

Mémoire de Master Recherche

Contribution à l'étude et la définition des NCIPs (Neighborhood and Context Interaction Primitives) dans un contexte MANet

Etudiant: Jérémie ALBERT
Encadrant: Serge CHAUMETTE

7 juin 2007



Table des matières

1	Introduction	3
1.1	Contexte	3
1.2	Objectifs	3
1.3	Etat de l'art : middlewares et systèmes d'exploitation existants	4
2	Définitions générales et exemples	6
2.1	Terminal mobile communicant	6
2.2	Zone de couverture, portée	6
2.3	Voisin unilatéral	6
2.4	Voisins réciproques	6
2.5	Voisin unilatéral strict	7
2.6	Connexion	7
2.7	Groupe	8
2.8	<i>Tuple</i>	8
2.9	<i>Tuple Space</i>	8
2.10	Traitement passif	9
3	Propriétés éventuelles du réseau et des terminaux	10
3.1	Liste des propriétés éventuelles du réseau et des terminaux	10
3.2	Conséquences de la vérification de ces propriétés	10
4	Classification, contexte d'<i>applicabilité</i> et résultat des NCIPs (Neighborhood and Context Interaction Primitives) identifiées	13
4.1	Réponse passive à l'évolution du voisinage	14
4.2	Action sur le voisinage	18
4.3	Opérations locales destinées à faciliter les interactions dans les MANets	24
4.4	Opérations complexes qui ne nécessitent pas une action sur le voisinage	25
4.5	Opérations complexes qui nécessitent une action sur le voisinage	28
5	Algorithmes reposant sur les NCIPs	34
5.1	Présentation des algorithmes	34
5.2	Implémentation de l'algorithme de synchronisation	36
6	Conclusion	38
A	Réponse passive à l'évolution du voisinage	40
B	Action sur le voisinage	41
C	Opérations locales destinées à faciliter les interactions dans les MANets	42
D	Opérations complexes qui ne nécessitent pas une action sur le voisinage	43
E	Opérations complexes qui nécessitent une action sur le voisinage	44

Table des figures

2.1	A voisin unilatéral de B	6
2.2	A et B voisins réciproques	7
2.3	A voisin unilatéral strict de B	7
2.4	Exemples de groupes	8
2.5	Tuple Space local de A	9
2.6	<i>Tuple Space distribué</i>	9
4.1	Détection de l'apparition d'un voisin unilatéral	15
4.2	Détection de la disparition d'un voisin unilatéral strict	16
4.3	Détection de la transformation d'un voisin unilatéral strict en voisin réciproque	19
4.4	Détection de la transformation d'un voisin réciproque en voisin unilatéral strict	20
4.5	Détection de la transformation d'un voisin non réciproque à portée en voisin réciproque	20
4.6	Détection de la disparition d'un voisin réciproque	21
4.7	Détection de la disparition d'un voisin réciproque	22
4.8	Détection de l'ensemble des terminaux à portée	29
4.9	Détection de l'ensemble des terminaux à portée	29
4.10	Détection de l'apparition d'un voisin réciproque	30
4.11	Détection de l'apparition d'un voisin réciproque	30
4.12	A Randomized Algorithm to Obtain Rendezvous	32
5.1	Schéma temporel de l'algorithme de synchronisation	35
5.2	Un capteur Crossbow Mica2	37

Chapitre 1

Introduction

1.1 Contexte

Ce travail de recherche s'est déroulé dans le thème Systèmes et Objets Distribués (SOD) de l'équipe Langages, Systèmes et Réseaux (LSR) du LaBRI. Le principal objectif de cette équipe est de contribuer à la conception, au développement et à la validation d'outils et d'environnements d'exécution permettant de répondre aux objectifs des utilisateurs et des programmeurs dans un contexte réseau et en particulier MANet [19]. L'objectif de l'utilisateur est d'exécuter son application de façon transparente sur la meilleure configuration disponible, c'est à dire avec un maximum de simplicité, d'efficacité et de sécurité. L'objectif du programmeur est de développer son application de la façon la plus simple possible avec le résultat le plus pertinent possible.

Les MANets (Mobile Ad hoc Networks) sont des réseaux ad hoc constitués de terminaux mobiles qui peuvent être très hétérogènes en termes de technologie radio, de système d'exploitation, de capacité de calcul et de mémoire. Ils peuvent être composés de dispositifs tels que des capteurs, des téléphones mobiles, des PDAs, des ordinateurs et de tout autre dispositif doté d'une interface de communication. De cette grande hétérogénéité découle une grande diversité de problèmes tels que ceux liés à la mobilité (connaissance du voisinage à un instant donné), aux technologies radio (capacité à communiquer sur une fréquence unique) ou encore au manque de ressources (mémoire).

1.2 Objectifs

Les problèmes posés par la définition même des MANets et par leur hétérogénéité n'ont à ce jour pas été réellement pris en compte. En effet, les systèmes existants tels que Maté [8] ou Squawk [17] sont pour la plupart assimilables à des couches logicielles servant plutôt à masquer le côté mobile du dispositif qu'à les présenter au programmeur sous une forme exploitable..

L'objectif de ce travail de recherche est l'étude et la définition des primitives minimales ou canoniques dans les MANets. Nous considérons qu'une primitive est minimale si elle ne peut pas être décomposée en plusieurs primitives. L'objectif à terme de ce travail de recherche est la création d'un environnement de développement et d'exécution pour la classe des réseaux mobiles de type MANet. La particularité de ces travaux réside dans le fait que l'on considère seulement des relations deux à deux, c'est à dire qu'un nœud ne considère que ses voisins immédiats.

Pour commencer, nous avons défini un certain nombre de notions utilisées dans les réseaux MANets. Puis, à partir d'une recherche bibliographique, nous avons extrait les primitives utilisées (souvent de manière implicite pour les programmeurs) dans les systèmes existants. Nous avons ensuite complété cette liste avec de nouvelles primitives qui n'étaient ni définies, ni implicitement présentes dans aucun des systèmes étudiés en prenant soin de justifier leur caractère utile à travers un ou plusieurs exemples d'algorithmes les nécessitant. Pour finir, nous avons tenté d'implémenter l'une de ces primitives sur des téléphones P910i disposant d'une machine virtuelle Java et de la technologie

radio Bluetooth [2], et sur des capteurs Crossbow [3] équipés de processeurs Mica2 [4] et fonctionnant sous TinyOS [10]. De plus, tout au long de ce travail de recherche, nous avons identifié un ensemble de propriétés éventuelles du réseau et des terminaux qui influent sur la définition et les résultats de certaines primitives selon qu'elles sont satisfaites ou non. Certaines de celles-ci ne sont même réalisables que si une ou plusieurs de ces propriétés sont vérifiées. Nous définissons donc, pour chaque primitive, l'environnement minimal (c'est à dire l'ensemble des propriétés qui doivent être vérifiées) qui en permet la réalisation.

1.3 Etat de l'art : middlewares et systèmes d'exploitation existants

Afin d'avoir un aperçu des primitives des systèmes existants, nous avons étudié TinyOS [10], TOTA [12] et Zigbee [7].

1.3.1 TinyOS

TinyOS [10] est un système d'exploitation open-source conçu pour les réseaux de capteurs sans fil. Il s'appuie sur un paradigme événementiel et propose à l'utilisateur une utilisation simple des procédures de communication.

Les primitives que nous avons pu isoler à partir de la documentation de TinyOS [9, 11] sont des primitives minimales de communication, c'est à dire des fonctions d'envoi et de réception de messages.

1.3.2 Tuples On The Air (TOTA)

TOTA [12] est un intergiciel supportant les applications qui s'adaptent à leur contexte dans les réseaux mobiles de type MANet. TOTA [12] repose sur des *tuples* distribués pour la représentation des informations contextuelles. L'intergiciel propage des *tuples* sur le réseau sur la base des schémas spécifiques à l'application et met à jour les structures distribuées résultantes en fonction des changements dans le réseau. Les composants de l'application peuvent localement récupérer ces structures et les exploiter pour obtenir des informations contextuelles. Les définitions de *Tuple*, *Tuple Space local* et *Tuple Space distribué* sont données dans le chapitre suivant.

L'API de TOTA [12] fournit les primitives d'interaction suivantes :

- public void inject (TotaTuple tuple) ;
- public Vector read (Tuple template) ;
- public Vector readOneHop (Tuple template) ;
- public Tuple keyrd (Tuple template) ;
- public Vector keyrdOneHop (Tuple template) ;
- public Vector delete (Tuple template) ;
- public void subscribe (Tuple template, ReactiveComponent comp, String rct) ;
- public void unsubscribe (Tuple template, ReactiveComponent comp) ;

Chacune de ces primitives a été reprise et classée dans ce travail de recherche. Elles sont détaillées dans le chapitre 4 qui est consacré à l'étude des primitives.

1.3.3 Zigbee

Zigbee [7] est une technologie de communication sans fil conçue pour réseaux de capteurs. La couche radio de cette technologie est basée sur la spécification 802.15.4 de l'IEEE (Institute of Electrical and Electronics Engineers). Elle permet d'effectuer un contrôle bas niveau sur la couche de communication grâce à des commandes AT, dans le style des commandes qui permettent de dialoguer avec un modem en mode ASCII. Zigbee permet par exemple d'obtenir la puissance d'un

signal reçu grâce à la commande ATDB. Ce type de primitive de très bas niveau permet un contrôle très fin des communications entre les terminaux.

Chapitre 2

Définitions générales et exemples

2.1 Terminal mobile communicant

On appelle terminal mobile communicant tout dispositif doté d'une interface de communication. Dans la suite de ce mémoire, nous parlerons de terminal. Les téléphones mobiles, PDA, capteurs, ordinateurs sont, en ce sens, des terminaux.

2.2 Zone de couverture, portée

Soit A un terminal. La zone de couverture de A est l'ensemble des points de l'espace depuis lesquels les données émises par A peuvent être reçues. Si un terminal est dans la zone de couverture de A on dit aussi qu'il est à portée de A.

2.3 Voisin unilatéral

Soient A et B deux terminaux. A est dit voisin unilatéral de B si et seulement si B est à portée de A. On dit aussi que A est voisin visible de B. Cette définition est extraite de [18]. Elle ne fait aucune supposition quand à la visibilité de B par A.

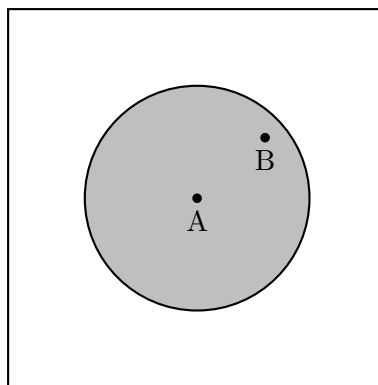


FIG. 2.1 – A voisin unilatéral de B

2.4 Voisins réciproques

Soient A et B deux terminaux. A et B sont dits voisins ou encore voisins réciproques si et seulement si ils sont mutuellement voisins unilatéraux. Cette définition est extraite de [18].

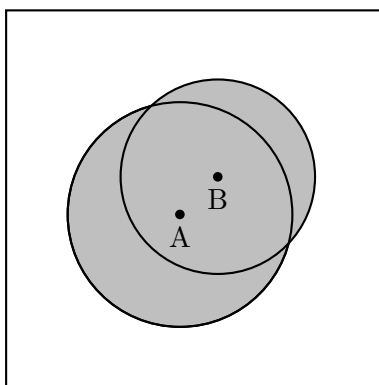


FIG. 2.2 – A et B voisins réciproques

2.5 Voisin unilatéral strict

Soient A et B deux terminaux. A est dit voisin unilatéral strict de B si et seulement si A est voisin unilatéral de B sans qu'ils soient voisins réciproques. Cette définition est extraite de [18].

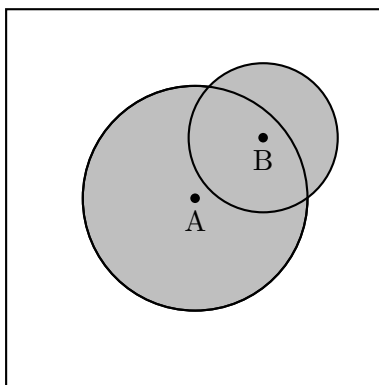


FIG. 2.3 – A voisin unilatéral strict de B

2.6 Connexion

Soient A et B deux terminaux. Il existe une connexion entre A et B si et seulement si :

- A et B sont voisins réciproques ;
- On peut détecter en temps réel et de manière passive le fait que A et B ne sont plus voisins réciproques.

Nous ne savons pas, pour l'instant, mettre en œuvre ce type de définition si le terminal ne dispose que d'une seule interface de communication.

