N° d'ordre : 3825

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET
D'INFORMATIQUE

Par **Lionel Barrère**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Étude et proposition de services dans les réseaux mobiles militaires de type MANet

---

Soutenue le : 9 juillet 2009

Après avis des rapporteurs :

Abderrahim Benslimane	Professeur
Frédéric Guidec	Maître de conférences (Hdr)

Devant la commission d'examen composée de :

Abderrahim Benslimane	Professeur	Examinateur
Richard Castanet	Professeur	Président
Serge Chaumette	Professeur	Directeur de thèse
Frédéric Guidec	Maître de conférences (Hdr)	Examinateur

– 2009 –

Remerciements

Je tiens tout d'abord à remercier tous les membres du jury pour l'intérêt qu'ils ont porté à mon travail et leur exprime ma profonde gratitude. J'étais très honoré de la présence de Frédéric Guidec rapporteur de ce manuscrit et le remercie pour les remarques constructives qu'il a su m'adresser et pour tout ce qu'il a pu m'apporter au cours de nos rencontres antérieures.

Je tiens également à exprimer ma sincère reconnaissance envers Richard Castanet qui a accepté de présider ce jury.

Mes prochains remerciements s'adresse à Serge Chaumette, mon directeur de thèse, tout d'abord pour m'avoir accueilli très tôt dans son équipe et pour avoir cru en moi au bon moment. Je n'oublierai pas les réflexions partant d'une idée sur un cahier de brouillon qu'elles aient abouties ou non à des résultats scientifiques.

Cette thèse ne serait pas ce qu'elle est sans l'enthousiasme des membres de la salle «CVT». Parmi les gens que j'ai pu y croiser je tiens particulièrement à remercier mes premiers coéquipiers Arnaud et Eve. Arnaud pour tous ses conseils qui m'ont permis de ne pas faire certaines erreurs et Eve pour sa bonne humeur permanente et sa convivialité. Je pense ensuite aux piliers des activités de la «CVT», pour les nombreux apéros, les sports en tout genre, les découvertes de jeux ou de musiques exotiques et pour tout le reste. Je remercie donc Fabien, Rémi, Jérémie, Julien, Martin, Jonathan et Achraf pour leur soutien psychologique direct et indirect.

Je pense maintenant à ceux pour qui le fonctionnement des ordinateurs reste mystérieux mais qui ont tout autant contribué à ces travaux. Je remercie tout d'abord mes parents pour m'avoir donner l'envie d'aller au bout sans m'y contraindre. Mes contributions estivales aux travaux de la ferme familiale m'ont permis de apprendre la valeur de l'effort de la manière la plus naturelle qui soit.

Enfin je remercie celle qui a motivé mes choix y compris les plus difficiles et qui m'a toujours soutenu. Merci Amélie, cette thèse n'est pas la tienne mais sans toi je ne serai vraisemblablement pas docteur.

Table des matières

1 Introduction générale **1**

2 Préliminaires sur les réseaux et état de l'art **9**
2.1 Définition des différents types de réseaux pertinents dans le cadre de notre étude . 9
2.1.1 Les réseaux statiques 9
2.1.2 Les réseaux ad hoc 10
2.1.3 Les réseaux mobiles ad hoc MANets (Mobile Ad hoc Networks) 10
2.2 Le routage dans les réseaux ad hoc 10
2.3 Les protocoles de routage dans les réseaux ad hoc 11
2.3.1 Le protocole AODV 11
2.3.2 Le protocole OLSR 11
2.4 L'approche opportuniste ou DTN (*Delay Tolerant Network*) dans les MANets . . 12
2.5 Quelques services classiquement étudiés dans les réseaux mobiles 14
2.5.1 Les systèmes de fichiers 14
2.5.2 La dissémination ou diffusion d'informations 16

I Première problématique : Collecte d'informations statiques **19**

Introduction **23**

3 Les réseaux de capteurs **27**
3.1 Les différents matériels 27
3.2 Notre plate-forme : Mica2 et TinyOS 28
3.2.1 Matériel : Crossbow Mica2 28
3.2.2 Le système d'exploitation : TinyOS 29
3.3 Exemples d'utilisation des réseaux de capteurs 30
3.4 Problèmes classiques et solutions existantes 34
3.4.1 Synchronisation temporelle 34
3.4.2 Localisation 37

4 Construction d'une vision continue des données collectées **41**
4.1 Description de l'application et spécificité du contexte militaire 41
4.2 Conception et problèmes algorithmiques 42
4.3 Définition et algorithmique du système proposé 43

4.3.1 Organisation du réseau 43
4.3.2 Capture, dissémination et perte éventuelle des mesures 45
4.3.3 Architecture du système d'interpolation 46
4.4 Résultats de l'évaluation de l'architecture d'interpolation 48
4.4.1 Capture des mesures 48
4.4.2 Interpolation des mesures 49

Conclusion **53**

II Seconde problématique : Travail collaboratif et information dynamique **55**

Introduction au problème **59**

5 Structure et modélisation de la solution proposée **61**
5.1 Structure de données utilisée 61
5.2 Gestion de la cohérence du document 65
5.2.1 États des nœuds de l'arbre 65
5.2.2 Transitions entre les différents états des nœuds 67
5.3 Versions et synchronisation entre utilisateurs 69
5.3.1 Gestions des versions 70
5.3.2 Synchronisation 71
5.4 Conclusion 72

6 Perte de droits, information floue et logique associée **75**
6.1 Problème 75
6.2 Le contexte hiérarchique militaire 76
6.3 Présentation de la solution pour récupérer des régions 76
6.4 Utilisation des cartes à puces et authentification de la hiérarchie 77
6.4.1 Introduction aux cartes à puces 77
6.4.2 Intérêt de l'utilisation des cartes à puces pour la réintroduction de verrous 78
6.5 Introduction de la hiérarchie 79
6.5.1 Prise en compte de la réintroduction de verrous au niveau de la structure des informations 79
6.5.2 Influence de l'introduction de nouveaux verrous sur la fusion de sous-parties 81
6.5.3 Algorithme de régénération de verrous 82
6.6 Information sur une surface 85
6.7 Logique associée aux changements d'états 86
6.7.1 Modification locale de l'état d'une information par constats 86
6.7.2 Transitions simples et informations exclusives 87
6.7.3 Synchronisation 87
6.8 Conclusion 89

7 Simulation **91**
7.1 Introduction . . . 91
7.2 Le Simulateur Madhoc . . . 92
7.3 Deux autres simulateurs existants . . . 93
7.3.1 ns-2 . . . 93
7.3.2 GloMoSim . . . 94
7.4 Les modèles de mobilité . . . 94
7.4.1 Le modèle de mobilité Random WayPoint (RWP) . . . 94
7.4.2 Le modèle de mobilité Human Mobility . . . 96
7.4.3 D'autres modèles de mobilité . . . 96
7.5 Codage de Shaadhoc dans Madhoc . . . 98
7.5.1 Adaptation de Shaadhoc à la discrétisation du temps . . . 98
7.5.2 Simulation du comportement des utilisateurs . . . 100
7.6 Résultats . . . 101
7.6.1 Validation qualitative . . . 101
7.6.2 Validation quantitative . . . 101
7.6.3 Mesures . . . 102
7.7 Conclusion . . . 112

8 Mise en œuvre de Shaadhoc **113**
8.1 Architecture logicielle . . . 113
8.2 Découverte du voisinage et des services . . . 115
8.2.1 Découverte de voisinage dans les réseaux mobiles IP (WiFi) . . . 115
8.2.2 Bluetooth et SDP . . . 117
8.3 Protocole de synchronisation . . . 118
8.4 Interface et scénario d'utilisation . . . 119
8.4.1 Verrouillage/Déverrouillage . . . 121
8.4.2 Découpage/Fusion . . . 122
8.5 Conclusion . . . 122

9 Conclusion générale **125**

Chapitre 1

Introduction générale

Le nombre de terminaux mobiles communicants (TMC) a fortement progressé ces dernières années et cette croissance s'est accompagnée de l'intégration de technologies comme WiFi [1] ou Bluetooth [2] dans de multiples appareils (téléphones mobiles, assistants personnels ou PDA, systèmes de navigation, etc) utilisés quotidiennement par une grande partie de la population. La connectivité supportée par ces équipements offre la possibilité de constituer des réseaux de façon spontanée, réseaux que l'on appelle généralement réseaux ad hoc.

