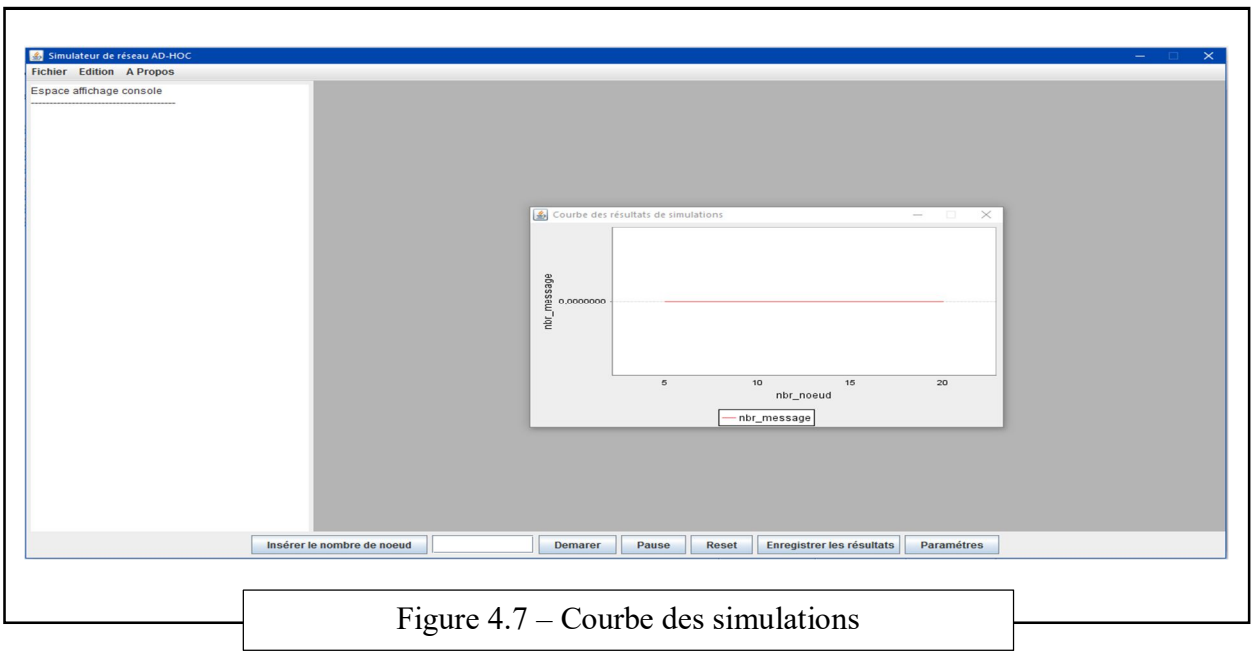

Figure 4.7 – Courbe des simulations

5. La classe tableau

Cette classe permet de générer un tableau, il stocke les informations de simulation, dans notre cas le nombre de message que l'algorithme va générer lors de la simulation et cela avec plusieurs topologie (nombre de nœud) définie au préalable dans le tableau auxquelles on peut y apporter des modifications, cette classe est appelée à partir de la classe fenêtre en sélectionnant tableau des paramètres dans le menu Edition, la figure suivante nous montre le tableau des paramètres :