Une autre définition possible ne présentant pas ce problème est la suivante :

Soient A et B deux terminaux. Il existe une connexion entre A et B si et seulement si :

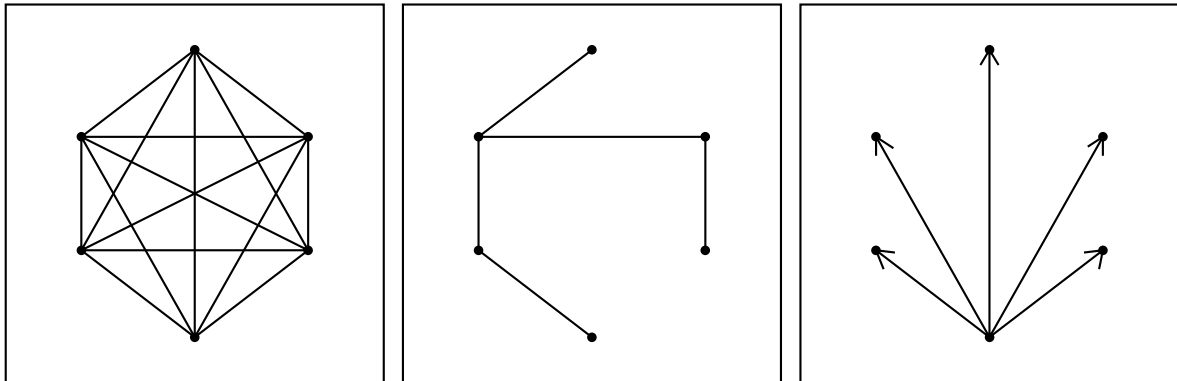
- A et B sont voisins réciproques ;
- On peut déterminer à tout instant (donc activement) s'ils sont ou non toujours voisins réciproques.

On dit aussi que A et B sont connectés.

2.7 Groupe

Un groupe est un ensemble de terminaux qui partagent une ou plusieurs caractéristiques communes. La définition générique d'un groupe ne pouvant être plus précise, nous allons en donner quelques exemples :

1. Ensemble de terminaux tous voisins réciproques les uns des autres.
2. Ensemble de terminaux voisins réciproques dont l'ensemble des connexions forme un arbre couvrant du graphe représentant le réseau.
3. Ensemble de terminaux qui possèdent un même voisin unilatéral.



(a) Chacun est voisin réciproque avec tous les autres
(b) Chacun possède une connexion qui fait partie d'un même arbre couvrant
(c) Chacun possède le même voisin unilatéral

FIG. 2.4 – Exemples de groupes

Sur la figure 2.4c, la flèche indique que le terminal à l'origine de celle-ci est voisin unilatéral du terminal qui se trouve à son extrémité. Ainsi, sur cette figure, 5 terminaux possèdent le même voisin unilatéral, autrement dit ils peuvent recevoir des messages de ce terminal.

2.8 Tuple

Un *tuple* est un objet contenant des données et des règles de propagation. Il se propage de *Tuple Space* en *Tuple Space* en respectant ses règles de propagation.

2.9 Tuple Space

Un *Tuple Space* est un espace mémoire où sont gérés les *tuples*.

2.9.1 Tuple Space local

Le *Tuple Space local* est le *Tuple Space* qui se trouve physiquement sur la machine locale (cf. fig. 2.5).

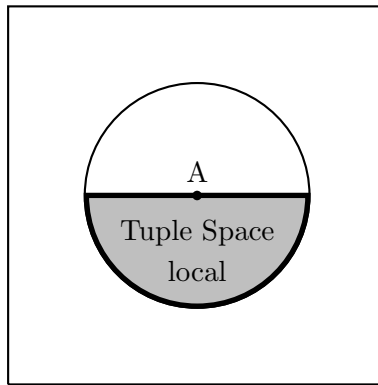


FIG. 2.5 – Tuple Space local de A

2.9.2 *Tuple Space distribué*

Le *Tuple Space distribué* (cf. fig. 2.6) est l'union du *Tuple Space local* et des *Tuple Spaces* qui se trouvent sur les autres terminaux du réseau.

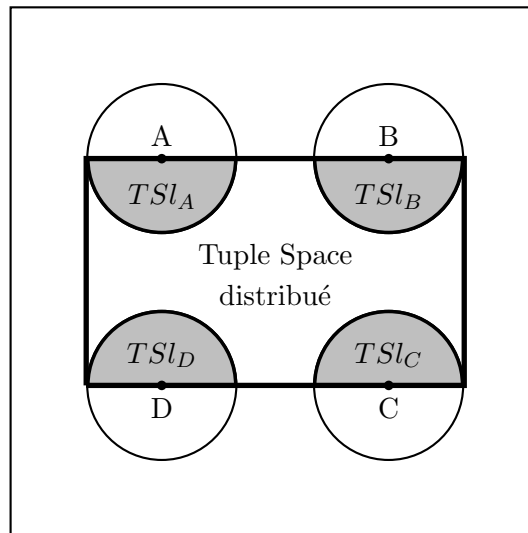


FIG. 2.6 – *Tuple Space distribué*

2.10 Traitement passif

On appelle traitement passif d'un terminal, tout traitement qui ne monopolise pas le terminal.

Chapitre 3

Propriétés éventuelles du réseau et des terminaux

Les propriétés éventuelles du réseau et des terminaux sont des propriétés qui influent sur l'existence et les résultats de certaines des primitives définies dans le chapitre suivant. Ces propriétés ne sont pas des suppositions sur lesquelles s'appuient l'ensemble des primitives. Elles permettent simplement de préciser quelles primitives sont réalisables et quelles primitives ne le sont pas en fonction du contexte. De plus, certaines primitives voient leur résultat changé en fonction de la vérification ou non de ces propriétés.

3.1 Liste des propriétés éventuelles du réseau et des terminaux

La liste des propriétés qui suit n'est certainement pas exhaustive. Celle-ci a été complétée tout au long de l'étude des primitives des systèmes existants. Il est probable que lors d'une mise en pratique de ce travail de recherche cette liste soit complétée. Voici la liste :

- P_1 : Lors d'une transmission, l'ordre des paquets est conservé ;
- P_2 : Chaque terminal possède un identifiant unique ;
- P_3 : Les terminaux peuvent se mettre en écoute de manière passive ;
- P_4 : Les terminaux peuvent modifier la puissance d'émission de leur(s) radio(s) ;
- P_5 : Les terminaux sont capables de communiquer en pair à pair ;
- P_6 : Les terminaux possèdent plusieurs interfaces de communication ;
- P_7 : Les terminaux régulièrement annoncent leur présence.

3.2 Conséquences de la vérification de ces propriétés

Les propriétés listées ci-dessus ont, lorsqu'elles sont vérifiées, les conséquences respectives suivantes :

- P_1 : **Rappel** : Lors d'une transmission, l'ordre des paquets est conservé.

Explication : Nous considérons ici les transmissions sans intermédiaires, c'est à dire sans routage. L'ordre des paquets successifs envoyés par un terminal à un autre terminal ne peut être différent entre la source et la destination. En effet, les messages circulent sur des ondes qui possèdent une vitesse constante dans un même milieu (environ 299 700 km/s dans l'air). Si on considère des messages envoyés à 100 mètres, ceux-ci mettent dans l'air moins de $2 \cdot 10^{-4}$ ms pour parcourir cette distance. Il est évident que le milieu de propagation entre deux messages successifs ne peut varier de manière suffisamment importante pour qu'un second message, envoyé juste après l'autre, puisse le dépasser.

Conséquence : Il n'est pas nécessaire de faire un contrôle sur l'ordre des paquets à l'arrivée. Cette considération est particulièrement intéressante lorsque l'on souhaite des communications fiables, c'est à dire qui mettent en œuvre des algorithmes de recouvrement des paquets perdus.

P_2 : Rappel : Chaque terminal possède un identifiant unique.

Explication : L'identifiant unique peut-être par exemple l'adresse physique de l'interface utilisée pour les communications. L'objectif est de faire en sorte qu'aucun autre terminal n'ait le même identifiant.

Conséquence : Lors de l'envoi d'un message, on est capable de l'adresser à un terminal unique. De même, lors de la réception d'un message, on peut identifier la source. Nous ne garantissons pas ici une sécurité des échanges. En effet, rien n'empêche un terminal de *sniffer* le réseau ou un autre d'usurper une identité. Un grand nombre d'algorithmes nécessitent la présence d'identités distinctes pour chaque terminal. Il est aisé de simuler logiquement la création d'une identité avec une grande probabilité d'unicité grâce à un algorithme tel que AES [15] ou RSA [16].

P_3 : Rappel : Les terminaux peuvent se mettre en écoute de manière complètement passive.

Explication : Ce que l'on entend par écoute passive est la capacité d'un terminal à écouter sur son (ou ses) interface(s) de communication tout en continuant à effectuer d'autres opérations.

Conséquence : Les algorithmes qui doivent effectuer des opérations tout en attendant l'arrivée de messages peuvent être implémentés.

P_4 : Rappel : Les terminaux peuvent modifier la puissance d'émission de leur(s) radio(s).

Explication : Les terminaux n'ont pas toujours besoin d'utiliser leur radio avec toute la puissance disponible, si les informations n'ont pas besoin d'être envoyées à une distance correspondant à la portée maximale. Cette propriété permet de supposer que le terminal est capable de définir la puissance avec laquelle il souhaite utiliser son (ou ses) interfaces de communication.

Conséquence : Le contrôle de puissance et la gestion d'énergie sont des problèmes importants lorsque l'on s'intéresse à des terminaux mobiles communicants. Une réduction de la puissance d'émission permet non seulement de réduire les collisions et conflits radio lorsque les terminaux se trouvent dans une zone dense mais aussi de réaliser une gestion plus souple de l'énergie.

P_5 : Rappel : Les terminaux sont capables de communiquer en pair à pair ;

Explication : Certaines technologies radio n'autorisent des communications entre deux terminaux que si l'un des deux est le maître et l'autre l'esclave. Or, la relation de voisinage pair-à-pair est un mode fondamental dont on souhaite disposer dans les réseaux MANets. Afin de pouvoir simuler ce type de communication, chaque terminal doit être capable de basculer de son rôle de maître à son rôle d'esclave et inversement. Cette capacité dépend de chaque terminal.

Conséquence : Les périphériques sont capables de communiquer avec l'ensemble de leurs voisins réciproques.

P_6 : Rappel : Les terminaux possèdent plusieurs interfaces de communication.

Explication : L'augmentation du nombre de fonctions et de la complexité des terminaux nous permet d'envisager que les terminaux possèdent plusieurs interfaces de communication.

Conséquence : Posséder plusieurs interfaces de communication permet par exemple d'être capable de recevoir des données tout en envoyant et/ou d'utiliser plusieurs interfaces de communication en même temps (sur des fréquences différentes bien entendu). Cette supposition influe aussi sur la définition de connexion vue précédemment (cf. section 2.6 page 7). Les primitives qui reposent sur la définition de connexion ont donc elles aussi plusieurs significations possibles en fonction du contexte.

P_7 : **Rappel** : Les terminaux annoncent leur présence régulièrement.

Explication : Chaque terminal annonce sa présence à l'ensemble des terminaux à sa portée de manière régulière. Le temps entre deux annonces successives peut-être défini en fonction du contexte. Une étude sur le délai optimal entre deux signalements de présence successifs a été réalisée en [18].

Conséquence : Chaque terminal peut détecter la présence de ses voisins unilatéraux.

Certaines primitives définies ci-après sont dépendantes de la vérification ou non des propriétés présentées dans cette section.

Chapitre 4

Classification, contexte d'*applicabilité* et résultat des NCIPs (Neighborhood and Context Interaction Primitives) identifiées

Dans chacune des figures ci-après, le terminal de référence est le terminal nommé A.

Nous listons ici l'ensemble des primitives identifiées en précisant pour chacune les caractéristiques suivantes :

Catégorie : La catégorie correspond à la (ou aux) manière(s) dont peut-être utilisée la primitive.

Elle peut être :

- Active : La primitive retourne immédiatement un résultat.
- Passive : La primitive retourne un résultat lorsqu'elle en a un (sauf si elle est définie avec timeout auquel cas elle retourne obligatoirement un résultat au bout d'un certain temps).

Contexte d'*applicabilité* : C'est l'ensemble des conditions sur les terminaux et le réseau afin que la primitive ait un sens, c'est à dire que la primitive ne peut retourner un résultat en dehors de ce contexte.

Résultat optimal : C'est le meilleur résultat possible, le résultat vers lequel on souhaite tendre.

Résultats dégradés : Ce sont les résultats obtenus "par dégradation" du résultat optimal.

Proposition d'implémentation : Un exemple d'implémentation possible de cette primitive (version optimale ou dégradée).

Résultat de l'implémentation proposée : Il est égal au résultat optimal dans le meilleur des cas, sinon à un des résultats dégradés.

Toutes les primitives passives peuvent-être implémentées avec ou sans timeout. Dans la suite du document, nous différencions simplement les primitives actives des primitives passives.

Pour chaque primitive se pose la question de la stabilité du résultat, c'est à dire du temps pendant lequel le résultat retourné reste valide. Celui-ci dépend du réseau considéré. Dans un réseau MANet sans propriétés particulières, la stabilité du résultat est nulle pour toutes les primitives de détection de changement de contexte (apparition d'un voisin, disparition d'un voisin, etc...). Par exemple, lors de la détection de l'apparition d'un voisin à un instant t , rien ne garantit la présence du voisin à l'instant $t + \epsilon$. Cependant, si la primitive a pour objet de savoir si on a déjà été en présence d'un autre terminal, alors le résultat de cette même primitive a une stabilité infinie. L'étude de la stabilité du résultat sera l'objet de futurs travaux de recherche.