Les réseaux ad hoc se forment de manière non planifiée et imprévisible, mais dans les configurations généralement étudiées leur composition et leur topologie évoluent la plupart du temps très peu. On peut alors les utiliser de la même façon que des réseaux statiques moyennant la mise en place d'un protocole de routage dédié tel que OLSR [3] ou AODV [4]. Les travaux présentés dans cette thèse sont au contraire centrés sur un sous-ensemble des réseaux ad hoc qui sont les réseaux de type MANet (*Mobile Ad hoc Network*) [5]. Ils ont la particularité d'être en constante évolution, leur composition et leur topologie changeant fréquemment en fonction des déplacements, des apparitions et des disparitions des équipements considérés. Ceci est particulièrement clair, par exemple, dans le cas des VANets [6] (*Vehicular Ad hoc Networks*), sous-ensemble des MANets dans lesquels les nœuds sont des véhicules.

L'utilisation des réseaux MANets soulève donc de nouveaux problèmes qui sont liés à cette non prédictibilité de la présence des nœuds et des liens de communication : un terminal présent à un instant donné peut ne plus l'être à l'instant suivant et peut même ne plus jamais réapparaître. Ce sont ces problèmes auxquels nous nous sommes confrontés dans cette thèse dont l'objectif est d'étudier et de proposer des services et des méthodologies de développement de services dans les réseaux mobiles militaires de type MANet. Nous nous plaçons en effet dans un contexte militaire, nos travaux se faisant en collaboration avec la DGA (Délégation Générale pour l'Armement) qui finance cette thèse.

Objectifs de cette thèse

Contributions méthodologiques

Dans le cadre de la numérisation du champ de bataille, les militaires en opération sur le terrain seront à court terme équipés de terminaux mobiles communicants (projet FÉLIN [7]). Les fantassins vont ainsi pouvoir devenir membres de réseaux ad hoc constitués dynamiquement. De nouvelles applications, qui restent à concevoir, pourront tirer parti de ces réseaux afin de leur fournir des services pertinents. L'objectif de cette thèse est d'apporter des guides méthodologiques et des applications de référence pour aider à la conception de ces nouveaux services en exploitant, lorsque cela est opportun, les spécificités du contexte militaire. Nous nous appuyons sur une approche applicative expérimentale pour élaborer nos propositions.

Problématiques de référence

Nous nous concentrons sur deux problématiques que sont la collecte d'information sur le terrain pour aider à la prise de décision par les opérationnels (personnels militaires) , et la manipulation collaborative d'un document. Ces deux sujets récurrents et représentatifs des travaux dans les réseaux mobiles, répondent à des besoins effectifs du domaine militaire. Ils sont porteurs comme nous le verrons de problématiques génériques du domaine.

Applications de référence

Les applications que nous avons conçues et développées permettent à la fois d'illustrer et aussi d'inspirer notre contribution méthodologique. Outre leur intérêt propre, elles serviront de référence lors de la conception de nouvelles applications qui auront à résoudre des difficultés similaires à celles rencontrées lors de leur réalisation. Ces applications sont des instanciations sur un problème particulier des problématiques standard présentées dans la section précédente. L'objet de la première application est de collecter des informations pour aider à la prise de décision et plus particulièrement de fournir aux opérationnels l'information la plus complète et la plus conforme à la réalité possible. La fonction de la deuxième application est la manipulation collaborative d'un document, dont elle permet l'édition de façon totalement distribuée et dont elle assure la cohérence à tout moment.

Problématiques de référence

Comme indiqué ci-dessus, les deux problématiques qui ont été retenues sont la collecte d'information pour l'aide à la décision et le travail collaboratif dans un réseau MANet.

Collecte d'information pour l'aide à la décision

Depuis quelques années maintenant, on cherche à obtenir une vision numérisée du monde réel. L'apparition de capteurs permettant de mesurer et de transmettre certaines valeurs physiques de l'environnement, comme la température ou l'amplitude des secousses sismiques, permet désormais de répondre à cette demande. Cette numérisation participe à ce que l'on appelle l'intelligence ambiante [8].

Dans le domaine militaire, cette connaissance du monde réel rend possible le développement d'applications dont l'objectif est d'aider les opérationnels dans leurs prises de décisions. Ces décisions peuvent avoir des conséquences en terme de coût ou de sécurité humaine (que ce soit dans le domaine civil ou militaire). C'est pourquoi les informations fournies par les applications doivent être les plus complètes et conformes à la réalité possible.

Or les phénomènes intrinsèques à l'environnement dans lequel sont placés les capteurs ont pour conséquence des défaillances matérielles ou des interférences qui induisent la rupture de certaines liaisons radio ou même la disparition (logique ou physique) de certains nœuds. En effet, les champs d'opération considérés peuvent être aussi variés qu'une forêt dans laquelle on souhaite identifier les départs de feu [9], les abords d'un cratère volcanique dont on souhaite surveiller l'activité sismique [10] ou bien encore un champ d'opération militaire dont on voudrait tirer certaines informations stratégiques [11]. De plus, les méthodes qui sont utilisées pour le déploiement des nœuds, comme le largage aérien, évitent certes une intervention humaine sur site, mais nécessitent l'auto-configuration du réseau. Dans ce type de contexte, on retrouve donc des caractéristiques identiques à celles des réseaux mobiles comme l'absence de planification et la volatilité des nœuds et des liens de communication. Les réseaux de capteurs peuvent ainsi être assimilés à des MANets de par les problématiques qu'ils soulèvent. Nous les abordons donc de manière similaire.

La survenue des défaillances que nous venons de décrire a pour conséquence la perte d'une partie de l'information qui peut pourtant être nécessaire à la prise de décision. Cette perte n'est pas une spécificité des réseaux de capteurs et peut survenir dans toute application mettant en jeu des moyens de communication et/ou du matériel non fiables. Mais alors qu'il s'agit d'un cas exceptionnel, ces pertes sont le cas général dans les MANets. Cette problématique est donc générique et naturellement pertinente dans les réseaux militaires. La démarche et l'approche que nous proposons, et qui sont mises en œuvre à travers une application de référence, pourront être reprises comme base méthodologique dans d'autres contextes applicatifs. Concrètement, nous proposons de reconstituer les valeurs perdues à partir de données historiques et/ou contextuelles. La solution mise en œuvre est basée sur une architecture modulaire utilisant des composants d'interpolation ou d'extrapolation.

Enfin, on souhaite généralement associer une date et une position géographique aux données collectées. Pour cela, certaines actions doivent être réalisées préalablement à toute prise de mesure, comme la synchronisation des nœuds et leur localisation géographique. Nous présenterons les choix de systèmes existants et les adaptations que nous avons réalisées et qui pourront là aussi être

réutilisées dans d'autres applications.

Travail collaboratif

Rappelons que dans le cadre de la numérisation du champ de bataille les fantassins vont être équipés de terminaux mobiles communicants (projet FÉLIN [7], Fantassin à Équipement de Liaison INtégré). Ces moyens de communication numériques permettront en particulier la constitution de réseaux MANets.

Un des intérêts majeurs de ces configurations exprimé par les militaires est l'accomplissement de tâches collaboratives comme l'édition d'un document partagé. On souhaite que plusieurs utilisateurs puissent collaborer simultanément à l'élaboration d'un unique document et que chacun profite des améliorations et plus généralement des modifications réalisées par les autres. La difficulté majeure provient de l'instabilité du réseau due à l'imprévisibilité des déplacements, des disparitions et des apparitions des nœuds le constituant, et qui peuvent même conduire à la création d'îlots. Dans ces conditions, aucun utilisateur ne peut avoir de vision globale du réseau et les modifications concurrentes du document peuvent remettre en question sa cohérence globale.

Les solutions proposées dans les réseaux statiques, comme les outils CVS [12] ou Subversion [13], reposent sur l'utilisation d'un serveur central. Après qu'un utilisateur a modifié sa copie locale d'un fichier, il soumet sa nouvelle version à ce serveur. Ces solutions mettent aussi en œuvre des systèmes de résolution de conflits entre versions divergentes qui reposent sur des décisions des utilisateurs.