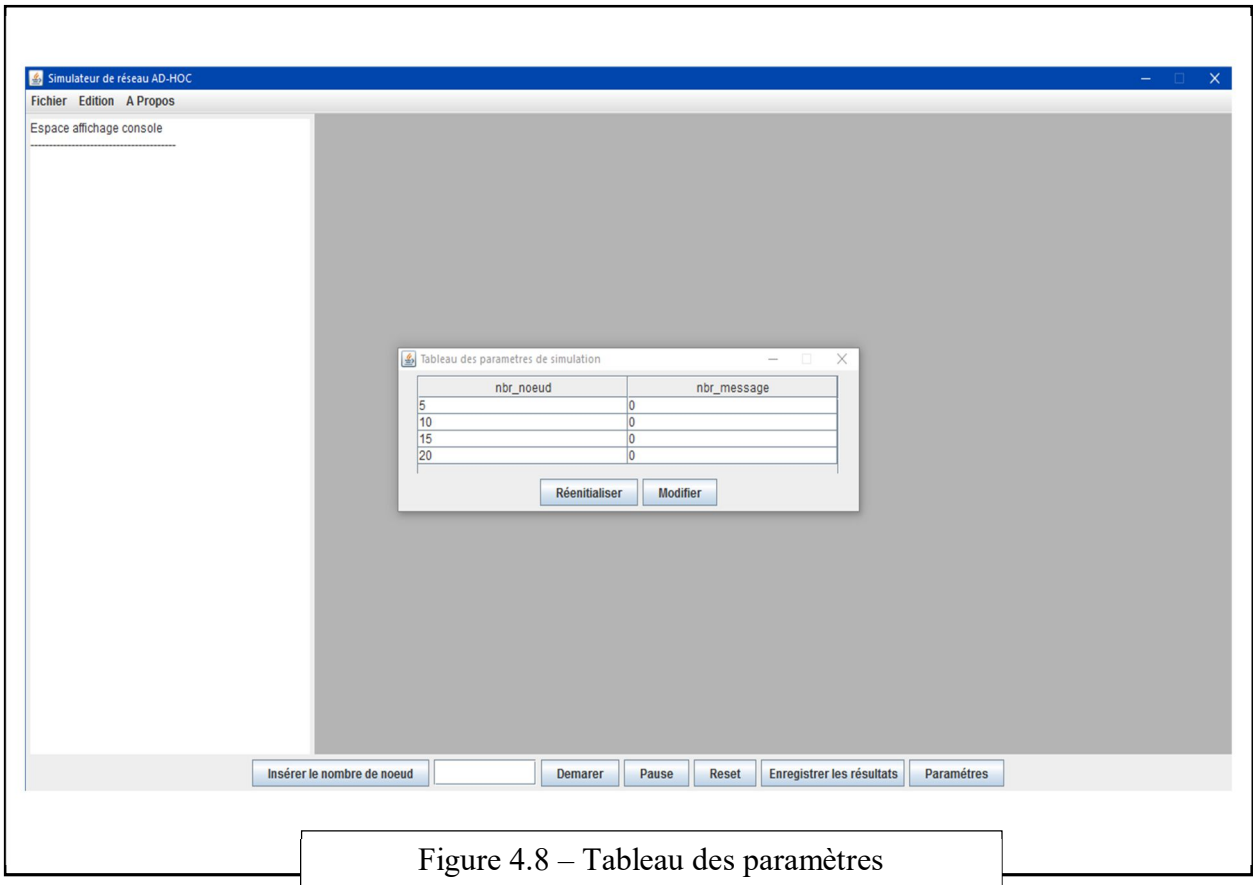

Figure 4.8 – Tableau des paramètres

Le bouton Réinitialiser sur le tableau des paramètres permet à l'utilisateur de mettre à zéro le contenue du tableau.

Le bouton Modifier permet de modifier le nombre de nœud pour lequel l'algorithme va être tester, la figure suivante nous montre comment changer les valeurs des nœuds :