L'objectif final de la définition de ces primitives est leur intégration dans un intergiciel qui permettra non pas de masquer la mobilité des terminaux mais de fournir des primitives simples et canoniques et une connaissance du contexte et du voisinage aux développeurs et aux utilisateurs.

4.1 Réponse passive à l'évolution du voisinage

Dans cette catégorie, se trouvent toutes les primitives minimales qui ne nécessitent pas une action sur le voisinage. Tout terminal étant dans l'incapacité d'envoyer des messages peut malgré tout exécuter ces primitives.

4.1.1 Détection de l'apparition d'un voisin unilatéral

Catégorie : Cette primitive est une primitive passive.

Explication : Cette primitive permet pour un terminal A de détecter de manière passive l'apparition d'un voisin unilatéral B, c'est à dire de détecter qu'il devient à portée de B. On peut identifier deux variantes de cette primitive :

- Si la propriété P_2 : "chaque terminal possède un identifiant unique" (identifiant généré grâce à un algorithme tel que AES [15] ou RSA [16] si nécessaire) est satisfaite, alors on peut considérer que non seulement on détecte un nouveau voisin mais qu'en plus celui-ci peut être identifié.
- Au contraire, si cette propriété n'est pas vérifiée, alors il est impossible de différencier les terminaux les uns des autres et la détection d'un nouveau voisin est donc impossible. Cette primitive n'indique donc qu'une seule information : la présence ou non d'au moins un voisin.

Contexte d'applicabilité : Deux terminaux A et B tels que A n'est pas à portée de B.

Résultat optimal : Cette primitive retourne vrai si et seulement si B est voisin unilatéral de A (**TRUE** \Leftrightarrow **B voisin unilatéral de A**).

Résultats dégradés :

- **TRUE** \Rightarrow **B voisin unilatéral de A**
- **B voisin unilatéral de A** \Rightarrow **TRUE**

Proposition d'implémentation : La détection de l'apparition d'un voisin unilatéral peut être faite de plusieurs manières. En Bluetooth, il s'agit de lancer le processus Inquiry afin de découvrir les voisins unilatéraux. Le principe de cette méthode est de *scanner* la bande de fréquences afin d'intercepter des messages d'annonce et ainsi d'identifier un terminal. Cette méthode fonctionne car un périphérique Bluetooth qui souhaite être découvert s'annonce régulièrement sur le réseau excepté lorsque lui-même exécute ce processus. Ainsi, si plusieurs terminaux lancent ce processus au même moment, ils ne se détectent pas. Pour les autres technologies de communication, on peut imaginer que les terminaux qui souhaitent être identifiés envoient un message de manière régulière sur une fréquence prédéfinie afin d'être détectés [18]. Quelle que soit la technologie radio utilisée, le principe reste le même, il s'agit de *sniffer* les paquets circulant sur le réseau.

Résultat de l'implémentation proposée : Que l'on considère un réseau fiable ou non fiable, la primitive retourne vrai si B devient voisin réciproque de A (**TRUE** \Rightarrow **B voisin unilatéral de A**).

Remarque : Un terminal ne peut pas déterminer si le contexte d'*applicabilité* est vérifié ou pas. Cette primitive ne fonctionne donc que si elle est exécutée en *tâche de fond* dès l'initialisation du terminal.

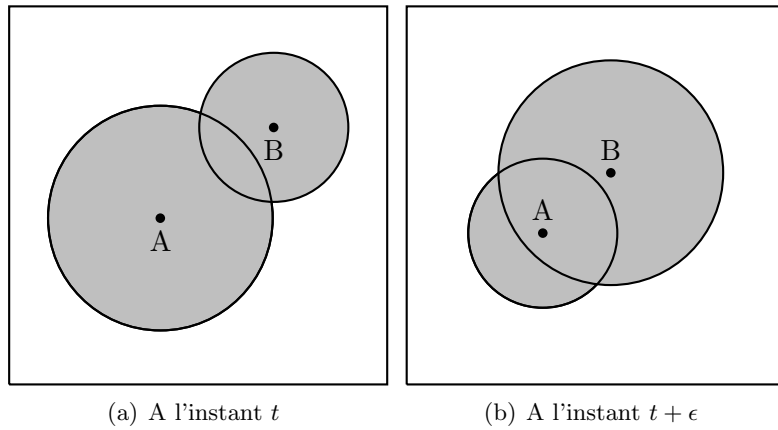


FIG. 4.1 – Détection de l'apparition d'un voisin unilatéral

4.1.2 Détection de la disparition d'un voisin unilatéral strict

Catégorie : Cette primitive est une primitive passive.

Explication : Cette primitive permet la détection de manière passive par un terminal A de la disparition d'un voisin unilatéral strict B, c'est à dire que A n'est plus à portée de B (B n'est jamais à portée de A). La détection de la disparition d'un voisin unilatéral strict dépend des suppositions faites sur le réseau. Lorsqu'un terminal A ne reçoit plus de signal d'un terminal B, cela peut être dû soit au fait que le terminal B n'émet plus, soit au fait que le terminal A est passé hors de portée du terminal B (cf. fig. 4.2). La disparition d'un voisin unilatéral strict n'est donc détectable que si celui-ci est supposé s'annoncer régulièrement.

Contexte d'applicabilité : Deux terminaux A et B tels que :

- B est voisin unilatéral strict de A.
- B annonce régulièrement sa présence (propriété P_7 page 12).

Résultat optimal : Cette primitive retourne vrai si et seulement si B n'est plus voisin unilatéral strict de A (**TRUE** \Leftrightarrow **B n'est plus voisin unilatéral strict de A**).

Résultats dégradés :

- **TRUE** \Rightarrow **B n'est plus voisin unilatéral strict de A**
- **B n'est plus voisin unilatéral strict de A** \Rightarrow **TRUE**

Proposition d'implémentation : Le terminal A écoute en permanence le signalement de présence du terminal B. Lorsqu'au bout d'un certain temps prédéfini plus aucun message n'a été reçu alors le terminal n'est plus considéré comme voisin unilatéral strict. Un périphérique Bluetooth configuré en mode découvrable fait partie des terminaux dont la disparition en tant que voisin unilatéral strict peut-être signalée.

Résultat de l'implémentation proposée :

- Si on considère un réseau fiable, c'est à dire sans perte de paquets, alors la disparition d'un voisin unilatéral strict n'est détectée qu'une fois le délai correspondant au délai entre deux signalement dépassé. Cette détection a donc lieu en temps réel pour un délai entre deux signalements qui tend vers 0.
On a donc **TRUE** \Leftrightarrow **B n'est plus voisin unilatéral strict de A**
- Si on considère un réseau non fiable, alors le résultat est le même que précédemment, si ce n'est qu'en plus, la disparition peut-être détectée alors qu'elle n'est pas réelle.
On a donc **B n'est plus voisin unilatéral strict de A** \Rightarrow **TRUE**

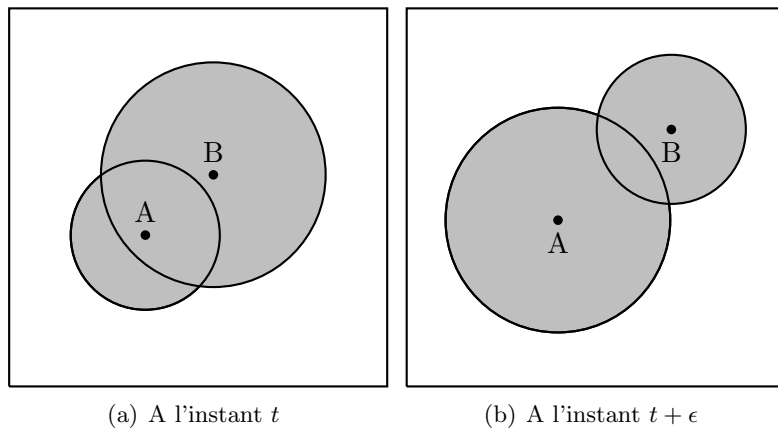


FIG. 4.2 – Détection de la disparition d'un voisin unilatéral strict

4.1.3 Calcul de la puissance du signal reçu

Catégorie : Cette primitive est une primitive passive.

Explication : Cette primitive correspond à la commande ATDB de la technologie Zigbee [7]

Contexte d'applicabilité : Deux terminaux A et B tels que B soit voisin unilatéral de A.

Résultat optimal : Cette primitive retourne un entier correspondant à la puissance du signal reçu de B en dBm.

4.1.4 Réception d'un message quelle que soit l'adresse de destination

Catégorie : Cette primitive est une primitive passive.

Contexte d'applicabilité : Aucun contexte particulier.

Résultat optimal : Cette primitive retourne le premier message qui parvient au terminal.

Résultat dégradé : Cette primitive retourne un message parvenu au terminal sans garantie sur le fait qu'il s'agit du premier.

Proposition d'implémentation : Le terminal *sniffe* le réseau. Si les transmissions de messages sont effectuées sur plusieurs fréquences, le terminal choisit la fréquence à écouter de manière aléatoire.

Résultat de l'implémentation proposée :

- Si les transmissions s'effectuent sur une unique fréquence et que le réseau est fiable, alors le premier message envoyé par un terminal voisin unilatéral de A est reçu par celui-ci (résultat optimal)
- Si les transmissions s'effectuent sur plusieurs fréquences ou/et que le réseau n'est pas fiable, alors seulement certains des messages qui parviennent à A sont reçus. On ne peut pas garantir ici que tous les messages sont reçus, on ne peut donc pas garantir que le premier est reçu (résultat dégradé).

4.1.5 Réception d'un message dont l'adresse de destination est l'adresse de broadcast ou l'adresse locale

Catégorie : Cette primitive est une primitive passive.

Contexte d'applicabilité : Chaque terminal possède un identifiant unique.

Résultat optimal : Cette primitive retourne le premier message qui parvient au terminal et dont l'adresse de destination est l'adresse de broadcast ou l'adresse locale.

Résultat dégradé : Cette primitive retourne un message parvenu au terminal et dont l'adresse de destination est l'adresse de broadcast ou l'adresse locale, sans garantie qu'il s'agisse du premier.

Proposition d'implémentation : Le terminal écoute sur son (ou ses) interface(s) de communication.

Résultat de l'implémentation proposée :

- Si les transmissions sont fiables et qu'elles s'effectuent sur une seule fréquence, le premier message envoyé par un terminal voisin unilatéral de A à A est reçu par A (résultat optimal).
- Si les transmissions s'effectuent sur plusieurs fréquences ou/et que le réseau n'est pas fiable, seuls certains des messages envoyés par un terminal voisin unilatéral de A à A sont reçus par A. On ne peut pas garantir ici qu'ils sont tous reçus par A, on ne peut donc pas garantir que A reçoive le premier (résultat dégradé).

Remarque : Cette primitive est une primitive classique de réception de message. Elle est implémentée dans la plupart des systèmes bas niveau rencontrés, comme TinyOS [10] par exemple.

4.1.6 Réception d'un message dont l'adresse de destination n'est ni l'adresse de broadcast, ni l'adresse locale

Catégorie : Cette primitive est une primitive passive.

Contexte d'applicabilité : Chaque terminal possède un identifiant unique.

Résultat optimal : Cette primitive retourne le premier message qui parvient au terminal et dont l'adresse de destination n'est ni l'adresse de broadcast, ni l'adresse locale.

Résultat dégradé : Cette primitive retourne un message parvenu au terminal et dont l'adresse de destination n'est ni l'adresse de broadcast, ni l'adresse locale, sans garantie qu'il s'agisse du premier.

Proposition d'implémentation : Le terminal *sniffe* le réseau. Si les transmissions de messages sont effectuées sur plusieurs fréquences, le terminal choisit la fréquence à écouter de manière aléatoire. De plus, une fois reçu, la primitive retourne le message si et seulement si l'adresse de destination du message n'est ni l'adresse de broadcast ni l'adresse locale.

Résultat de l'implémentation proposée :

- Si les transmissions s'effectuent sur une unique fréquence et que le réseau est fiable, alors le premier message envoyé par un terminal voisin unilatéral de A dont l'adresse de destination n'est ni l'adresse de broadcast ni l'adresse locale est reçu par celui-ci (résultat optimal).
- Si les transmissions s'effectuent sur plusieurs fréquences ou/et que le réseau n'est pas fiable, alors seulement certains des messages qui parviennent à A dont l'adresse de destination n'est ni l'adresse de broadcast ni l'adresse locale sont reçus. On ne peut pas garantir ici que tous les messages sont reçus, on ne peut donc pas garantir que le premier est reçu (résultat dégradé).

Remarque Cette primitive est semblable à la primitive de réception d'un message quelle que soit l'adresse de destination si ce n'est qu'en plus elle applique un filtre sur les paquets reçus en fonction de la destination.

4.1.7 Détection d'un changement de localisation géographique (GPS)

Catégorie : Cette primitive est une primitive passive.

Explication : Cette primitive permet de détecter, de manière passive, un changement de localisation géographique.

Contexte d'applicabilité : Un terminal disposant d'un récepteur GPS.