Ce type de solution n'est pas envisageable dans un réseau mobile dans lequel on ne peut jamais garantir l'accès à un serveur central. Nous proposons donc de distribuer totalement non seulement la manipulation du document, mais aussi la gestion de son contrôle. Cette approche est d'autant plus pertinente dans un contexte militaire que tout fantassin peut disparaître suite à un événement imprévisible. Notre démarche visant à décentraliser le contrôle est générique et l'expérience acquise pourra être réexploitée dans d'autres applications. Concrètement, nous mettons en place un système de verrous décentralisés qui passent d'entité en entité (de fantassin en fantassin) au gré des connexions qui s'établissent entre elles. Nous proposons aussi une gestion de conflits automatique basée sur la notion originale d'information floue.

Applications de référence

Nous avons retenu deux applications de référence.

Collecte locale de température et synthèse d'une vision globale

L'application de référence que nous avons choisie pour étudier la problématique de la collecte d'informations consiste à mesurer un phénomène physique sur le terrain. En l'occurrence il s'agit

de l'évolution de la température en plusieurs points. Dans une application militaire réelle il pourrait par exemple s'agir du suivi d'un agent chimique, plutôt que de la température. Les mesures collectées sont ensuite assemblées pour servir d'aide à la décision.

Synchronisation temporelle et localisation

Deux opérations doivent être réalisées préalablement à la prise de mesures, à savoir la synchronisation temporelle des capteurs et leur localisation.

Synchronisation temporelle. On peut synchroniser les nœuds de façon relative ou de façon absolue. Simon *et al.* proposent une méthode relative dans [11] appelée FTSP (*Flooding Time Synchronization Protocol*). On peut l'utiliser pour ordonner des échantillons, mais elle ne fonctionne que lorsque le réseau est connexe. Nos travaux supposant la topologie du réseau incertaine, nous avons choisi une méthode de synchronisation absolue. Nous avons donc adapté l'algorithme FTSP [11] pour prendre en compte le fait que certains capteurs peuvent être équipés de GPS, auquel cas on peut utiliser l'horloge absolue fournie par ce matériel.

Localisation des nœuds. Nous avons étudié plusieurs méthodes de localisation existantes afin de déterminer la plus adaptée à nos besoins et nous avons retenu la solution MoteTrack proposée par Lorincz *et al.* à Harvard [14].

Traitement de l'information

Rappelons que les réseaux de capteurs sont instables par nature et de par leur contexte d'utilisation. Les avaries matérielles, temporaires ou définitives, sont fréquentes. La perte d'information, même si elle est ponctuelle dans l'espace et transitoire dans le temps, est donc un problème majeur.

Cependant, l'évolution des paramètres mesurés est généralement continue dans le temps et/ou dans l'espace. C'est le cas par exemple des variations de température. Les données perdues ont donc souvent une relation de dépendance forte avec les données effectivement collectées. Il en résulte que ces informations permettent d'estimer par le calcul les valeurs de celles qui ont été perdues (ou même la tendance à venir).

En nous basant sur cette constatation, nous proposons une architecture qui rend possible l'interpolation ou l'extrapolation des valeurs manquantes en utilisant différentes méthodes qui peuvent être inter-changées en fonction du contexte. Cette architecture a été déployée et testée sur des mesures de température en utilisant l'algorithme *Cubic Spline* [15] (employé par ailleurs en graphisme) pour réaliser l'interpolation. Cette méthode permet de générer un polynôme d'un ordre égal au nombre de valeurs disponibles et d'interpoler des valeurs grâce à cette fonction continue. Des résultats de l'utilisation de cette architecture sont présentés dans ce document.

D'un point de vue applicatif, l'architecture proposée est générique et peut être employée dans un cadre bien différent de celui des réseaux de capteurs. Elle est en fait adaptable à la gestion de toute source d'information qui peut être interpolée ou extrapolée par le calcul.

Mise à jour collaborative d'une carte stratégique

L'application de référence que nous avons choisie pour étudier la problématique du travail collaboratif dans un MANet consiste en l'édition distribuée d'une carte stratégique (figure 1.1). Chaque fantassin annote la carte en fonction de ses observations et ces annotations sont diffusées (d'une façon qui sera détaillée plus loin) aux autres fantassins de manière à assurer à chacun la vision la plus complète (la plus travaillée) possible de la situation des forces en présence sur le terrain.

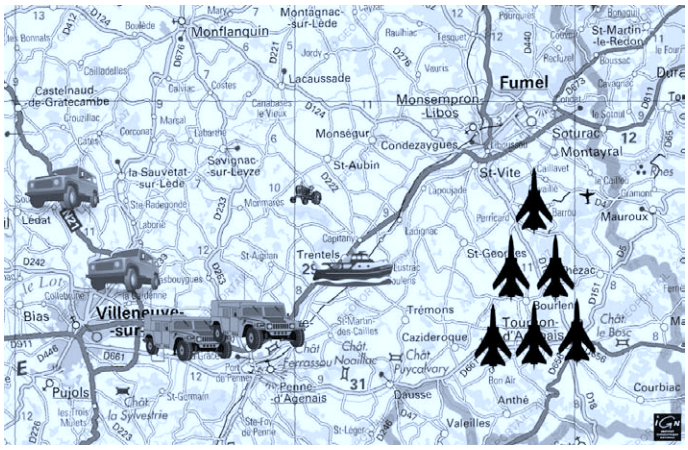


FIG. 1.1 – Exemple de carte stratégique pouvant être manipulée par notre application

Or, les réseaux MANets étant très volatiles, toute tâche collaborative s'avère complexe. Ceci est d'autant plus difficile lorsque cette tâche consiste à modifier un document de façon totalement distribuée, donc sur plusieurs nœuds. C'est cette problématique que nous étudions au travers de cette application.

La méthode que nous proposons consiste à scinder le document en différentes parties puis à répartir le travail afférent à chaque bloc entre les différents utilisateurs. Chaque partie du document est ensuite modifiée de manière exclusive par un seul utilisateur à la fois. Pour cela, un état, assimilable à un verrou, est adjoint à chaque partie. Au gré de leurs rencontres, les unités mobiles peuvent mettre à jour leurs versions du document et se transmettre les verrous. Chaque partie du document peut être redécoupée par l'entité mobile qui dispose du verrou associé, et ceci, de façon récursive, aussi finement que nécessaire. Les mises à jour circulent dans le réseau grâce à un processus de synchronisation entre les différentes versions du document global (détaillé dans cette thèse).

Il y a une contrepartie à cette gestion totalement décentralisée qui est ce que nous appelons la **perte de verrous**. Ce problème survient lorsqu'une entité qui dispose d'un verrou sur une partie de la carte disparaît définitivement. Ce verrou est alors perdu et cette partie du document n'est plus modifiable. Ce problème n'a pas de solution dans un réseau MANet générique constitué par des utilisateurs indifférenciés. Mais dans le contexte militaire certains nœuds peuvent disposer de droits supplémentaires du fait de leur position dans la hiérarchie. La réintroduction de verrous sur les parties considérées comme perdues par les supérieurs hiérarchiques permet alors de gérer ce problème de perte de verrous. Nous proposons aussi une solution pour gérer la cohérence du système lorsqu'un nœud (un fantassin) considéré à tort comme définitivement perdu revient dans le réseau. Nous introduisons pour cela la notion originale d'**information floue**. De façon simplifiée disons que cela consiste à ne pas écarter une information dont on ne sait pas dire si elle est valide, comme la présence éventuelle d'un ennemi. Il s'agit en quelque sorte d'un principe de précaution.

Du point de vue de la mise en œuvre, ce travail a conduit au développement de l'application Shaadhoc que nous avons amenée à un stade pré-industriel grâce à un financement obtenu dans le cadre de la labellisation Carnot du LaBRI. Cette application peut être réutilisée pour manipuler des documents autres que des cartes stratégiques, comme par exemple un texte devant être élaboré par plusieurs utilisateurs.

Organisation de ce document

Ce document est découpé en deux parties et un chapitre initial.