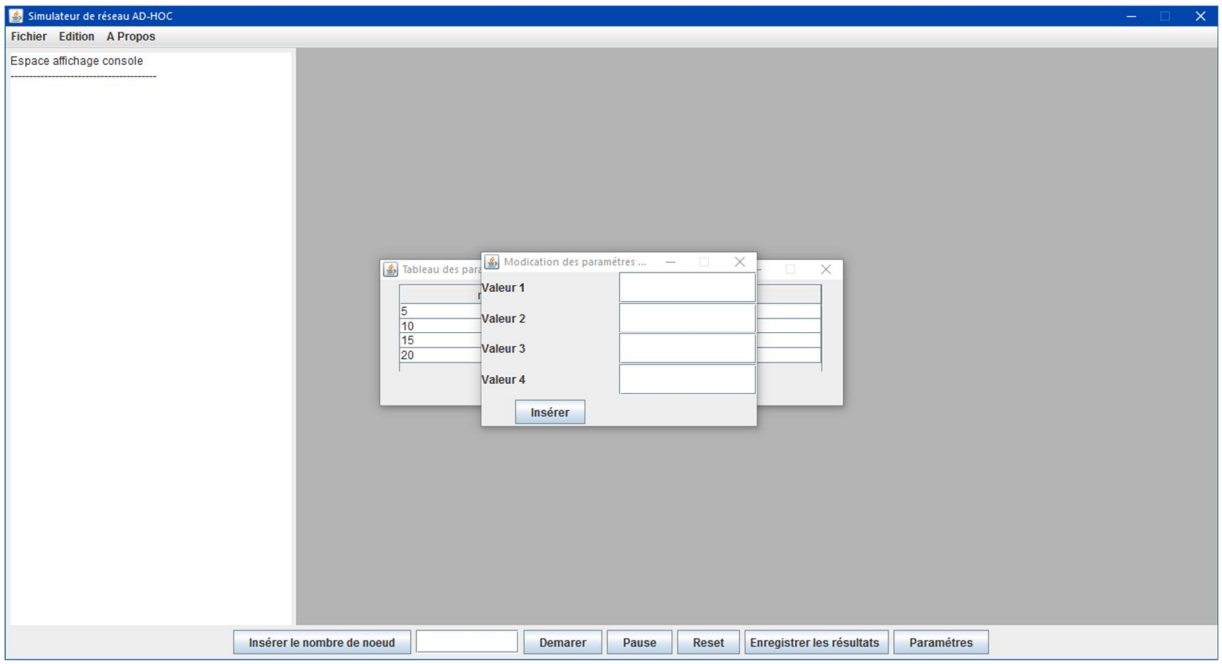

Figure 4.9 – Tableau pour la modification des valeurs des nœuds

Le tableau permettant de modifier les valeurs des nœuds offre à l'utilisateur 4 valeurs à insérer, le nombre de valeurs à insérer sera définie par l'utilisateur dans une future mise à jour.

6. Etude des performances

6.1 Evaluation de l'algorithme

Nous avons effectué des simulations pour évaluer la performance de l'algorithme qu'on a choisi pour effectuer l'élection de leader, pour ce faire on a effectué une comparaison de nombre de message par rapport aux nombre de nœuds présent lors de la simulation, on a pu avoir deux courbes montrant la performance de notre algorithme par rapport aux autre algorithme n'utilisant pas le concept de la génération d'un arbre couvrant pour l'élection.

6.2 Configuration et métriques de simulation

Le système de simulation se compose de deux modules : le réseau et l'algorithme d'élection, les paramètres principaux de la simulation sont présentés dans le Tableau 4.1.

Les nœuds du réseau sont randomisés sur le territoire carré. Le nombre total de nœuds est varié pour examiner l'effet de l'échelle du système sur la performance.

Nombre de nœuds	5,10,15,20
Plage de communication	250

Tableau 4.3 – Paramètres de simulation

6.3 Résultat de l'étude analytique

Comme on peut le distinguer dans la figure 4.10, les résultats sont convainquant, l'algorithme choisie pour notre simulation a permis de générer beaucoup moins de message par rapport à l'algorithme de Hatzis et al [24]. qui peut arriver à générer 103 message contre 57 message pour 20 nœuds présent lors de la simulation.

Cela démontre que l'algorithme de Vasudevan, Kuros et Towsely et le mieux adapter en terme de performance pour les réseaux ad hoc, et permet d'économiser de l'énergie. La figure suivante représente les résultats obtenus lors de la simulation :