Résultat optimal : Cette primitive retourne la nouvelle position du terminal lorsque celle-ci change.

Proposition d'implémentation : Cette primitive lit en permanence la position du terminal, lorsque celle-ci change, elle retourne la nouvelle position du terminal.

Résultat de l'implémentation proposée : Le résultat obtenu est le même que le résultat optimal.

4.2 Action sur le voisinage

Dans cette catégorie se trouvent toutes les primitives minimales qui nécessitent une action sur le voisinage.

4.2.1 Détection de la transformation d'un voisin unilatéral en voisin réciproque

Catégorie : Cette primitive est une primitive passive.

Explication : Cette primitive est utilisée afin de détecter de manière passive si un terminal B, qui est un voisin unilatéral de A, est à portée de celui-ci.

Contexte d'applicabilité : Deux terminaux A et B tels que B soit voisin unilatéral de A.

Résultat optimal : Cette primitive retourne vrai si et seulement si A et B sont voisins réciproques (**TRUE** \Leftrightarrow A et B voisins réciproques).

Résultats dégradés :

- **TRUE** \Rightarrow A et B voisins réciproques
- **A et B voisins réciproques** \Rightarrow **TRUE**

Proposition d'implémentation : A tente de *pinguer* le terminal B. Si le *ping* fonctionne alors A peut considérer que le terminal B est un voisin réciproque.

Résultat de l'implémentation proposée : La stratégie d'implémentation proposée permet de déterminer si le terminal était voisin réciproque à un instant t. Cette stratégie ne permet pas de détecter passivement la transformation de la relation de voisinage entre A et B. Nous ne savons pas, pour l'instant, implémenter cette primitive de manière passive. De plus, cette stratégie ne fonctionne à 100% que dans le cas où les communications sont sûres, c'est à dire sans perte de paquets. Implémentée comme cela, si cette primitive retourne vrai alors A et B sont voisins réciproques (**TRUE** \Rightarrow A et B voisins réciproques).

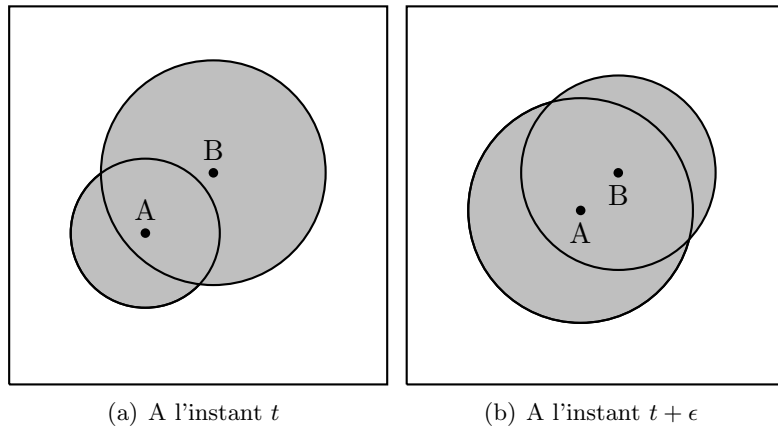


FIG. 4.3 – Détection de la transformation d'un voisin unilatéral strict en voisin réciproque

4.2.2 Détection de la transformation d'un voisin réciproque en voisin unilatéral strict

Catégorie : Cette primitive est une primitive passive.

Explication : Cette primitive est utilisée de manière passive afin de déterminer la transformation d'un voisin réciproque en voisin unilatéral strict.

Contexte d'applicabilité : Deux terminaux A et B voisins réciproques.

Résultat optimal : Cette primitive retourne vrai si et seulement si B est un voisin unilatéral strict de A (**TRUE** \Leftrightarrow **B voisin unilatéral strict de A**).

Résultats dégradés :

- **TRUE** \Rightarrow **B voisin unilatéral strict de A**
- **B voisin unilatéral strict de A** \Rightarrow **TRUE**

Proposition d'implémentation : La démarche est sensiblement la même que pour la primitive précédente : A tente de *pinguer* le terminal B. Si le *ping* échoue alors A peut considérer que le terminal B est un voisin unilatéral strict.

Résultat de l'implémentation proposée : Cette implémentation permet de déterminer, lors de la perte d'un voisin réciproque, si celui-ci est un voisin unilatéral strict ou non. Cependant, le résultat n'est garanti que si le réseau est fiable. De plus, le terminal B peut devenir un voisin unilatéral strict de A sans que celui-ci le détecte. La primitive retourne vrai si B devient voisin unilatéral strict de A (**TRUE** \Rightarrow **B voisin unilatéral strict de A**).

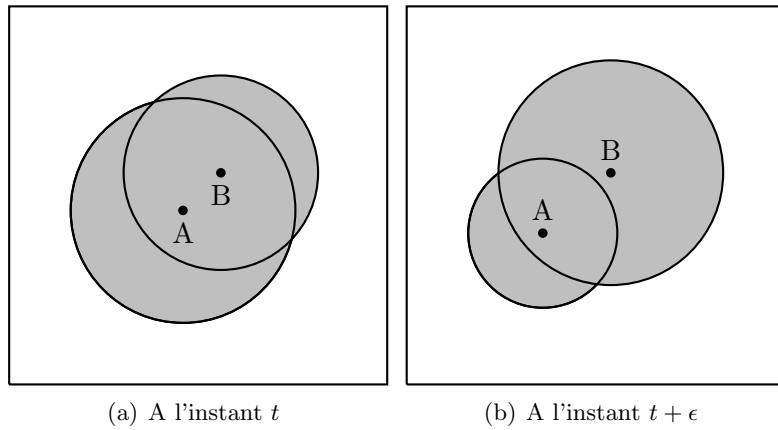


FIG. 4.4 – Détection de la transformation d'un voisin réciproque en voisin unilatéral strict

4.2.3 Détection de la transformation d'un voisin non réciproque à portée en voisin réciproque

Catégorie : Cette primitive est une primitive passive.

Explication : Cette primitive est utilisée de manière passive par un terminal A afin de détecter la transformation d'un terminal B à portée de A (tel que B ne soit pas un voisin unilatéral de A) en voisin réciproque.

Contexte d'application : Deux terminaux A et B tels que A soit voisin unilatéral strict de B.

Résultat optimal : Cette primitive retourne vrai si et seulement si A et B sont voisins réciproques (**TRUE** \Leftrightarrow A et B voisins réciproques).

Résultats dégradés :

- **TRUE** \Rightarrow A et B voisins réciproques
- A et B voisins réciproques \Rightarrow **TRUE**

Proposition d'implémentation : La démarche est sensiblement la même que pour la primitive précédente : A tente de *pinguer* le terminal B. Si le *ping* réussit alors A peut considérer que le terminal B est un voisin réciproque.

Résultat de l'implémentation proposée : Que l'on considère un réseau fiable ou non fiable, la primitive retourne vrai si B devient voisin réciproque de A (**TRUE** \Rightarrow A et B voisins réciproques).

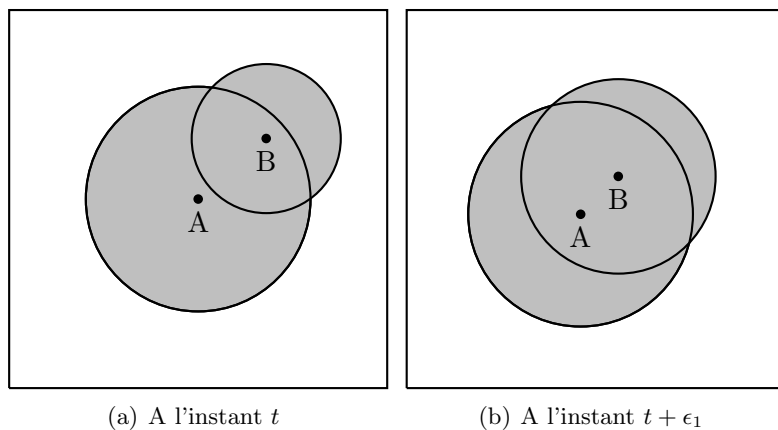


FIG. 4.5 – Détection de la transformation d'un voisin non réciproque à portée en voisin réciproque

4.2.4 Détection de la disparition d'un voisin réciproque

Catégorie : Cette primitive est une primitive passive.

Explication : Cette primitive est utilisée de manière passive afin de détecter la disparition d'un voisin réciproque.

Contexte d'application : Deux terminaux A et B tels que A et B soient voisins réciproques.

Résultat optimal : Cette primitive retourne vrai si et seulement si B n'est pas un voisin unilatéral de A (**TRUE** \Leftrightarrow B n'est pas un voisin unilatéral de A).

Résultats dégradés :

- **TRUE** \Rightarrow B n'est pas un voisin unilatéral de A
- B n'est pas un voisin unilatéral de A \Rightarrow **TRUE**

Proposition d'implémentation : La démarche est sensiblement la même que pour la primitive précédente : A tente de *pinguer* le terminal B. Si le *ping* échoue alors A peut considérer que le terminal B n'est plus un voisin réciproque. De plus, comme les messages de signalement de B ne parviennent plus à A, on est bien dans les cas des figures 4.6 et 4.7.

Résultat de l'implémentation proposée : Cette implémentation permet de déterminer, lors de la perte d'un voisin réciproque, si celui-ci est hors de portée. Cependant le résultat n'est garanti que si le réseau est fiable. De plus, le terminal A peut devenir un voisin unilatéral strict de B sans être capable de le détecter. La primitive retourne vrai si B n'est plus voisin réciproque de A et B n'est pas un voisin unilatéral strict de A (**TRUE** \Rightarrow B n'est pas un voisin unilatéral de A).

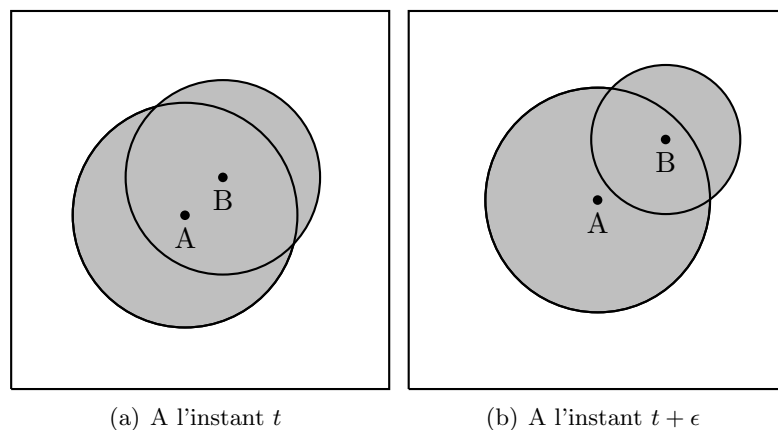


FIG. 4.6 – Détection de la disparition d'un voisin réciproque

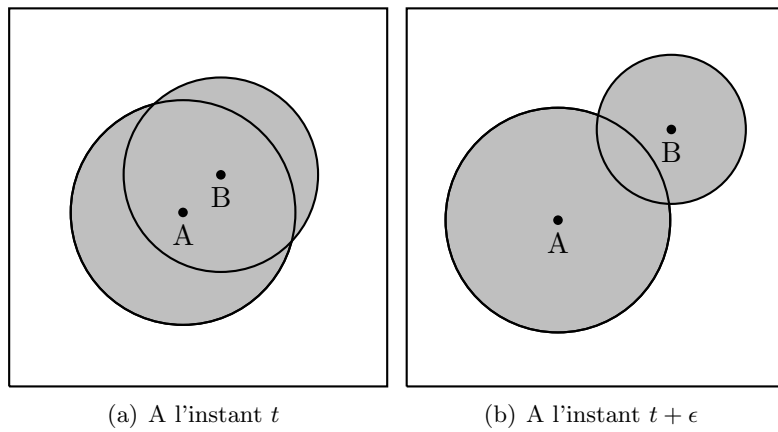


FIG. 4.7 – Détection de la disparition d'un voisin réciproque

4.2.5 Signalement de l'apparition d'un nœud à ses voisins

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive consiste, pour un terminal, à annoncer sa présence à ses voisins.

Contexte d'applicabilité : Aucun contexte particulier.

Résultat optimal : Tous les voisins du terminal qui exécute cette primitive sont avertis de la présence de celui-ci.

Proposition d'implémentation : Une implémentation possible consiste en l'envoi d'un message sur l'adresse de broadcast permettant d'annoncer la présence du terminal. Le délai optimal entre deux signalements successifs a été étudié en [18].

Résultat de l'implémentation proposée :

- Si le réseau est fiable et est composé de terminaux possédant chacun une identité unique, alors cette primitive permet au terminal de signaler sa présence à tous ses voisins.
- Si le réseau est fiable et est composé de terminaux anonymes, alors cette primitive permet au terminal de signaler à tous ses voisins qu'ils ont au moins un voisin unilatéral.
- Si le réseau est non fiable et est composé de terminaux possédant chacun une identité unique, alors cette primitive permet au terminal de signaler sa présence à seulement certains de ses voisins (d'aucun à tous).
- Si le réseau est non fiable et est composé de terminaux anonymes, alors cette primitive permet au terminal de signaler à seulement certains de ses voisins (d'aucun à tous) qu'ils ont au moins un voisin unilatéral.