Le premier chapitre consiste en une présentation des réseaux et en particulier des réseaux ad hoc et des MANets. Nous nous intéressons aussi aux techniques de routage et aux problématiques classiques des services dans les réseaux mobiles.

La première partie est consacrée à l'étude de la collecte d'information en particulier dans un réseau de capteurs. Elle est composée de deux chapitres. Le premier présente ce type de réseaux et les problématiques auxquelles on est confronté lorsque l'on souhaite les exploiter. Le deuxième chapitre expose une solution pour tenter de construire une vision continue des données provenant d'un champ d'opération dans lequel est placé un réseau de capteurs, lorsque certains échantillons ne parviennent pas à l'application cliente.

Dans la deuxième partie nous présentons un système d'édition collaborative d'un document pour les réseaux MANets. Cette partie est constituée de quatre chapitres. Le premier est une modélisation du système sous-jacent et présente les structures de données utilisées et les règles qui régissent son fonctionnement. Le chapitre suivant étend ce modèle et montre comment nous proposons d'utiliser le contexte militaire pour résoudre un problème de perte d'information (en l'occurrence d'un verrou) lié à la nature des réseaux MANet. Nous présentons dans un troisième chapitre des résultats d'évaluation par simulation. Enfin un quatrième chapitre expose comment ce système a été mis en œuvre et a donné lieu au développement de l'application Shaadhoc.

Nous concluons en présentant un bilan des contributions de cette thèse qui sont à la fois applicatives et méthodologiques, et en présentant des perspectives de recherche futures.

Chapitre 2

Préliminaires sur les réseaux et état de l'art

Dans ce chapitre nous présentons les caractéristiques des réseaux statiques, des réseaux ad hoc et des MANets ainsi que les problématiques spécifiques associées. Nous définissons la notion de service dans les réseaux MANets et nous présentons un aperçu des solutions aux problèmes classiques.

2.1 Définition des différents types de réseaux pertinents dans le cadre de notre étude

Un réseau est un ensemble de terminaux interconnectés par un moyen de communication. Selon leur méthode de constitution et d'administration on distingue les réseaux statiques, les réseaux ad hoc et les MANets (Mobile Ad hoc Networks).

2.1.1 Les réseaux statiques

Les réseaux statiques [16] sont des réseaux qui ont vocation à être utilisés par des groupes humains dont la composition est connue à l'avance et évolue relativement peu rapidement, comme par exemple les collaborateurs d'une entreprise ou les membres d'un laboratoire de recherche. Ces réseaux, le plus souvent filaires, sont mis en œuvre grâce à du matériel permettant de créer une infrastructure, comme des commutateurs, des routeurs ou dans certains cas des points d'accès sans fil. Les terminaux sont le plus souvent des postes fixes mais peuvent aussi être des postes nomades ayant accès à un relais sans fil. Les services fournis aux membres de ces réseaux sont le plus souvent localisés sur des serveurs qui connaissent *a priori* chacun des utilisateurs et leur apportent ainsi des services spécifiques : accès à un système de fichiers, serveur de courrier électronique, etc.

2.1.2 Les réseaux ad hoc

Les réseaux ad hoc [17] sont des réseaux souvent temporaires constitués le plus souvent lors d'un rassemblement, par exemple une conférence. Ils utilisent la plupart du temps une infrastructure proche de celle fournie par un réseaux statique et à laquelle les terminaux accèdent via des points d'accès sans fil. Ils peuvent aussi se former, et c'est en cela qu'ils se différencient, sans aucune infrastructure, en utilisant uniquement des connexions reliant les terminaux entre eux. Une fois constitué, un réseau ad hoc a une composition relativement stable qui peut être conservée durant toute la durée de l'événement pour lequel il a été créé. En configuration ad hoc, les services sont portés par les terminaux qui le constituent et permettent par exemple de partager des données qui ne sont portées que par un seul d'entre eux. Les périphériques utilisés sont le plus souvent des ordinateurs portables ou des assistants personnels équipés d'une connexion sans fil WiFi ou Bluetooth.

La constitution et l'évolution des réseaux ad hoc que nous définissons ici sont très proches de ce que l'on trouve avec les logiciels d'échange peer-to-peer (comme Gnutella [18]).

2.1.3 Les réseaux mobiles ad hoc MANets (Mobile Ad hoc Networks)

Les MANets [5] sont le type de réseau le plus instable. Ils ne comportent pas d'infrastructure et les terminaux peuvent apparaître ou disparaître à tout moment suivant leurs capacités à communiquer ou leurs intérêts applicatifs ; ces changements peuvent donc être choisis ou subis. Par exemple, un réseau MANet peut-être constitué par les personnes à l'intérieur d'un commerce à un instant donné. Les terminaux utilisés sont très hétérogènes : ordinateurs portables, assistants personnels, téléphones ou même capteurs.

Les services spécifiques aux réseaux MANets ne peuvent pas reposer sur la supposition de la présence d'un terminal à un instant donné. L'accès à un service peut donc être discontinu, voire éphémère.

2.2 Le routage dans les réseaux ad hoc

Le routage est le mécanisme qui consiste à acheminer les messages des différents expéditeurs à leurs destinataires respectifs. Ceci passe par la définition de tables de routage qui permettent de déterminer pour chaque nœud susceptible de transmettre un paquet, le prochain nœud directement accessible auquel envoyer ce paquet. Dans un réseau statique les tables de routage sont figées.

Dans les réseaux ad hoc et MANets la composition et la topologie du réseau ne sont pas connues à l'avance. Il est donc impossible de construire des tables de routage statiques. Plusieurs protocoles ont donc été proposés pour maintenir les tables valides. On peut les classer en deux catégories distinctes qui sont les protocoles proactifs et les protocoles réactifs. Les protocoles

proactifs tels que OLSR (*Optimized Link State Routing Protocol*) [3] ou DSDV (Destination-Sequenced Distance-Vector) [19] maintiennent en permanence les routes vers les différents nœuds du réseau alors que les protocoles réactifs comme AODV (*Ad hoc On Demand Distance Vector*) [4] ou DSR (*Dynamic Source Routing*) [20] construisent une route « à la demande ».

Afin de préciser la notion de protocole de routage pour les réseaux ad hoc, nous détaillerons deux protocoles parmi les plus répandus, AODV et OLSR.

2.3 Les protocoles de routage dans les réseaux ad hoc

2.3.1 Le protocole AODV

AODV est le plus populaire des protocoles de routage réactif pour réseau ad hoc. Les routes sont définies à la demande, i.e. lorsqu'un nœud souhaite connaître le premier voisin auquel il doit s'adresser pour contacter un autre nœud du réseau. Chaque nœud conserve donc une table contenant le prochain saut vers chaque destination.

Lorsqu'un nœud *A* souhaite connaître la route vers un nœud *C*, il diffuse un paquet de type RREQ (Route Request). Lorsqu'un nœud *B* reçoit ce paquet et qu'il ne connaît pas la route vers *C* alors il ré-émet un paquet RREQ et enregistre que le message reçu provient de *A*. Lorsque *C* reçoit ce paquet il répond en *unicast* à *B* par un message de type RREP. A sa réception, *B* enverra à son tour un message RREP à *A* et ainsi une route bidirectionnelle (seul type de route supporté par AODV) est créée. Lorsqu'un nœud reçoit plusieurs fois un message RREQ d'une même séquence et d'une même origine, par exemple s'il existe un cycle, il ne considère que le premier message reçu pour constituer la route.

Lorsqu'un nœud détecte la disparition d'un de ses voisins qui était le prochain saut vers une destination donnée, alors il diffuse un message de type RERR afin d'invalider cette route.

La différence majeure entre AODV et DSR est qu'AODV effectue un routage « hop by hop », c'est à dire que les paquets ne contiennent que l'adresse de leur destination finale contrairement à DSR dans lequel les paquets contiennent la route complète, c'est-à-dire l'adresse de chaque nœud par lequel le paquet devra passer pour atteindre sa destination. Le routage est d'une certaine manière plus dynamique dans AODV car la route n'est pas déterminée à l'avance par l'émetteur.