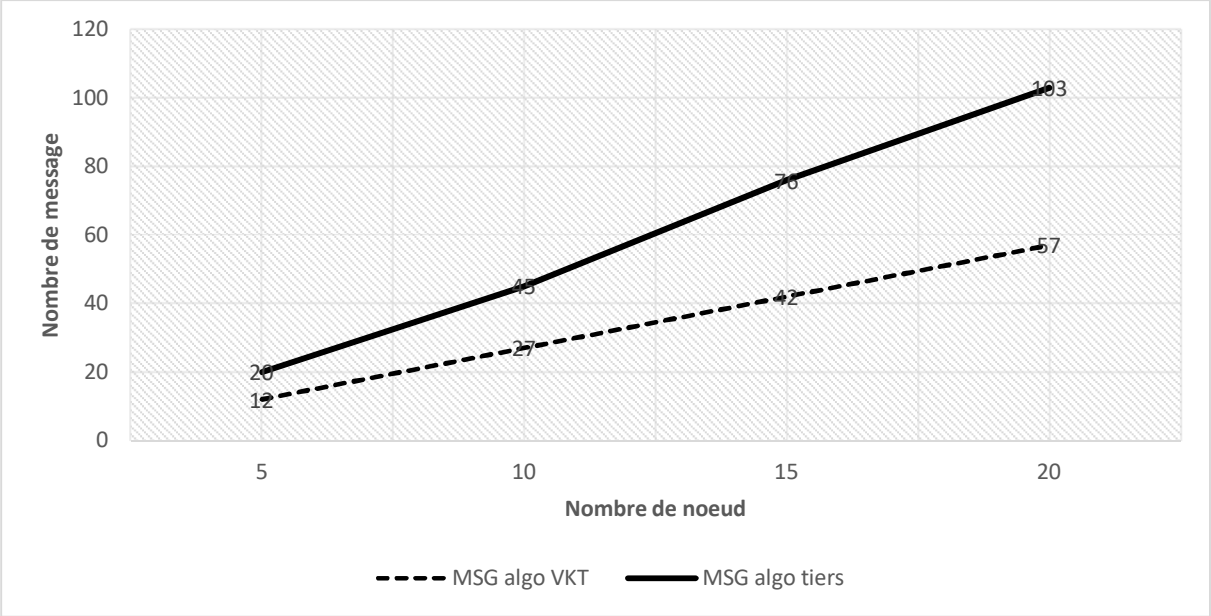

Figure 4.10 – Courbes des résultats analytiques

La courbe représenté en ligne discontinu montre le résultat de la simulation de l'algorithme de Vasudevan, Kuros et Towsely, l'autre courbe représenté en ligne gras montre aussi les résultats du protocole d'élection de Hatzis et al [24], leurs protocoles se bases sur une idée, c'est qu'à chaque fois que deux nœuds se rencontrent il se transmette leurs identificateurs et les comparent, un processus qu'on qualifie de long et de répétitifs, ce qui induit l'augmentation de la complexité de l'algorithme.

6.4 Résultat obtenu par simulation

Grace à la fonctionnalité qu'offre le simulateur et qui permet de tester les performances de l'algorithme en générant une courbe qui met en évidence le nombre message par rapport au nombre de nœuds en tenant compte de la configuration et les métriques de simulations, nous obtenant le courbe montré par la figure suivante :