4.2.6 Signalement de l'appartenance d'un nœud à un groupe à ses voisins

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive consiste pour un terminal à annoncer à ses voisins son appartenance à un groupe.

Contexte d'applicabilité : Chaque terminal possède une identité unique.

Résultat optimal : Tous les voisins du terminal qui exécute cette primitive sont avertis de son appartenance au dit groupe.

Proposition d'implémentation : Une implémentation possible consiste en l'envoi d'un message spécifique sur l'adresse de broadcast permettant d'annoncer l'appartenance du terminal au groupe à ses voisins.

Résultat de l'implémentation proposée :

- Si le réseau est fiable, alors cette primitive permet au terminal de signaler son appartenance au groupe à tous ses voisins.
- Si le réseau est non fiable, alors cette primitive permet au terminal de signaler son appartenance au groupe à seulement certains de ses voisins (d'aucun à tous).

4.2.7 Vérification de la présence d'un terminal connu

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive permet de savoir si un terminal dont l'adresse est connue est à portée.

Contexte d'applicabilité : Chaque terminal possède une identité unique.

Résultat optimal : Cette primitive retourne vrai si et seulement si le terminal recherché est à portée (**TRUE** \Leftrightarrow **Le terminal recherché est à portée**).

Résultats dégradés :

- **TRUE** \Rightarrow **Le terminal recherché est à portée**
- **Le terminal recherché est à portée** \Rightarrow **TRUE**

Proposition d'implémentation : Une implémentation possible de cette primitive peut consister en un simple *ping* vers le terminal dont la présence est testée.

Résultat de l'implémentation proposée : Si le *ping* réussit alors le terminal recherché est un voisin réciproque. (**TRUE** \Rightarrow **Le terminal recherché est voisin réciproque**).

Remarque : Cette primitive qui est relativement simple possède un grand intérêt dans les réseaux dont la technologie radio ne permet pas une découverte rapide des voisins. Par exemple, si la technologie radio utilisée est la technologie Bluetooth, la découverte de dispositifs voisins prend au minimum 10.24 s d'après la spécification. Cette méthode permet donc, lorsque l'on souhaite communiquer avec un terminal particulier connu, de s'abstenir de la recherche de voisins, coûteuse en temps.

4.2.8 Envoi d'un message asynchrone adressé ou avec comme destination l'adresse de broadcast

Catégorie : Cette primitive est une primitive active.

Explications : Cette primitive consiste à envoyer un message de manière asynchrone soit à un terminal, soit à l'ensemble des terminaux à portée.

Contexte d'applicabilité : Aucun contexte particulier.

Résultat optimal : Le message envoyé parvient au(x) destinataire(s) du message si celui-ci (resp. ceux-ci) est (resp. sont) à portée du terminal (**TRUE** \Leftrightarrow **Le message parvient à l'ensemble des destinataires à portée**).

Résultats dégradés :

- **TRUE** \Rightarrow Le message parvient à l'ensemble des destinataires à portée
- **TRUE** \Rightarrow Le message parvient à certains des destinataires à portée
- Le message parvient à l'ensemble des destinataires à portée \Rightarrow **TRUE**
- Le message parvient à certains des destinataires à portée \Rightarrow **TRUE**

Remarque : Cette primitive de bas niveau est implémentée dans TinyOS [10]. L'envoi est asynchrone, ce qui signifie qu'aucun acquittement de message n'est attendu de la part du ou des destinataire(s).

4.2.9 Envoi d'un message synchrone adressé ou avec comme destination l'adresse de broadcast

Catégorie : Cette primitive est une primitive active.

Explications : Cette primitive consiste à envoyer un message adressé de manière synchrone à un terminal. L'envoi est synchrone, ce qui signifie que le destinataire doit, à la réception de ce message, faire en sorte que l'émetteur soit informé de la bonne réception.

Contexte d'applicabilité : Aucun contexte particulier.

Résultat optimal : Cette primitive retourne vrai si et seulement si le destinataire reçoit le message (**TRUE** \Leftrightarrow Le message est correctement reçu par l'ensemble des destinataires).

Résultats dégradés :

- **TRUE** \Rightarrow Le message est correctement reçu par l'ensemble des destinataires
- **TRUE** \Rightarrow Le message est correctement reçu par certains des destinataires
- Le message est correctement reçu par l'ensemble des destinataires \Rightarrow **TRUE**
- Le message est correctement reçu par certains des destinataires \Rightarrow **TRUE**

Remarque : Cette primitive, pourtant de bas niveau, n'est pas implémentée dans TinyOS [10]. C'est au programmeur d'effectuer la gestion de l'envoi et de la réception des acquittements.

4.3 Opérations locales destinées à faciliter les interactions dans les MANets

Les primitives définies ici effectuent des opérations locales, c'est à dire qu'elles ne nécessitent pas l'utilisation d'une interface de communication.

4.3.1 Génération de paires de clefs asymétriques

Catégorie : Cette primitive est une primitive active.

Explication : Cette clef peut permettre l'échange de données chiffrées entre deux terminaux.

Contexte d'applicabilité : Aucun contexte particulier.

Résultat optimal : Cette primitive retourne une clef asymétrique.

Remarque : La sécurisation des échanges peut passer par le chiffrement des données envoyées sur le réseau. Cependant, le chiffrement et le déchiffrement grâce à ce type de clef sont coûteux en temps et en énergie.

4.3.2 Génération d'une clef symétrique

Catégorie : Cette primitive est une primitive active.

Explication : Une clef symétrique permet, par exemple, lorsqu'elle est partagée par deux terminaux, d'échanger des données chiffrées.

Contexte d'applicabilité : Aucun contexte particulier.

Résultat optimal : Cette primitive retourne une clef symétrique.

Remarque : Le cryptage et décryptage d'informations utilisant une clef symétrique sont plus rapides qu'avec une clef asymétrique. De plus, le cryptage et le décryptage de données sont gourmands en ressources CPU et en énergie. L'utilisation de ce type de clef est donc très intéressant si elle est partagée par deux terminaux.

4.3.3 Création d'un *Tuple Space*

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive permet de créer un *Tuple Space local*, prêt à accueillir des *Tuples*.

Contexte d'applicabilité : Aucun contexte particulier.

Résultat optimal : Cette primitive retourne vrai si et seulement si la création du *Tuple Space local* a réussi.

4.4 Opérations complexes qui ne nécessitent pas une action sur le voisinage

Les opérations complexes sont des opérations qui utilisent une ou plusieurs des primitives minimales définies précédemment. Nous répertorions ici celles qui ne nécessitent pas une action sur le voisinage, c'est à dire que toutes les primitives minimales qui la composent sont des primitives qui appartiennent à la catégorie "Réponse passive à l'évolution du voisinage".

4.4.1 Estimation de la distance d'un terminal à partir de la puissance du signal

Catégorie : Cette primitive est une primitive passive.

Explications : L'affaiblissement d'un signal dans un milieu est régi par des lois physiques. L'utilisation de celles-ci permet une estimation de la distance séparant deux terminaux.

Contexte d'applicabilité : Deux terminaux A et B tels que B soit voisin unilatéral de A.

Résultat optimal : Cette primitive retourne un entier correspondant à la distance séparant A et B (unité à préciser).

Proposition d'implémentation : Cette primitive s'appuie sur la primitive du calcul de la puissance de signal reçu. En fonction du résultat de celle-ci et des lois physiques de propagation des ondes de l'environnement (forêt, eau liquide, glace, air, vide, etc..) dans lequel se trouvent les 2 terminaux, cette primitive retourne la distance séparant A et B.

Résultat de l'implémentation proposée : Le résultat de la primitive dépend de l'uniformité du milieu dans lequel se trouvent les deux terminaux, de la connaissance précise de la puissance utilisée lors de l'envoi et de la précision de mesure de la puissance de signal reçu.

4.4.2 Détection d'une perte imminente éventuelle de voisin unilatéral

Catégorie : Cette primitive est une primitive passive.

Explication : Cette primitive permet d'anticiper, de manière passive, une perte éventuelle de voisin unilatéral. Elle peut utiliser la primitive précédente. L'objectif de cette primitive est de permettre une réaction avant la perte effective d'un voisin unilatéral.

Contexte d'applicabilité : Deux terminaux A et B tels que B soit voisin unilatéral de A.

Résultat optimal : Cette primitive retourne vrai lorsqu'un terminal unilatéral est sur le point d'être perdu (**TRUE** \Leftrightarrow **Un voisin unilatéral est sur le point de disparaître**).

Résultats dégradés :

- **TRUE** \Rightarrow **Un voisin unilatéral est sur le point de disparaître**
- **TRUE** \Rightarrow **Un voisin unilatéral est peut-être sur le point de disparaître**
- **Un voisin unilatéral est sur le point de disparaître** \Rightarrow **TRUE**
- **Un voisin unilatéral est peut-être sur le point de disparaître** \Rightarrow **TRUE**

Proposition d'implémentation : Cette primitive étudie la distance séparant les terminaux A et B. En fonction de celle-ci ou de la variation de celle-ci, et de modèles de mobilité spécifiques au contexte dans lequel se trouvent les terminaux [6], la primitive détecte une éventuelle perte imminente de voisin unilatéral.

Résultat de l'implémentation proposée : Il est impossible dans un réseau mobile d'anticiper à coup sûr la perte d'un voisin unilatéral. Le résultat optimal n'est donc pas réalisable dans un réseau MANet défini sans contraintes spécifiques. (**TRUE** \Rightarrow **Un voisin unilatéral est peut-être sur le point de disparaître**).

4.4.3 Lecture de l'ensemble des *tuples* présents dans le *Tuple Space local*

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive correspond à la primitive *public Vector read(Tuple template)* de TOTA [12].

Contexte d'applicabilité : Un terminal A possédant un *Tuple Space local*.

Résultat optimal : Cette primitive retourne l'ensemble des *tuples* localement présents dans le *Tuple Space local* qui sont identiques au *tuple* passé en paramètre.

4.4.4 Lecture de l'unique *tuple* correspondant au *tuple* passé en paramètre dans le *Tuple Space local*

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive correspond à la primitive *public Tuple keyrd(Tuple template)* de TOTA [12].

Contexte d'applicabilité : Un terminal A possédant un *Tuple Space local*.

Résultat optimal : Cette primitive retourne l'unique *tuple* du *Tuple Space local* qui possède le même identifiant que celui du *tuple* passé en paramètre.

Remarque : Chaque *tuple* possède un identifiant unique.

4.4.5 Suppression d'un *tuple* du *Tuple Space local*

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive correspond à la primitive *public Vector delete (Tuple template)* de TOTA [12].

Contexte d'applicabilité : Un terminal A possédant un *Tuple Space local*.

Résultat optimal : Cette primitive retourne, après les avoir supprimés, l'ensemble des *tuples* qui se trouvent dans le *Tuple Space local* et qui correspondent au *tuple* passé en paramètre.

4.4.6 Association de l'arrivée d'un *tuple* avec le déclenchement d'un événement

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive correspond à la primitive *public void subscribe (Tuple template, ReactiveComponent comp, String rct)* de TOTA [12].

Contexte d'applicabilité : Un terminal A possédant un *Tuple Space local*.

Résultat optimal : Cette primitive retourne vrai lorsqu'elle réussit à associer le déclenchement d'un événement à l'arrivée d'un *tuple* correspondant au *tuple* passé en paramètre.

4.4.7 Suppression de l'association de l'arrivée d'un *tuple* avec le déclenchement d'un événement

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive correspond à la primitive *public void unsubscribe (Tuple template, ReactiveComponent comp)* de TOTA [12].

Contexte d'applicabilité : Un terminal A possédant un *Tuple Space local*.

Résultat optimal : Cette primitive retourne vrai lorsqu'elle réussit à supprimer une association faite grâce à la primitive précédente.

4.5 Opérations complexes qui nécessitent une action sur le voisinage

De même, les opérations complexes définies ici sont des opérations qui utilisent une ou plusieurs primitives minimales définies précédemment. Nous répertorions ici celles qui nécessitent une action sur le voisinage, c'est à dire que les primitives minimales qui la composent peuvent être extraites de la catégorie "Action sur le voisinage".

4.5.1 Détection de l'ensemble des terminaux à portée

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive permet de déterminer l'ensemble des terminaux à portée, que ceux-ci soit réciproques ou non.

Contexte d'application : Un réseau où chaque terminal peut avoir une portée différente.

Résultat optimal : Cette primitive retourne l'ensemble des terminaux à portée (**Liste de terminaux de taille $n \Leftrightarrow n$ terminaux à portée**).

Résultats dégradés :

- **Retourne une liste de n terminaux \Rightarrow au moins n terminaux à portée**
- **Retourne une liste de n terminaux \Rightarrow au plus n terminaux à portée**
- **Retourne une liste de n terminaux \Rightarrow exactement n terminaux à portée**
- **Au moins n terminaux à portée \Rightarrow Retourne une liste de n terminaux**
- **Au plus n terminaux à portée \Rightarrow Retourne une liste de n terminaux**
- **Exactement n terminaux à portée \Rightarrow Retourne une liste de n terminaux**

Proposition d'implémentation : Dans l'exemple des figures 4.8 et 4.9, B ne peut à aucun moment communiquer avec A.