2.3.2 Le protocole OLSR

OLSR est un protocole de routage proactif destiné aux réseaux ad hoc, développé à l'INRIA Rocquencourt, et standardisé par l'IETF (RFC3626). Il est devenu la référence en terme de routage dans ce type de réseaux. OLSR est basé sur l'état des liens et utilise des nœuds ayant des caractéristiques de connectivité particulières pour relayer les messages de contrôle : ce sont les relais multipoints ou (MPRs). L'élection des MPRs se fait en deux étapes, dans un premier temps les nœuds font une découverte de voisinage grâce à l'échange de messages « HELLO ». Ensuite

chaque nœud envoie périodiquement à ses voisins la liste de ses voisins. Chacun définit ses MPRs comme l'ensemble minimal des nœuds permettant d'atteindre tous les voisins à deux sauts.

Les MPRs sont ensuite les seuls nœuds autorisés à retransmettre les paquets de contrôle (de type TC) sur l'état des liens du réseau pour éviter un trop grand surcoût en terme de nombre de messages. A la réception de ces messages (TC) un nœud peut calculer la table de routage lui permettant d'atteindre tous les autres nœuds du réseau.

Conclusion sur les techniques de routage dans les réseaux ad hoc et MANets. L'intérêt de ces protocoles de routage est évident dans les réseaux ad hoc classiques pour lesquels il semble naturel de s'adresser à des utilisateurs à plusieurs sauts et ainsi d'atteindre l'ensemble du réseau comme on le fait dans un réseau statique. En revanche, on peut se poser la question de la fiabilité de tels systèmes dans des réseaux MANets dans lesquels on ne suppose rien sur la topologie. En effet, dans ce cadre on ne peut à aucun moment faire de supposition sur la présence d'un nœud après qu'une route vers celui-ci a pu être construite. D'ailleurs lorsque l'on se place dans un cadre de réseaux mobiles on préfère souvent joindre un service plutôt qu'un nœud particulier. C'est ce que nous verrons dans la section suivante.

2.4 L'approche opportuniste ou DTN (*Delay Tolerant Network*) dans les MANets

Nous avons présenté l'approche qui consiste à déterminer une route dans le réseau pour acheminer un message de l'expéditeur au destinataire. Ce mécanisme suppose qu'un chemin existe entre ces nœuds, c'est-à-dire qu'ils font partie d'une même composante connexe. Une autre approche, dite opportuniste ou DTN [21], tient compte du cas où le réseau peut être partitionné. On considère dans ce cadre que si aucune route complète n'existe à un instant donné entre deux nœuds, on peut tout de même faire transiter un message dans le réseau de proche en proche entre les nœuds accessibles dans l'espoir que l'un d'entre eux sera à un moment donné à portée du destinataire. Cette technique consiste en général à utiliser des nœuds potentiellement mobiles (*data mule*) qui ne sont pas, *a priori*, les desitnataires d'un message (appelé *bundle* dans [22]) pour le mettre dans un cache local et le retransmettre si l'occasion se présente. Cette méthode est appelée *store-and-forward*. La figure 2.1 présente un scénario utilisant une approche opportuniste pour transmettre un message du nœud E au nœud R en utilisant les nœuds T et U comme mules de données. Les nœuds porteurs du message sont grisés. La stratégie présentée dans cette figure consiste à transmettre systématiquement un message à tous les nœuds à portée.

Il existe plusieurs techniques de transmission de données opportunistes qui ont pour but d'être plus efficaces (en nombre de messages) que la simple diffusion évoquée précédemment. Vahdat et Becker définissent dans [23] un système de routage dit « épidémique » dans lequel chaque nœud sert de transmetteur de données et peut porter un certains nombre de messages en fonction de

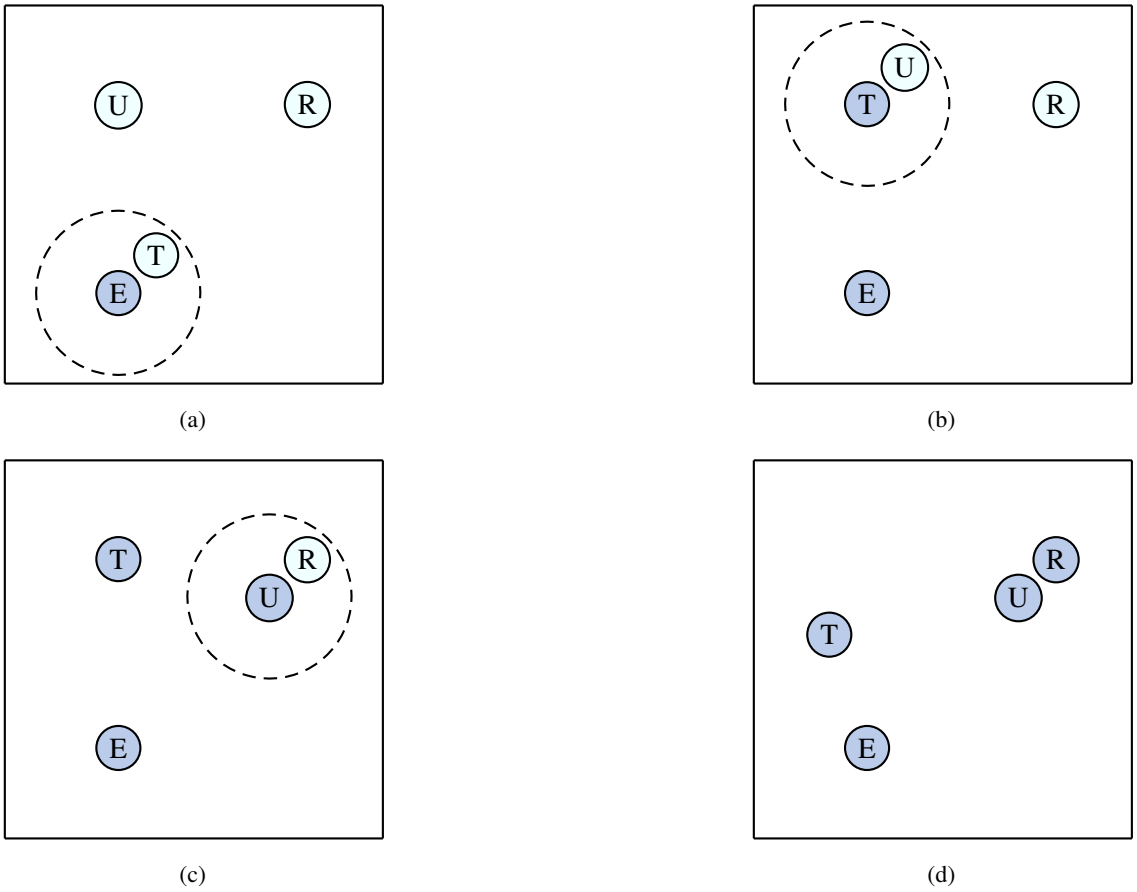


FIG. 2.1 – Stratégie de diffusion d'un message avec une approche opportuniste

ses capacités. Lorsqu'un nœud A rencontre un nœud B, A envoie la liste décrivant les messages qu'il porte à B, puis B demande à A les messages qui lui manquent. L'optimisation qui est faite dans ce système consiste à associer à chaque message un nombre maximum de sauts similaire à un TTL (lorsque ce nombre atteint la valeur de 1, il ne peut plus être délivré qu'au destinataire). La détermination de cette valeur est importante car plus ce nombre est grand plus le message va se propager rapidement mais aussi encombrer le réseau. Dans [24], Lindgren *et al.* proposent de transmettre le message aux nœuds qui ont la plus grande probabilité d'être sur le chemin du destinataire. Chaque nœud calcule une probabilité d'atteindre chaque autre nœud du réseau en fonction du nombre de fois où ils se sont rencontrés. Cette propriété est transitive, c'est-à-dire que si A a une forte probabilité de rencontrer B et C alors B a une probabilité moyenne de pouvoir transmettre un message à C. Un nœud ne transmet un message que si le voisin en question a une meilleure probabilité de le faire arriver à destination. Les résultats présentés dans cet article montrent que cette technique est aussi bonne que la précédente dans un contexte de mobilité aléatoire et que dans un contexte de mobilité communautaire, ce qui est la cible initiale de ces travaux, elle est bien meilleure. On trouve d'autres travaux dans la littérature, on peut par exemple citer les travaux de Harras *et al.* [25]. Ces recherches ont toutes pour principe l'utilisation de mules avec différentes stratégies de diffusion des messages suivant le contexte, comme dans l'article de Small et Hass dans lequel des baleines sont utilisées comme mules [26].