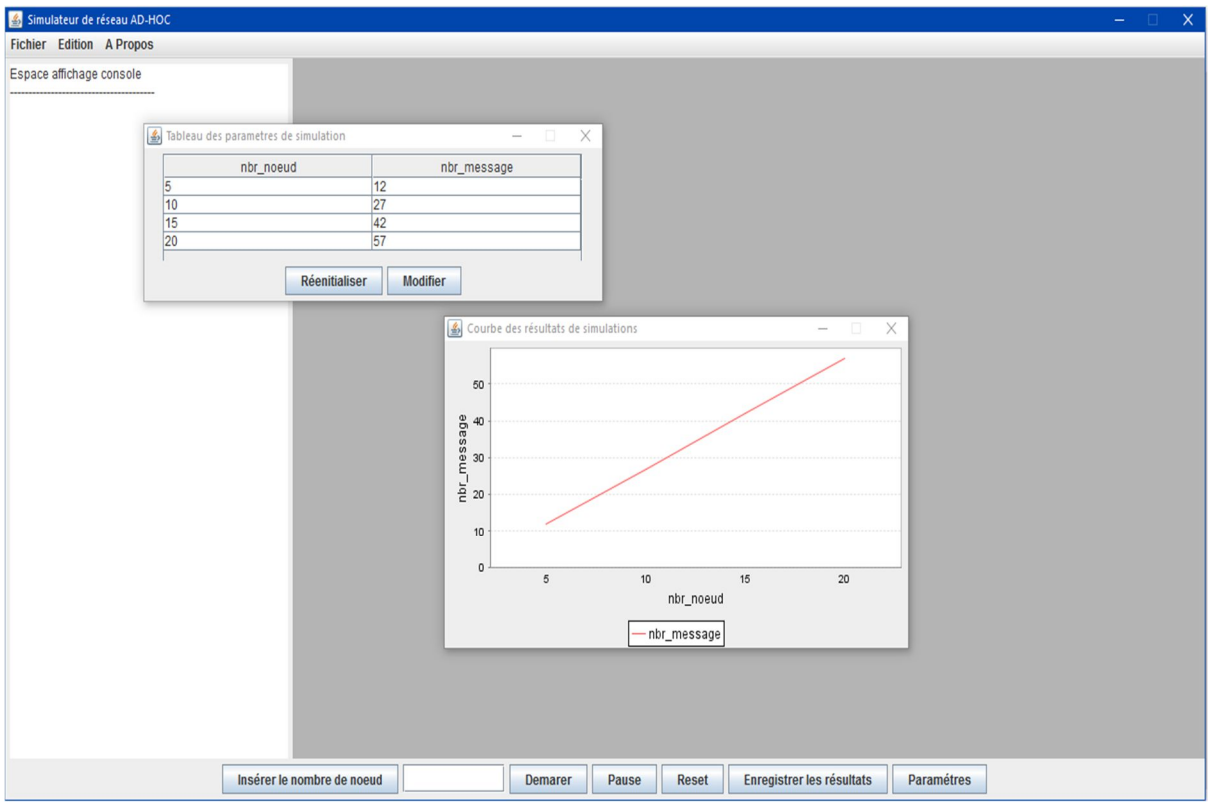

Figure 4.11 – Courbe obtenu par simulation

Comme on peut le remarquer sur le graphe, la courbe obtenue grâce au simulateur est identique à celle à notre étude analytique.

Conclusion

La phase de la réalisation est l'étape la plus importante dans le cycle de vie d'une application de simulation. Dans ce chapitre nous avons présenté cette étape cruciale de notre projet par une brève description des outils nécessaires à la simulation, nous avons implémenter le protocole de Vasudevan, Kurose et Towsely, et nous avons essayé de le comparer avec un autre selon certains paramètres qui sont les nombres de messages par rapport au nombres de nœuds.

Conclusion générale

Nous sommes parvenus, par le biais de ce projet, à réaliser sous Eclipse/java un simulateur qui permet de simuler un algorithme d'élection de leader dans les réseaux ad hoc.

L'étude que nous avons menée jusqu'ici, nous a permis d'acquérir de nouvelles connaissances sur la programmation orienté objet et distribuer sous Eclipse/java.

Le but de l'élection de leader est d'élire un chef unique parmi plusieurs processus qui sont candidats à l'élection. Le but du leader est de synchroniser les différents processus présents dans un réseau. Pour cela, nous avons choisi et étudié un algorithme qui permet de faire l'élection dans les réseaux mobiles.

Finalement, en guise de perspective, nous souhaitons proposer une amélioration au protocole implémenté dans notre travail afin d'améliorer la complexité en nombre de message et d'approfondir les simulations.

Bibliographie

[1] Sun. Mobile ad hoc networking : An essential technology for pervasive computing. In Proceedings International Conférences on Info-Tech & Info-Net, pages 316-321, 2001.

[2] http://www.guill.net/index.php?cat=3&pro=2&rfc=17 16/03/2017

[3] www.pouf.org/documentation/securite/html/node28.html 16/03/2017

[4] http://geoooffrey.free.fr/dsr_aodv/adhoc.html 21/04/2017

[5] Charles E Perkins. Ad hoc networking. Addison Wesley Professional, 2001.

[6] M Weiser. Hot topics: Ubiquitous computing. IEEE Computer, 26(10):71-72, Oct 1993.

[7] S Dolev. Self-Stabilization. The MIT Press, 2000.

[8] Bertrand Ducourthial, Introduction aux systèmes répartis, 2007.

[9] https://infoscience.epfl.ch/record/50023/files/Ds98b.pdf

[10] ecariou.perso.univ-pau.fr/cours/Sd-m1/cours-modele-Sd.pdf

[11] depinfo.cname.fr/enseignemant/CycleSpecialisation/SAR/cours-torance.pdf

[12] beru.univ-brest.fr/~singhoff/ENS/UE-Systemes-repartis/CM/sd.pdf

[13] I. Gupta, R. van Renesse, and K. P. Birman,2000, A Probabilistically Correct Leader Election Protocol for Large Groups, *Technical Report* , Cornell University

[14] R. Bakhshi, W. Fokkink, J. pang, and J. Van de Pol,c2008 "Leader Election in Anonymous Rings:Franklin Goes Probabilistic", *TCS*, Vol. 273, pp. 57-72.

[15] http://www-inf.int-evry.fr/cours/AlgoRep/Web/5.6.html

[16] D-S, HIRSCHBERG and J-B. SINCLAIR. Decentralized extrema-finding in circular configuration of processus. Commun. ACM 23, pages 627-628, November 1980

[17] J. E. Burns. A formal model for message passing systems. TR-91, Indiana University, September 1980.