1. Le terminal A envoie un message M_1 en broadcast (donc à l'attention de tous les terminaux à sa portée).
2. Les terminaux à sa portée qui ont reçu le message M_1 et qui ne peuvent lui répondre directement, répondent par un envoi en broadcast d'un message M_2 .
3. Les terminaux à sa portée qui ont reçu le message M_1 et qui peuvent lui répondre lui répondent grâce à un message M_3 et font suivre les messages de type M_2 .
4. Le terminal A reçoit l'ensemble des réponses directes M_3 et des réponses indirectes M_2 .

Résultat de l'implémentation proposée : Cette implémentation retourne l'ensemble ou seulement certains des terminaux à portée (**Liste de terminaux de taille $n \Rightarrow$ au moins n terminaux à portée**).

Remarque : Cette primitive permet de déterminer quels étaient les terminaux à portée à un instant t . Elle ne garantit absolument pas la pérennité de l'information.

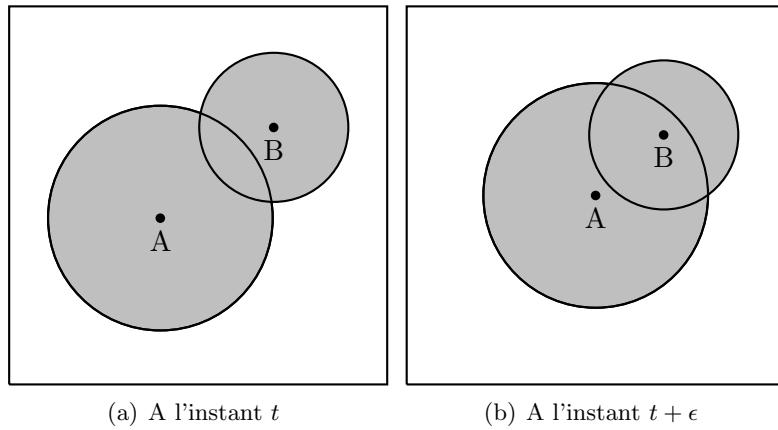


FIG. 4.8 – Détection de l'ensemble des terminaux à portée

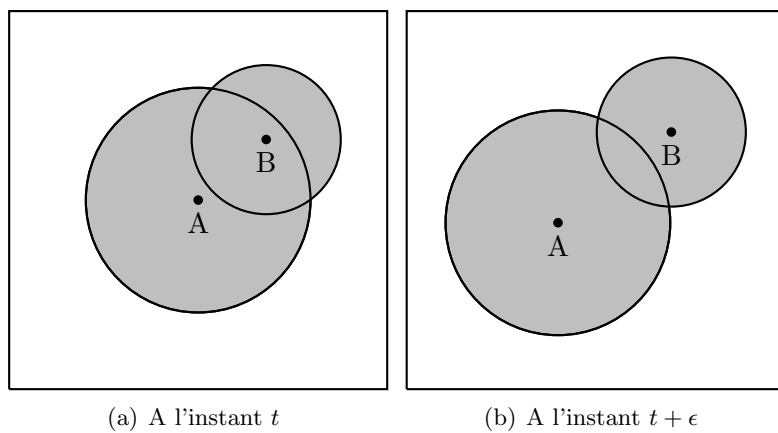


FIG. 4.9 – Détection de l'ensemble des terminaux à portée

4.5.2 Détection de l'apparition d'un voisin réciproque

Catégorie : Cette primitive est une primitive passive.

Explication : Cette primitive permet de détecter, de manière passive, l'apparition d'un voisin réciproque. Elle est canonique car elle utilise successivement deux primitives minimales.

Contexte d'applicabilité : Aucun contexte particulier.

Résultat optimal : La primitive retourne vrai si et seulement si l'apparition d'un nouveau voisin réciproque est détectée (**TRUE** \Leftrightarrow **Apparition d'un voisin réciproque**).

Résultats dégradés :

- **TRUE** \Rightarrow **Apparition d'un voisin réciproque**
- **Apparition d'un voisin réciproque** \Rightarrow **TRUE**

Proposition d'implémentation : Cette détection peut avoir lieu de deux manières différentes dépendantes du réseau :

- A détecte un voisin unilatéral B puis devient lui même voisin unilatéral de B (cf. fig. 4.10). Il s'agit de l'association successive des primitives "Détection de l'apparition d'un voisin unilatéral" (cf. fig. 4.1 page 15) et "Détection de la transformation d'un voisin unilatéral en voisin réciproque" (cf. fig. 4.3 page 19).

- A détecte qu'il est un voisin unilatéral de B puis B devient son voisin unilatéral (cf. fig. 4.11). Il s'agit ici de l'association des primitives "Test permettant de déterminer si un terminal est à ma portée" (cf. fig. 4.8 page 29) et "Détection de la transformation d'un voisin non réciproque à portée en voisin réciproque" (cf. fig. 4.5 page 20). Cependant, dans cette version, lors de l'utilisation de la primitive "Détection de la transformation d'un voisin non réciproque à portée en voisin réciproque", nous devons nous assurer que le résultat de la primitive "Test permettant de déterminer si un terminal est à ma portée" est toujours vrai.

Résultat de l'implémentation proposée : Cette primitive retourne vrai si l'apparition d'un nouveau voisin est détectée (**TRUE** \Rightarrow **Apparition d'un voisin réciproque**).

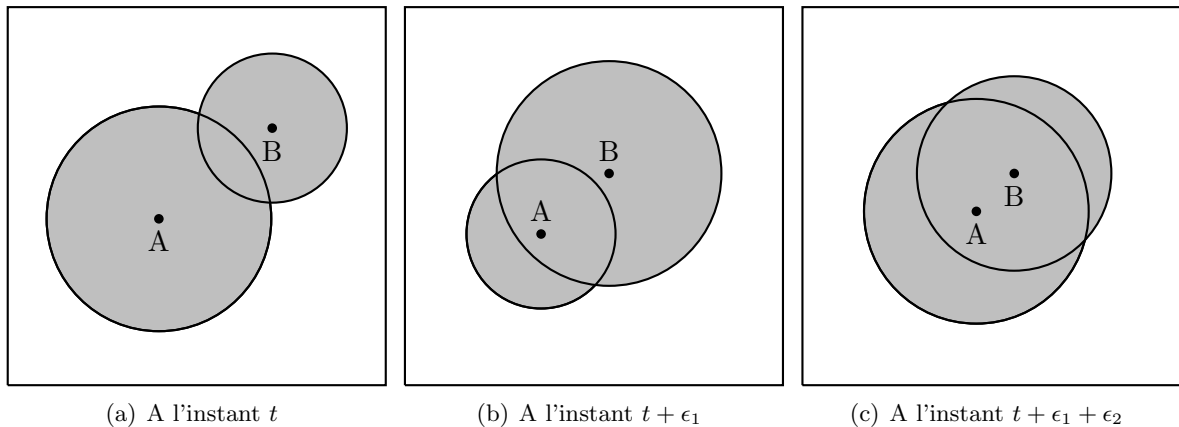


FIG. 4.10 – Détection de l'apparition d'un voisin réciproque

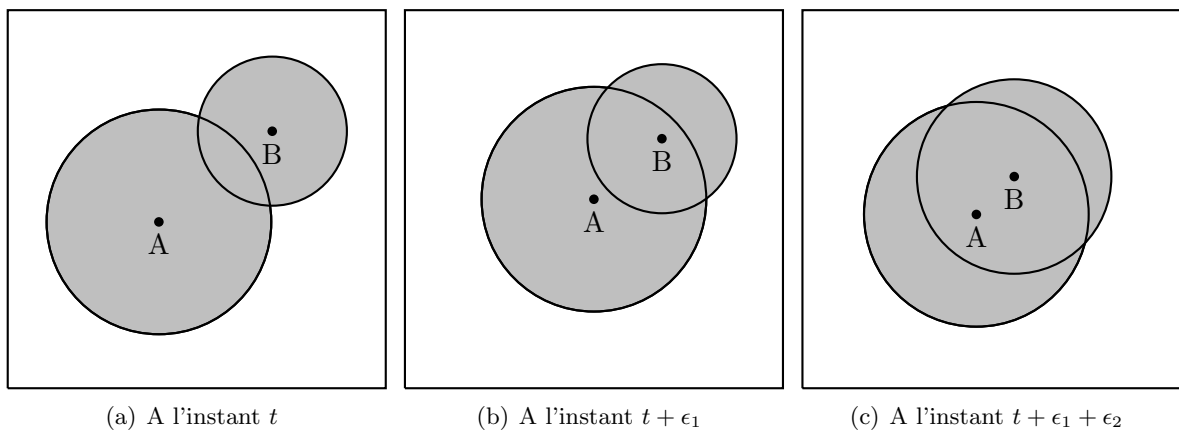


FIG. 4.11 – Détection de l'apparition d'un voisin réciproque

4.5.3 Etablissement d'une connexion avec un voisin

Catégorie : Cette primitive est une primitive active.

Explication : L'objectif de cette primitive est d'établir une connexion telle que définie dans le chapitre 2. La définition de connexion à considérer est directement dépendante de la vérification ou non de la propriété P_6 définie précédemment : "Les terminaux possèdent plusieurs interfaces de communication.". Nous devons donc considérer les deux cas de figure suivants :

1. Cas où les terminaux ne possèdent qu'une seule interface de communication.
2. Cas où les terminaux possèdent plusieurs interfaces de communication.

Contexte d'application : Deux terminaux A et B voisins réciproques.

Résultat optimal : Cette primitive retourne vrai si et seulement si les terminaux A et B sont connectés (**TRUE** \Leftrightarrow **A et B connectés**).

Résultats dégradés :

- **TRUE** \Rightarrow **A et B connectés**
- **A et B connectés** \Rightarrow **TRUE**

1. Si les terminaux ne possèdent qu'une seule interface de communication, nous testons régulièrement la connexion en vérifiant que les terminaux A et B sont toujours voisins réciproques.
2. Si les terminaux possèdent plusieurs interfaces de communication, nous considérons que sur chaque terminal une radio est utilisée pour émettre, et une autre pour recevoir. Il s'agit ici de reproduire une connexion classique d'un réseau filaire avec la création de socket en mode connecté. Chaque terminal émet "en continu" sur une fréquence propre. Dès que l'un des deux terminaux ne reçoit plus le flux de données, on considère que la connexion n'existe plus. On est ainsi informé en temps réel de la perte de la connexion. Cependant, les avantages de la détection en temps réel de la rupture d'une connexion entraînent deux inconvénients : une utilisation excessive d'énergie et une augmentation des possibles collisions des paquets si les terminaux sont nombreux.

Résultat de l'implémentation proposée : Cette primitive retourne vrai si les terminaux A et B sont connectés (**TRUE** \Rightarrow **A et B connectés**).

4.5.4 Fermeture d'une connexion avec un voisin

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive sert à fermer une connexion entre deux terminaux. La fermeture de connexion par un des deux terminaux connectés doit entraîner la fermeture immédiate de la connexion sur l'autre terminal.

Contexte d'applicabilité : Deux terminaux A et B connectés.

Résultat optimal : Cette primitive retourne vrai si et seulement si A et B ne sont plus connectés (**TRUE** \Leftrightarrow **A et B ne sont plus connectés**).

Résultats dégradés :

- **TRUE** \Rightarrow **A et B ne sont plus connectés**
- **A et B ne sont plus connectés** \Rightarrow **TRUE**

Proposition d'implémentation : Les terminaux sont connectés donc la fermeture de connexion peut s'effectuer grâce à un simple envoi de message prédéfini.

Résultat de l'implémentation proposée : Cette primitive retourne vrai si et seulement si A et B ne sont plus connectés (**TRUE** \Leftrightarrow **A et B ne sont plus connectés**).

4.5.5 Détection d'une perte de connexion

Catégorie : Cette primitive est une primitive passive.

Explication : Cette primitive sert à détecter de manière passive une perte de connexion entre deux terminaux.

Contexte d'applicabilité : Deux terminaux A et B connectés.

Résultat optimal : Cette primitive retourne vrai si et seulement si A et B ne sont plus connectés (**TRUE** \Leftrightarrow **A et B ne sont plus connectés**).

Résultats dégradés :

- **TRUE** \Rightarrow **A et B ne sont plus connectés**
- **A et B ne sont plus connectés** \Rightarrow **TRUE**

Proposition d'implémentation :

1. Si les terminaux ne possèdent qu'une seule interface de communication, alors il s'agit de vérifier la validité de la connexion tous les temps t.
2. Si les terminaux possèdent plusieurs interfaces de communication, alors il s'agit de vérifier la validité du flux entrant.

Résultat de l'implémentation proposée : Cette primitive retourne vrai si A et B ne sont plus connectés (**TRUE** \Rightarrow **A et B ne sont plus connectés**).

4.5.6 Etablissement d'un rendez-vous avec un voisin

Catégorie : Cette primitive est une primitive active.