Les approches DTN sont très bien adaptées aux caractéristiques des réseaux MANets car elles utilisent les liens disponibles pour transmettre un message au lieu de supposer l'existence d'une route. Cependant suivant les contextes de mobilités dans lesquels on se trouve on peut ressentir le désir d'adapter la méthode de diffusion. Par exemple, Ott *et al.* proposent dans [27] une solution hybride qui laisse à l'application le choix d'utiliser AODV ou une approche oppurtuniste.

2.5 Quelques services classiquement étudiés dans les réseaux mobiles

Il existe principalement deux approches lors de la conception d'un service dans un réseau de type MANet. La première consiste à tenter d'offrir un service habituellement disponible dans un réseau statique, comme un système de fichiers partagés, en l'adaptant aux contraintes qui proviennent des caractéristiques du réseau en lui-même. L'autre approche consiste à concevoir des services qui prennent naturellement en compte dès leur conception les problèmes liés à l'instabilité du réseau. Les services conçus dans ce sens ne permettent souvent pas d'effectuer des opérations aussi complexes que les approches classiques mais fonctionnent quelle que soit la connectivité du réseau.

2.5.1 Les systèmes de fichiers

L'élaboration d'un système de fichiers partagés dans un réseau mobile constitue un défi important. Il est en effet très difficile de gérer les lectures/écritures dans un environnement présentant

une grande instabilité de connexion. Nous retenons deux propositions de la littérature qui sont très représentatives de ce domaine. Nous verrons que chacune d'entre elles impose des conditions supplémentaires sur le réseau pour fonctionner correctement.

Le système de fichier AdHocFS

AdHocFS [28] est un système de fichiers conçu pour fonctionner sur un modèle pair à pair dans un réseau constitué d'utilisateurs mobiles. Ils sont rassemblés en unités appelées groupes. La contrainte nécessaire à la constitution d'un groupe est que ses membres doivent pouvoir communiquer deux à deux (graphe complet). A partir de cette supposition, chaque groupe gère en interne un accès exclusif sur chaque fichier mais deux groupes peuvent modifier le même fichier en même temps. Les groupes dépendent d'un serveur de référence commun qui sert de point de synchronisation entre eux. Les connexions au serveur permettent de créer un historique des modifications de chaque fichier pour gérer, dans certains cas, le fait que deux groupes modifient le même fichier. Lorsqu'un conflit de version apparaît tout de même, alors le deuxième groupe se synchronisant avec le serveur de référence doit faire un choix, au niveau utilisateur, de la version à conserver et de même pour tous les groupes qui auraient modifié cette version. Les limitations quant à la mobilité dans AdHocFS se traduisent donc par la nécessité d'avoir un graphe complet dans un groupe mais aussi par le besoin de se connecter régulièrement à un serveur identifié pour éviter au maximum les conflits.

Le système de fichier MFS

MFS [29] est un autre système de fichiers destiné à des utilisateurs mobiles. Il fonctionne sur un schéma client/serveur dans lequel le serveur ne doit pas disparaître du réseau. La mobilité des nœuds se limite ici au fait que l'on accepte que les communications puissent être dégradées : les connexions avec le serveur peuvent être discontinues, le débit peut devenir trop faible, etc. Les accès de bas niveau aux fichiers, aussi bien en lecture qu'en écriture, se font par l'intermédiaire d'un cache. Lorsqu'un fichier n'est pas dans le cache, il est récupéré sur le serveur, et lorsqu'un fichier est fermé, les modifications éventuelles qui lui ont été apportées sont enregistrées sur le serveur. Les serveurs ne pouvant pas disparaître, la mobilité dans MFS est très limitée ; seuls les terminaux utilisateurs sont mobiles.

A ce jour, les recherches bibliographiques menées dans le cadre de cette thèse n'ont pas permis d'identifier un système de fichiers réellement adapté à notre définition des MANets. Les systèmes étudiés nécessitent de contraindre la mobilité à tel point que le contexte réseau en est lui-même modifié et devient alors statique d'une certaine manière.

2.5.2 La dissémination ou diffusion d'informations

Contrairement aux systèmes de fichiers, la diffusion d'information manipule des données statiques, non modifiables. Ce domaine se consacre à définir des méthodes pour qu'un nœud du réseau qui dispose d'une donnée puisse la diffuser, vers un nœud en particulier ou vers un ensemble de nœuds qui peut être éventuellement constitué de tous les membres du réseau.

La plate-forme DoDWAN (*Document Dissemination in Wireless Ad hoc Networks*)

La plate-forme DoDWAN [30] développée au VALORIA (Université de Bretagne Sud) est un système de dissémination de documents basé sur le paradigme *publish/subscribe*. Lorsqu'un utilisateur souhaite partager un document, on construit un descripteur qui contient des informations sur sa nature et son contenu. Ce descripteur est ajouté à un catalogue qui se compose des descripteurs des documents disponibles dans le cache local et des profils de document pouvant intéresser l'utilisateur local. Chaque participant diffuse régulièrement son catalogue afin d'informer ses voisins. Lorsqu'un nœud reçoit un catalogue qui contient un document l'intéressant et qu'il n'a pas ce document dans son cache local il envoie une requête en *unicast* au nœud ayant envoyé le catalogue. Ce dernier diffuse en réponse ce qui permet d'envoyer en une seule fois ce document à tous les nœuds de son voisinage qui sont intéressés. L'effet de bord positif induit par ce mécanisme est que des nœuds n'ayant pas envoyé de requêtes peuvent tout de même recevoir le document. Les envois de profils permettent aux utilisateurs d'adapter la composition de leur catalogue en fonction des profils de leurs voisins et des données présentes sur leur terminal.

Les applications de type communautaire

Les réseaux opportunistes tels que les MANets ont des caractéristiques qui permettent à des applications de type communautaire de voir le jour. Celles-ci permettent à des utilisateurs de s'échanger des informations directement sans passer par un fournisseur de services tiers. Elles reposent bien souvent sur un système de diffusion d'information qu'elles exploitent pour fournir un service applicatif de haut niveau. Par exemple, les utilisateurs d'un réseau routier disposent localement de données relatives aux conditions de trafic comme leur vitesse à un instant donné. Si cette information est partagée et cumulée, on peut en déduire la fluidité du réseau où les ralentissements éventuels [31]. Ce type d'application est appelé application communautaire et les recherches dans ce domaine s'étendent de la gestion de convois routiers [32] aux systèmes de partage de *playlist* de fichiers audio [33, 34] comme tunA [35, 36], développé au MIT. Certains industriels proposent même des logiciels de rencontre qui recherchent des personnes à portée radio [37].

D'un point de vu théorique il n'y a que l'imagination des concepteurs qui puisse limiter la création de logiciels de ce type. En pratique, les applications dans les réseaux MANets souffrent

du manque de modèle économique identifié. On peut imaginer proposer une application qui fournirait un service de localisation d'amis dans une ville. Celui-ci n'utiliserait que des communications deux à deux en transmettant les informations des positions géographiques de proche en proche entre les nœuds. Aujourd'hui cette application est peu lucrative pour les opérateurs téléphoniques qui ne pourraient que la vendre. Certaines sociétés tentent de profiter de ce créneau comme Aka'aki[38]. Leur modèle économique ne semble pas très bien défini, et on peut supposer que la gratuité actuelle sera remplacée à termes par un modèle financé par les utilisateur ou de la publicité.