[18] W.R. Franklin "On an improved algorithm for decentralized extrema finding in circular configurations of processors" Communications of the ACM 25,5 (1982) pp 336-337.

[19] G.L. Peterson. An o(n log n) unidirectional algorithm for the circular extrema problem. IEEE Transactions on Programming Languages and Systems, 4: 758-762, 1982.

[20] D. Dolev, M. Klawe, and M. Rodeh. An O(n log n) unidirectional distributed algorithm for extrema-finding in a circle. Journal of algorithms, 3:245-260, 1982.

[21] H. Attiya and J. Welch, *Distributed Computing: Fundamentals, Simulations and Advance Topics*, John Wiley & Sons inc., 2004, chap. 3

[22] N. Santoro, *Design and Analysis of Distributed Algorithms*, Wiley, 2006.

[23] https://www.cs.tau.ac.il/~afek/p66-gallager.pdf

[24] K-P. Hatzis, G-P. Pentaris, P-G. Spirakis, V-T. Tampakas, and R.B. Tan. Fundamental control algorithms in mobile networks. Proc. 11th Annual ACM Symposium on Parallel Algorithms and Architectures, pages 251-260, 1999.

[25] Sudarshan Vasudevan, Jim Kurose, Don Towsely, Desing and Analyse of a leader Election Algorithme for Mobile Ad Hoc Networks,University of Massachusettes, Amherst, pages 1-11, 2004

[26] N.Malpani, J-L. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. Fourth International Workshop on Discrete algorithms and Methods for mobile computing and communications, Boston, MA, pages 96-103, August 2000.

[27] A. Velayutham and S. Chaudhuri. Analysis of a leader election algorithm for mobile ad hoc networks. Technical report. Iowa State University, 2003

[28] G. Montcouquiol (IUT Orsay), 2006-2007, http://www.math.u-psud.fr/~montcouq/Enseignement/Apprentis/arbre.pdf

[29] Laurant philippe, Univ-Franche-Comté, Algorithmiqe Distribuée, Détection de la terminaison, 2013-2014. http://members.femto-st.fr/sites/femto-st.fr.Laurent-Philippe/files/content/lessons/terminaison.pdf

[30] Sacha Krakowiak, univ- Josef Fourier, Introduction aux algorithme répartis 2003-2004

[31] http://dictionnaire.sensagent.leparisien.fr/Eclipse%20(logiciel)/fr-fr/ 02/06/2017

[32] https://www.java.com/fr/download/faq/whatis_java.xml 05/05/2017

[33] http://www.journaldunet.com/solutions/pratique/dictionnaire-duwebmastering/technologies-langages/19494/java-definition.html 14/03/2017

[34] http://jbotsim.sourceforge.net/ 02/06/2017

[35] Julien CARSIQUE, Le routage dans les réseaux mobiles ad hoc, 2002-2003

Résumé

Le problème d'élection constitue une brique dans les systèmes distribués, filaires ou non filaires. La définition classique de ce problème est d'élire finalement un leader unique d'un ensemble de nœuds. Cependant, dans les réseaux mobiles ad-hoc caractérisés essentiellement par des changements de topologies induites par la mobilité des sites rendent le processus d'élection plus difficile.

Nous avons donné un aperçu des travaux existant dans la littérature, et une présentation des différentes solutions pour le problème d'élection dans les systèmes dynamiques, puis nous avons choisi et simulé un des protocoles intéressants dans ce domaine ensuite nous avons terminé par le test de performance de l'algorithme choisi puis comparer les résultats par rapport un autre protocoles d'élection.

Mot clés : Election de leader, Réseau ad-hoc, protocoles

Abstract

The problem of election constitutes a brick in distributed systems, wired or non-wired. The classic definition of this problem is to finally elect a single leader of a set of nodes. However, in ad hoc mobile networks characterized mainly by changes in topologies induced by site mobility, the election process becomes more difficult.

We gave an overview of the literature and a presentation of the different solutions for the problem of election in dynamic systems, then we chose and simulated one of the interesting protocols in this field and then we finished the test of Performance of the selected algorithm and then compare the results against another election protocol.

Keywords : Leader election, Ad hoc networks, protocols.