Explication : L'établissement d'un rendez-vous consiste en l'application par plusieurs terminaux d'un même algorithme dont le but est de faire en sorte qu'un certain nombre de ceux-ci se consacrent exclusivement les uns aux autres.

Contexte d'applicabilité : Au moins deux terminaux voisins réciproques.

Résultat optimal : Cette primitive retourne vrai si et seulement si un rendez-vous est pris avec un ou plusieurs terminaux.

Proposition d'implémentation : Un exemple d'algorithme permettant une prise de rendez-vous entre deux terminaux nous est donné par Yves Métivier [14] pour un cadre statique :

Each vertex v repeats forever the following actions :
the vertex v selects one of its neighbours $c(v)$ chosen at random ;
the vertex v sends 1 to $c(v)$;
the vertex v sends 0 to its neighbours different from $c(v)$;
the vertex v receives messages from all its neighbours.
(* There is a rendezvous between v and $c(v)$ if v receives 1 from $c(v)$ *)

FIG. 4.12 – A Randomized Algorithm to Obtain Rendezvous

Résultat de l'implémentation proposée : Cet algorithme ne fonctionne que dans un cadre statique et si la supposition minimale réaliste "chaque terminal possède un identifiant unique" est vérifiée. Cette primitive repose aussi sur les primitives minimales d'envoi de messages adressés et de réception de messages.

4.5.7 Injection d'un tuple dans le Tuple Space distribué

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive correspond à la primitive *public void inject (TotaTuple tuple)* de TOTA [12]. Elle est utilisée pour injecter le *tuple* passé en paramètre dans le réseau TOTA [12]. Une fois injecté, le *tuple* commence à se propager selon les règles de propagation se trouvant dans sa définition.

Contexte d'applicabilité : Un terminal A possédant un *Tuple Space local*.

Résultat optimal : Cette primitive retourne vrai si et seulement si le *tuple* a été correctement injecté dans le *Tuple Space distribué*.

4.5.8 Lecture des *tuples* présents dans le *Tuple Space distribué*

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive correspond à la primitive *public Vector readOneHop (Tuple template)* de TOTA [12].

Contexte d'applicabilité : Un terminal A possédant un *Tuple Space local*.

Résultat optimal : Cette primitive retourne l'ensemble des *tuples* présents dans le *Tuple Space distribué* et qui correspondent au *tuple* passé en paramètre.

4.5.9 Lecture de l'unique *tuple* correspondant à l'identifiant du *tuple* passé en paramètre dans le *Tuple Space distribué*

Catégorie : Cette primitive est une primitive active.

Explication : Cette primitive correspond à la primitive *public Vector keyrdOneHop (Tuple template)* de TOTA [12].

Contexte d'applicabilité : Un terminal A possédant un *Tuple Space local*.

Résultat optimal : Cette primitive retourne l'unique *tuple* parmi les *tuples* du *Tuple Space distribué* qui possède le même identifiant que celui du *tuple* passé en paramètre.

Remarque : Chaque *tuple* possède un identifiant unique. La gestion de cet identifiant est invisible au niveau de l'application.

Chapitre 5

Algorithmes reposant sur les NCIPs

Chacun des algorithmes ci-dessous repose sur une ou plusieurs primitives définies dans le chapitre précédent. Nous montrons ainsi qu'un algorithme permettant une action complexe peut-être défini grâce à quelques primitives minimales.

5.1 Présentation des algorithmes

5.1.1 Algorithme de synchronisation

Cet algorithme est un algorithme de gestion du partage d'une ressource critique entre 2 ou 3 terminaux. Tous les terminaux communiquent entre eux afin de déterminer lequel peut utiliser la ressource pendant une durée λ . Le pseudo code de cet algorithme est le suivant :

```
main() {
    boolean alone := true;

    while (alone = true) {
        sleep(1);
        send(BROADCAST_ADDR, 0);
        if (recevoir() != NULL) {
            alone != false;
        }
    }

    while (true) {
        int number := random();
        neighbor random_neighbor := choose_random_neighbor();
        send(BROADCAST_ADDR, 0);
        send(voisin_aléatoire, number);
        int[] result := receive();
        if (result[random_neighbor] != 0 && number > result[random_neighbor]) {
            // nous disposons ici de la ressource pour exactement 10 secondes
        }
        else {
            sleep(10);
        }
    }
}
```

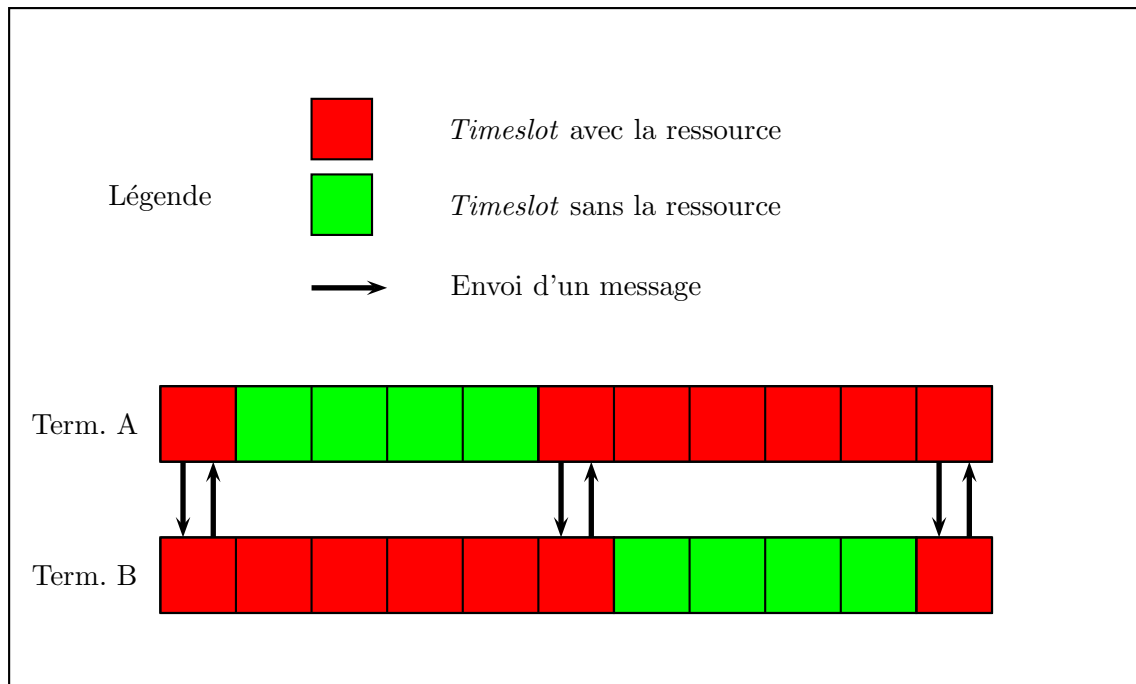


FIG. 5.1 – Schéma temporel de l'algorithme de synchronisation

Cet algorithme ne peut exister que si la propriété P_2 : “Chaque terminal possède un identifiant unique.” est vérifiée. Il utilise uniquement les primitives “Envoi d'un message asynchrone adressé ou avec comme destination l'adresse de broadcast” et “Réception d'un message dont l'adresse de destination est l'adresse de broadcast ou l'adresse locale”.

5.1.2 Algorithme 'je suis entendu'

Cet algorithme peut-être utilisé dans le cas suivant : Un nœud n (muni d'une identité id_n) dispose d'une information i . Il souhaite ne plus garder cette information, mais qu'elle soit maintenant conservée par un autre nœud du réseau. Il n'a pas besoin de savoir qui est le nouveau nœud qui conserve l'information. Quand il est informé qu'un nœud l'a entendu et conserve donc maintenant la valeur, il peut supprimer sa copie.

Le nœud n applique l'algorithme suivant (proposé par S. Chaumette) :

```
main() {
    broadcast(type=store, (id=id_n, value=i));
    while (true); /* le process sera dérouté sur whenReceivedMessage
                  * à l'arrivée d'un message
                  */
}

whenReveiveMessage(type, contenu) {
    if (type=heard and contenu.id==id_n) { /* quelqu'un m'a entendu */
        discard(i); /* je supprime ma copie locale de i */
    }
}
}
```

Les autres nœuds font la chose suivante :

```
main() {
    while (true); /* le process sera dérouté sur whenReceivedMessage
```

```

        * à l'arrivée d'un message
        */
    }

whenReceiveMessage(type, contenu) {
    if (type=store) {
        i=contenu.value
        broadcastMessage(type=heard, (ud=id_n))
    }
    if (type=heard) {
        broadcastMessage(type=heard, (id=contenu.id_n))
    }
}

```

Après un certain temps, le ou les nœuds qui ont la valeur appliquent à leur tour le premier code.

Comme précédemment, cet algorithme ne peut exister que si la propriété P_2 : “Chaque terminal possède un identifiant unique.” est vérifiée. Il utilise uniquement une variante de la primitive “Détection de l’ensemble des terminaux à portée”. En effet, il ne s’agit pas ici de connaître l’identité des terminaux à portée mais simplement de déterminer si à un instant donné il y avait au moins un nœud qui était à portée du nœud n .

Remarquons, même si ceci est hors du propos de ce document que l’algorithme présente les intérêts suivants :

Sécurité/Confidentialité : Aucun nœud (excepté celui qui possède la valeur) ne connaît la nouvelle localisation de l’information i .

Redondance : La données se trouve parfois dupliquée sur le réseau.

Gestion d’énergie : Equilibrage de la consommation (électrique, mémoire, ...) pour stocker l’information i .

5.2 Implémentation de l’algorithme de synchronisation

Nous avons souhaité implémenter cet algorithme sur des téléphones mobiles P910i et des capteurs Crossbow Mica2.

5.2.1 Implémentation sur des téléphones mobiles SonyEricsson P910i

Les téléphones P910i disposent de la technologie radio Bluetooth et d’une machine virtuelle Java. Nous avons donc décidé de programmer cet algorithme grâce à l’API Java pour Bluetooth : l’API JSR-82 [13]. L’algorithme qui précède considère que les terminaux communiquent deux-à-deux. Toute communication Bluetooth repose sur une relation maître/esclave. Rien n’empêche un esclave d’avoir plusieurs maîtres, un maître d’avoir plusieurs esclaves et en théorie un terminal d’être à fois maître et esclave. De manière à simuler une communication pair-à-pair, chaque terminal est à la fois maître et esclave de tous les autres. Cependant, malgré la spécification, certains équipements radio ne sont pas capables d’être dans le même temps maître et esclave. Malheureusement, les P910i font partie de ceux-ci.

Note technique : Pour vérifier si un terminal peut à la fois être maître et esclave, il faut contrôler les propriétés du périphérique Bluetooth local en appelant la méthode statique `getProperty` de la classe `LocalDevice` avec comme paramètre “`bluetooth.master.swith`”. Le terminal Bluetooth peut être dans le même temps maître et esclave si et seulement si cette méthode retourne vrai.

Etant donné que la limitation vient du téléphone, nous avons décidé d'utiliser le modèle de gamme supérieure, à savoir les téléphones mobiles SonyEricsson P990i. Ceux-ci ne sont malheureusement pas non plus capables de passer du rôle de maître au rôle d'esclave. Nous n'avons donc pas pu implémenter cette primitive sur les téléphones dont nous disposions.

5.2.2 Implémentation sur des capteurs Crossbow Mica2

Les capteurs Crossbow Mica2 (cf. fig. 5.2) ont pour système d'exploitation TinyOS [10] que nous avons présenté en introduction. La programmation sur ce système d'exploitation est une programmation événementielle qui utilise le langage nesC [5]. L'avantage de la programmation événementielle est qu'elle permet notamment une gestion simplifiée des communications.



FIG. 5.2 – Un capteur Crossbow Mica2

L'implémentation que nous avons mise en place utilise les diodes des capteurs afin de nous indiquer où ils en sont de leur synchronisation. Les diodes vertes et jaunes correspondent respectivement aux signalement d'un envoi et d'une réception. Lorsque la diode rouge est allumée, cela signifie que le capteur dispose de la ressource distribuée de manière exclusive. L'algorithme fonctionne avec deux ou trois capteurs.

5.2.3 Résultats

En pratique, lorsque que cet algorithme est exécuté par deux terminaux, chacun obtient la ressource une fois sur deux. La ressource distribuée est donc utilisée à chaque nouvelle itération, c'est à dire 100% du temps.

De même, lorsque que cet algorithme est exécuté par trois terminaux, chacun obtient la ressource en moyenne une fois sur quatre. La ressource distribuée est donc utilisée en moyenne trois fois toutes les quatre itérations, c'est à dire 75% du temps.

Chapitre 6

Conclusion

La diversité des réseaux MANets, de par les terminaux qui les composent et leurs technologies de communication, engendre de nombreuses contraintes lors du développement d'une application. La diversité des solutions apportées rivalise avec la diversité des problèmes. Nous pensons que fournir un environnement minimal générique permettant l'accès au maximum de NCIPs réalisables en fonction du contexte est une voie prometteuse pour la programmation sur les MANets.