Première partie

Première problématique : Collecte d'informations statiques

Table des matières

Introduction **23**

3 Les réseaux de capteurs **27**
3.1 Les différents matériels 27
3.2 Notre plate-forme : Mica2 et TinyOS 28
3.2.1 Matériel : Crossbow Mica2 28
3.2.2 Le système d'exploitation : TinyOS 29
3.3 Exemples d'utilisation des réseaux de capteurs 30
3.4 Problèmes classiques et solutions existantes 34
3.4.1 Synchronisation temporelle 34
3.4.2 Localisation 37

4 Construction d'une vision continue des données collectées **41**
4.1 Description de l'application et spécificité du contexte militaire 41
4.2 Conception et problèmes algorithmiques 42
4.3 Définition et algorithmique du système proposé 43
4.3.1 Organisation du réseau 43
4.3.2 Capture, dissémination et perte éventuelle des mesures 45
4.3.3 Architecture du système d'interpolation 46
4.4 Résultats de l'évaluation de l'architecture d'interpolation 48
4.4.1 Capture des mesures 48
4.4.2 Interpolation des mesures 49

Conclusion **53**

Introduction

Durant ces dernières années, l'envie de connaître et de mesurer un maximum de paramètres physiques a cru de façon considérable. Cette tendance relève surtout du besoin de contrôler son environnement. La baisse du coût des technologies permettant d'effectuer ce type de mesures facilite l'accès à ce matériel pour les laboratoires de recherche mais aussi pour le grand public qui profite ainsi des progrès techniques dans ce domaine. On peut par exemple citer la diffusion massive de stations météorologiques dans nos foyers. Ce besoin d'observer l'environnement se retrouve dans beaucoup de secteurs que ce soit dans le domaine des sciences naturelles ou de la sécurité avec la multiplication des systèmes de vidéo-surveillance et de détection d'intrusion.

Les informations collectées par les systèmes de mesure actuels [9, 10, 11] sont généralement transmises sans modification au système d'interprétation : nous parlerons dans la suite d'**information statique**. Une fois une donnée collectée, le système d'observation se charge de la transmettre aux entités qui souhaitent la récupérer. Les objectifs que l'on souhaite atteindre sont de maximiser le nombre de nœuds qui recevra cette information tout en minimisant les coûts en termes de nombre de messages échangés, de consommation d'énergie ou de temps de transmission. Cependant, on ne peut éviter que certains messages soient perdus. C'est pourquoi nous proposons dans cette partie une méthode pour reconstituer une information complète à partir de données partielles.

Les technologies actuelles

Les systèmes de mesure et d'observation du milieu nécessitent d'utiliser un matériel spécifique qui soit doté d'une capacité d'observation et de la possibilité de communiquer. Depuis les années 2000, nous avons assisté à l'apparition de nouvelles technologies qui facilitent le développement d'applications répondant aux besoins exprimés plus haut. Parmi ces technologies nous en citerons deux principales qui sont les réseaux de capteurs [39] qui permettent de mesurer des phénomènes et la technologie RFID [40] qui permet d'attacher de l'information numérique à des objets.

La technologie RFID

La technologie RFID (*Radio Frequency IDentification*) permet d'attacher de l'information numérique à un produit ou à une personne. Le matériel utilisé par la technologie RFID se compose de deux types d'éléments qui sont les marqueurs ou radio-étiquettes (Tags) et les lecteurs. Le marqueur est lui-même composé d'une antenne qui permet la communication radio et aussi, le plus souvent, l'alimentation électrique par induction électromagnétique, même s'il existe quelques composants auto-alimentés [41]. Cette technologie permet de récupérer l'information des radio-étiquettes attachées à des objets d'une manière automatique. Les lecteurs sont généralement connectés par l'intermédiaire de postes de travail à des bases de données classiques dont les entrées sont modifiées en fonction du contenu des marqueurs RFID. On parle aussi de l'*« Internet of Things »* [42] pour ce qui est du fait de lier de l'information numérique à notre environnement (naturel ou matériel). La technologie RFID peut par exemple remplacer les codes barres dans la gestion des stocks ou être utilisée dans le cadre du contrôle d'accès à certains bâtiments. Ainsi l'utilisation des RFIDs permet à la fois d'enregistrer les entrées/sorties des produits dans un stock pour un commerce, de gérer l'accès à des bâtiments comme un laboratoire de recherche ou d'aider au chronométrage d'épreuves sportives comme le marathon de Paris (Système ChampionChip® [43]). Cette technologie a pour avantage majeur son faible coût de fabrication qui permet d'ores et déjà de la déployer à grande échelle. Par contre, la nécessité d'utiliser des lecteurs « actifs » ne permet pas de l'utiliser pour déployer des applications autonomes. La technologie RFID n'est donc pas destinée au développement d'applications totalement décentralisées qui permettraient à des marqueurs de s'échanger directement des données. Enfin, la grande diffusion des RFIDs pour gérer des systèmes d'information ayant de grands enjeux économiques pose la question de la sécurité et de la confidentialité des données qu'ils contiennent [44].

Les cartes à puce sans contact

L'utilisation des cartes à puce sans contact [45] est très différente de celle des marqueurs RFID. Elles sont plus utilisées dans des applications nécessitant un degré de sécurité élevé, car elle ont des capacités cryptographiques importantes.

Si l'on abstrait le domaine d'application, le principe de fonctionnement des cartes à puce est très semblable à celui des radio-étiquettes. En effet, elles nécessitent aussi l'utilisation de lecteurs pour la communication avec un poste de travail et/ou pour leur alimentation électrique. Ainsi, même si les puissances de calculs sont très différentes, les infrastructures nécessaires et les capacités de l'une ou l'autre de ces technologies sont très proches en terme de déploiement de services mobiles.

Les réseaux de capteurs

Les réseaux de capteurs [39] constituent un nouveau type de plates-formes qui est apparu ces dernières années. On parle aussi de *Wireless Sensor Networks ou WSN*. Un réseau de capteurs est constitué d'entités autonomes du point de vue de leur alimentation, de leur système de communication et de leur unité de calcul. Ces nœuds peuvent être équipés de composants capables de mesurer des paramètres physiques du milieu dans lequel ils sont placés ; ce sont ces équipements que l'on appelle capteurs. Ceux-ci peuvent avoir des fonctions très variées comme des mesures de température, de luminosité, d'accélération, de pression atmosphérique. Certains nœuds peuvent même être équipés d'un système de positionnement géographique (GPS).

L'utilisation de réseaux de capteurs permet le développement d'applications distribuées et mobiles. Les nœuds sont en effet totalement autonomes et peuvent de ce fait évoluer sans dépendre d'un quelconque système externe.

Présentation de notre solution

Nous présentons dans la suite une méthode originale destinée à assister la collecte d'information sur un réseau de capteurs. Cette méthode n'utilise pas de routage ni d'adressage et elle n'emploie que des communications deux à deux. Nous proposons une architecture générique qui peut être utilisée pour suivre l'évolution de phénomènes physiques dans un contexte de champ de bataille, de situation de crise, de catastrophe naturelle ou de tout phénomène nécessitant une vision globale d'un site ou d'un terrain opérationnel. Elle permet de faire face dans une certaine mesure à la perte de données qui est assez fréquente dans un tel système. En effet des pertes peuvent découler d'interférences radio, de la destruction (volontaire ou involontaire) de nœuds, ou de défaillances telles qu'un manque d'énergie. Ces problèmes peuvent subvenir de façon temporaire ou permanente.

Chapitre 3

Les réseaux de capteurs

Dans un premier temps nous présenterons plus en détails les réseaux de capteurs d'un point de vue matériel et logiciel. Ensuite nous présenterons des systèmes existants, les problématiques récurrentes que sont la synchronisation temporelle et la localisation des nœuds et leurs solutions actuelles. Par la suite nous présenterons notre solution qui inclut la modification d'un algorithme de synchronisation temporelle existant, la description de notre architecture et ses résultats.

3.1 Les différents matériels

Chaque nœud pouvant intervenir dans la composition d'un réseau de capteurs comporte deux parties distinctes :

- le nœud à proprement parler qui est constitué d'un microcontrôleur utilisé comme unité de calcul et d'une unité de transmission radio pour la communication ;
- différents types de capteurs qui permettent de mesurer des grandeurs physiques comme la température ou l'intensité lumineuse.