Dans ce travail de recherche, nous nous sommes tout d'abord intéressés aux systèmes existants conçus pour les réseaux MANets afin d'identifier les primitives qu'ils utilisent ou fournissent aux programmeurs. Nous avons ensuite défini une méthodologie de classification des primitives sur laquelle nous nous sommes appuyés afin de classer l'ensemble des primitives, que nous les ayons identifiées dans les systèmes précédemment étudiés ou que nous les ayons définies. Enfin, nous avons illustré l'utilisation de certaines d'entre elles grâce à la définition de deux algorithmes qui les utilisent.

Afin de compléter cette classification, une étude sur la stabilité des résultats retournés par les primitives en fonction du contexte dans lequel celles-ci ont été appelées doit être effectuée. Notre futur travail doit aussi s'orienter vers une extension du modèle de réétiquetage de graphe dynamique DA-GRS (Dynamicity Aware Graph Relabeling Systems) [1] défini dans l'équipe SOD afin de prendre en compte l'asymétrie des relations de voisinage.

L'objectif final de ce travail de recherche est le développement d'un intergiciel qui permettra non pas de masquer la mobilité des terminaux mais de fournir des primitives simples et canoniques aux développeurs et une connaissance du contexte et du voisinage aux utilisateurs.

Bibliographie

- [1] Arnaud Casteigts and Serge Chaumette. Dynamicity-Aware Graph Relabeling Systems (DAGRS), a local computation-based model to describe MANet algorithms. In *International Conference on Parallel and Distributed Computing and Systems (PDCS'05)*, Phoenix, USA, 2005. ACTA Press.
- [2] Bluetooth Consortium. *Bluetooth Core Specification v2.0*, november 2004.
- [3] Crossbow. Crossbow, wireless sensor networks. available on <http://www.xbow.com/>.
- [4] Crossbow. Mica2 868, 916 mhz. available on http://www.xbow.com/Products/Product_pdf_files/Wireless_p
- [5] David Gay, Philip Levis, and Robert von Behren. The nesc Language : A Holistic Approach to Networked Embedded Systems. In *Proceedings of Programming Language Design and Implementation (PLDI)*, 2003. available on <http://csl.stanford.edu/pal/pubs/nesc-pldi03.pdf>.
- [6] Luc Hogie. *Delay Tolerant Networks : Modelling, Simulation and Broadcast-based Applications*. PhD thesis, Le Havre University (France) and Luxembourg University (Luxembourg), april 2007.
- [7] MexStream Inc. *XBee™ XBee-PRO™ OEM RF Modules, Product Manual v1.xAx*, october 2006.
- [8] P. Levis and D. Culler. Mate : A tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA*, Oct. 2002. To appear.
- [9] Philip Levis. TEP 111 : message.t. <http://www.tinyos.net/tinyos-2.x/doc/html/tep111.html>.
- [10] Philip Levis. Tinyos : An open-source operating system designed for wireless embedded sensor networks. available on <http://www.tinyos.net/>.
- [11] Philip Levis. TEP 116 : Packet Protocols, december 2004. available on <http://www.tinyos.net/tinyos-2.x/doc/html/tep116.html>.
- [12] Marco Mamei, Franco Zambonelli, and Letizia Leonardi. *Tuples On The Air : a Middleware for Context-Aware Computing in Dynamic Networks*. Los Alamitos, CA, USA, 2003.
- [13] Sun Microsystems. Java™ APIs for Bluetooth™ Wireless Technology (JSR-82), Specification Version 1.0a, april 2002.
- [14] Yves Métivier, Nasser Saheb, and Akka Zemmari. Analysis of a randomized rendezvous algorithm. *Inf. Comput.*, 184(1) :109–128, 2003.
- [15] National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES), november 2001. available on <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.
- [16] R. L. Rivest, A. Shamir, and L. M. Adelman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Technical Report MIT/LCS/TM-82, 1977.
- [17] Doug Simon, Cristina Cifuentes, Dave Cleal, John Daniels, and Derek White. Java™ on the bare metal of wireless sensor devices : the squawk java virtual machine. In *VEE '06 : Proceedings of the second international conference on Virtual execution environments*, pages 78–88, New York, NY, USA, 2006. ACM Press.
- [18] Arnaud Troël, Frédéric Weis, and Michel Banâtre. *Découverte automatique entre terminaux mobiles communicants*, october 2003.
- [19] The Free Encyclopedia Wikipedia. Mobile ad-hoc network. available on <http://en.wikipedia.org/wiki/MANet>.

Annexe A

Réponse passive à l'évolution du voisinage

Nom de la primitive	Catégorie	Conditions d' <i>applicabilité</i>	Résultat optimal
Détection de l'apparition d'un voisin unilatéral	Passive	Deux terminaux A et B tels que A n'est pas à portée de B.	Retourne vrai si et seulement si B est voisin unilatéral de A
Détection de la disparition d'un voisin unilatéral strict	Passive	Deux terminaux A et B tels que B est voisin unilatéral strict de A et annonce régulièrement sa présence.	Retourne vrai si et seulement si B n'est plus voisin unilatéral strict de A
Calcul de la puissance du signal reçu	Passive	Deux terminaux A et B tels que B soit voisin unilatéral de A.	Retourne un entier correspondant à la puissance du signal de B reçu en dBm.
Réception d'un message quelle que soit l'adresse de destination	Passive	Aucun contexte particulier.	Retourne le premier message qui parvient au terminal.
Réception d'un message dont l'adresse de destination est l'adresse de broadcast ou l'adresse locale	Passive	Chaque terminal possède un identifiant unique.	Retourne le premier message dont l'adresse de destination est l'adresse de broadcast ou l'adresse locale, qui parvient au terminal.
Réception d'un message dont l'adresse de destination n'est ni l'adresse de broadcast, ni l'adresse locale	Passive	Chaque terminal possède un identifiant unique.	Retourne le premier message dont l'adresse de destination n'est ni l'adresse de broadcast, ni l'adresse locale, qui parvient au terminal.
Détection d'un changement de localisation géographique (GPS)	Passive	Un terminal disposant d'un récepteur GPS.	Retourne la nouvelle position du terminal dès que celle-ci change.

TAB. A.1 – Réponse passive à l'évolution du voisinage

Annexe B

Action sur le voisinage

Nom de la primitive	Catégorie	Conditions d' <i>applicabilité</i>	Résultat optimal
Détection de la transformation d'un voisin unilatéral en voisin réciproque	Passive	Deux terminaux A et B tels que B soit voisin unilatéral de A.	Retourne vrai si et seulement si A et B sont voisins réciproques
Détection de la transformation d'un voisin réciproque en voisin unilatéral strict	Passive	Deux terminaux A et B voisins réciproques.	Retourne vrai si et seulement si B est un voisin unilatéral strict de A
Détection de la transformation d'un voisin non réciproque à portée en voisin réciproque	Passive	Deux terminaux A et B tels que A soit voisin unilatéral strict de B.	Retourne vrai si et seulement si A et B sont voisins réciproques
Détection de la disparition d'un voisin réciproque	Passive	Deux terminaux A et B tels que A et B soient voisins réciproques.	Cette primitive retourne vrai si et seulement si B n'est pas un voisin unilatéral de A
Signalement de l'apparition d'un nœud à ses voisins	Active	Aucun contexte particulier.	Tous les voisins de A sont avertis de la présence de celui-ci
Signalement de l'appartenance d'un nœud à un groupe à ses voisins	Active	Chaque terminal possède une identité unique.	Tous les voisins de A sont avertis de son appartenance au dit groupe
Vérification de la présence d'un terminal connu	Active	Chaque terminal possède une identité unique.	Cette primitive retourne vrai si et seulement si le terminal recherché est à portée
Envoi d'un message asynchrone adressé ou avec comme destination l'adresse de broadcast	Active	Aucun contexte particulier.	Le message envoyé parvient au(x) destinataire(s) du message si celui-ci (resp. ceux-ci) est (resp. sont) à portée du terminal
Envoi d'un message synchrone adressé ou avec comme destination l'adresse de broadcast	Active	Aucun contexte particulier.	Retourne vrai si et seulement si le destinataire reçoit le message

TAB. B.1 – Action sur le voisinage

Annexe C

Opérations locales destinées à faciliter les interactions dans les MANets

Nom de la primitive	Catégorie	Conditions d' <i>applicabilité</i>	Résultat optimal
Génération de paires de clefs asymétriques pour sécuriser les échanges	Active	Aucun contexte particulier.	Retourne une clef asymétrique.
Génération d'une clef symétrique pour sécuriser les échanges	Active	Aucun contexte particulier.	Retourne une clef symétrique.
Création d'un <i>Tuple Space</i>	Active	Aucun contexte particulier.	Retourne vrai si et seulement si la création du <i>Tuple Space local</i> a réussi.

TAB. C.1 – Opérations locales destinées à faciliter les interactions dans les MANets

Annexe D

Opérations complexes qui ne nécessitent pas une action sur le voisinage

Nom de la primitive	Catégorie	Conditions d'applicabilité	Résultat optimal
Estimation de la distance d'un terminal à partir de la puissance du signal	Passive	Deux terminaux A et B tels que B soit voisin unilatéral de A.	Retourne un entier correspondant à la distance séparant A et B.
Détection d'une perte imminente éventuelle de voisin unilatéral	Passive	Deux terminaux A et B tels que B soit voisin unilatéral de A.	Retourne vrai lorsqu'un terminal unilatéral est sur le point d'être perdu
Lecture de l'ensemble des <i>tuples</i> présents dans le <i>Tuple Space local</i>	Active	Un terminal A possédant un <i>Tuple Space local</i> .	Retourne l'ensemble des <i>tuples</i> localement présents dans le <i>Tuple Space local</i> qui correspondent au <i>tuple</i> passé en paramètre
Lecture de l'unique <i>tuple</i> correspondant au <i>tuple</i> passé en paramètre dans le <i>Tuple Space local</i>	Active	Un terminal A possédant un <i>Tuple Space local</i> .	Retourne l'unique <i>tuple</i> du <i>Tuple Space local</i> qui possède le même identifiant que celui du <i>tuple</i> passé en paramètre.
Suppression d'un <i>tuple</i> du <i>Tuple Space local</i>	Active	Un terminal A possédant un <i>Tuple Space local</i> .	Supprime le <i>tuple</i> correspondant au <i>tuple</i> passé en paramètre qui se trouvent dans le <i>Tuple Space local</i>
Association de l'arrivée d'un <i>tuple</i> avec le déclenchement d'un événement	Active	Un terminal A possédant un <i>Tuple Space local</i> .	Retourne vrai si et seulement si elle réussit à associer le déclenchement d'un événement à l'arrivée d'un <i>tuple</i> correspondant au <i>tuple</i> passé en paramètre
Suppression de l'association de l'arrivée d'un <i>tuple</i> avec le déclenchement d'un événement	Active	Un terminal A possédant un <i>Tuple Space local</i> .	Retourne vrai si et seulement si elle réussit à supprimer une association faite grâce à la primitive précédente.

TAB. D.1 – Opérations complexes qui ne nécessitent pas une action sur le voisinage

Annexe E

Opérations complexes qui nécessitent une action sur le voisinage

Nom de la primitive	Catégorie	Conditions d' <i>applicabilité</i>	Résultat optimal
Détection de l'ensemble des terminaux à portée	Active	Chaque terminal peut avoir une portée différente	Retourne l'ensemble des terminaux à portée
Détection de l'apparition d'un voisin réciproque	Passive	Aucun contexte particulier.	Retourne vrai si et seulement si l'apparition d'un nouveau voisin réciproque est détectée
Etablissement d'une connexion avec un voisin	Active	Deux terminaux A et B voisins réciproques.	Retourne vrai si et seulement si les terminaux A et B sont connectés
Fermeture d'une connexion avec un voisin	Active	Deux terminaux A et B connectés.	Retourne vrai si et seulement si A et B ne sont plus connectés
Détection d'une perte de connexion	Passive	Deux terminaux A et B connectés.	Retourne vrai si et seulement si A et B ne sont plus connectés
Etablissement d'un rendez-vous avec un voisin	Active	Au moins deux terminaux voisins réciproques.	Retourne vrai si et seulement si un rendez-vous est pris avec un ou plusieurs terminaux
Injection d'un <i>tuple</i> dans le <i>Tuple Space distribué</i>	Active	Un terminal A possédant un <i>Tuple Space local</i> .	Retourne vrai si et seulement si le <i>tuple</i> a été correctement injecté dans le <i>Tuple Space distribué</i>
Lecture des <i>tuples</i> présents dans le <i>Tuple Space distribué</i>	Active	Un terminal A possédant un <i>Tuple Space local</i> .	Retourne l'ensemble des <i>tuples</i> présents dans le <i>Tuple Space distribué</i> et qui correspondent au <i>tuple</i> passé en paramètre
Lecture de l'unique <i>tuple</i> correspondant à l'identifiant du <i>tuple</i> passé en paramètre dans le <i>Tuple Space distribué</i>	Active	Un terminal A possédant un <i>Tuple Space local</i> .	Retourne l'unique <i>tuple</i> parmi les <i>tuples</i> du <i>Tuple Space distribué</i> qui possède le même identifiant que celui du <i>tuple</i> passé en paramètre

TAB. E.1 – Opérations complexes qui nécessitent une action sur le voisinage