L'offre disponible sur le marché s'est récemment étoffée et diversifiée, tant en terme d'unités de calcul que de moyens de communication. Si la plupart des équipements sont munis de microcontrôleurs basiques tel qu'un Atmel ATMEGA 128L [46] par exemple pour la série des Mica2 [47] ou MicaZ [48], certains nœuds sont dotés d'unités de calcul plus performantes comme des Intel XScale [49] tel qu'on les trouve sur la plupart des assistants personnels (PDA). D'autre part on retrouve des systèmes de communication variés qui peuvent être propriétaires et qui utilisent les bandes de fréquences 433/915Mhz ou bien des technologies standardisées comme Bluetooth [2] ou la norme IEEE 802.15.4 [50] qui sert de base à Zigbee [51]. Cette dernière norme prend une place de plus en plus importante au sein du marché (tableau 3.1 page suivante). Cette technologie utilise la bande des 2,4 Ghz et a l'avantage d'être mise en œuvre sur des composants à très faible consommation d'énergie. La plupart des nœuds destinés aux réseaux de capteurs se

Modèle	Connectivité	Processeur/Microcontrôleur	Caractéristiques
Crossbow Mica2	433/915Mhz	Atmel ATMEGA 128L	8-bit, 16 Mhz
Crossbow MicaZ	2.4Ghz, 802.15.4 (couche basse de ZigBee)	Atmel ATMEGA 128L	8-bit, 16 Mhz
Mulle	Bluetooth	Renesas M16C/62 MCU	16-bit, 10Mhz
TelosB	2.4Ghz, 802.15.4	Texas Instruments MSP430	16-bit, 8/16Mhz
BTnode	433/915Mhz et Bluetooth	Atmel ATMEGA 128L	8-bit, 16 Mhz
Imote (Intel Mote)	2.4Ghz, 802.15.4	Intel XScale	32-bit, 416Mhz
Sun SPOT	2.4Ghz, 802.15.4	Arm920T	32-bit, 180Mhz

TAB. 3.1 – Différents modèles de nœuds de réseau de capteurs représentatifs du marché

contentent d'intégrer des composants 802.15.4 et laissent la charge éventuelle des protocoles ZigBee à un microcontrôleur. Dans ce cas, celui-ci a la responsabilité d'implémenter cette couche ou d'utiliser l'implémentation de son choix.

Le tableau 3.1 récapitule les matériels les plus répandus sur le marché.

3.2 Notre plate-forme : Mica2 et TinyOS

3.2.1 Matériel : Crossbow Mica2

Le réseau de capteur que nous utilisons se compose de nœuds Crossbow Mica2 (figure 3.1(a)) développés à l'origine à l'université de Berkeley. Cette plate-forme est l'une des plus utilisées au moment où ce document est écrit et était une des seules disponibles dans le commerce lorsque ces travaux ont été initiés. Chaque Mica2 est équipé d'un processeur cadencé à 7,37 Mhz et doté de 4ko de RAM, de 128 ko de mémoire flash et d'un transmetteur radio à 433 MHz.

Les Mica2 peuvent être équipés de plusieurs types de circuits intégrés permettant d'effectuer des mesures de phénomènes naturels. Chaque carte supporte plusieurs capteurs sous forme de composants électroniques. L'équipe SOD du LaBRI dispose de Mica2 munis de deux types de cartes :

1. La carte **Crossbow MTS300** est équipée d'un capteur de température, d'intensité lumineuse, d'un microphone et d'un buzzer à 4kHz (figure 3.1(b)).

2. La carte **Crossbow MTS420**, beaucoup plus complète, comporte un capteur d'humidité et de température, un capteur de lumière ambiante, un baromètre, un accéléromètre deux-axes et une puce GPS (Sirf Star II) (figure 3.1(c)). Le coût d'une telle carte est environ quatre fois supérieur à celui d'une MTS300, soit environ 400$ contre 100$.

(a) Mica2 (b) MTS300 (c) MTS420

FIG. 3.1 – Un Mica2 et des cartes de capteurs

3.2.2 Le système d'exploitation : TinyOS

TinyOS [52, 53] est un système d'exploitation *open-source* offrant un mécanisme de programmation événementiel et conçu pour les réseaux de capteurs. Ce système repose sur une architecture à composants. En effet, TinyOS est constitué de plusieurs modules disponibles pour les applications et offrant des fonctions de capture de mesures ou de communication. Il n'existe pas d'exécutable pour le noyau du système, il est construit au moment de la compilation de l'application en fonction des composants qu'elle utilise. Le langage de programmation associé, le nesC [54], qui est une extension du langage C, permet de déclarer les composants ainsi que les liens qui les unissent et de faire l'association code/composants. Un code écrit en nesC se compose d'une partie de déclarations qui a un rôle assez proche de celui d'un fichier déclarant des interfaces (*header ou .h en C*) et d'une partie code dans laquelle on implémente les fonctions nécessaires au programme.

Pour illustrer notre propos, nous utilisons l'application Blink qui fait partie des exemples fournis dans la distribution de TinyOS. Cette application fait clignoter à intervalle régulier la diode rouge dont sont équipés les noeuds de type Mica2 (ces noeuds sont en fait équipés de trois LEDs, une rouge, une jaune et une verte). Son implémentation se compose de deux fichiers (présentés ci-après) qui sont *Blink.nc* (fig 3.2) pour la partie interface et *BlinkM.nc* (fig 3.3) pour la partie réalisation. On peut noter dans le fichier *Blink.nc* (fig. 3.2 page suivante) la déclaration des composants qui seront utilisés par le programme (ligne 4). TinyOS implémente certaines fonctions basiques comme la gestion de *timers* ou l'accès aux leds. Les liens entre les différentes interfaces et les différentes implémentations sont déclarés des lignes 5 à 8. La figure 3.4 représente les redirections effectives vers les implémentations conformément au fichier *Blink.nc*. On remarque que ces redirections sont injectives, par exemple un appel à l'interface principale de contrôle de l'application *Main.StdControl* sera redirigé vers *SingleTimer* et vers *BlinkM*. Le fichier *BlinkM.nc* (fig. 3.3 page 31) commence quant à lui par les déclarations des interfaces qui y sont implémentées puis celles qui y sont utilisés. L'implémentation des fonctions du programme à proprement parler se situe dans la partie *implementation* (à partir de la ligne 10). On distingue dans cette section de code deux types de fonctions. On trouve d'une part des fonctions préfixées par le mot réservé

command. L'appel à ces fonctions se fait de manière classique grâce à l'utilisation du mot réservé *call*. Celles-ci peuvent être synchrones ou asynchrones. Dans ce dernier cas leur définition doit être préfixée par le mot *async* et leur résultat sera récupéré sous la forme d'un événement. Une fonction préfixée par le mot réservé *event* (cf. ligne 26) sera exécutée lors de la réception d'un événement. Dans cet exemple l'événement est de la forme *Timer.fired*. Il a été défini à la ligne 19 qui a pour effet d'armer un *timer* déclenché chaque seconde.

Blink.nc

```
1: configuration Blink {
2: }
3: implementation {
4:   components Main, BlinkM,SingleTimer, LedsC;
5:   Main.StdControl -> SingleTimer.StdControl;
6:   Main.StdControl -> BlinkM.StdControl;
7:   BlinkM.Timer -> SingleTimer.Timer;
8:   BlinkM.Leds -> LedsC;
9: }
```

Blink.nc

FIG. 3.2 – Source du fichier Blink.nc

3.3 Exemples d'utilisation des réseaux de capteurs

L'utilisation de capteurs pour la surveillance de paramètres environnementaux a été très tôt au coeur des préoccupations militaires. Dès 1949 les États-Unis ont entamé des recherches dans le but de déployer des capteurs hydrophoniques au fond de l'océan pour suivre les déplacements des sous-marins. Ce projet a conduit à la constitution du système SOSUS [55] (*SOund SUrveillance System*) qui fut opérationnel dès 1961 et utilisé pendant la guerre froide. Il est encore utilisé par la NOAA (*National Oceanographic and Atmospheric Administration*) avec pour mission l'observation de phénomènes naturels tels que les séismes sous-marins [56] ou l'activité animale [57].

Au début des années 80 la notion de réseau de capteurs distribués (*DSN : Distributed Sensor Network*) est apparue et a été précisée lors du « Distributed Sensor Nets workshop » [58]. La figure 3.5 montre un nœud mobile qui avait pour objectif une démonstration de ce nouveau type de réseau. Le contexte d'utilisation de ce matériel était le suivi d'avions volant à faible altitude. Ces images permettent de bien se rendre compte des dimensions d'un nœud il y a vingt ans et laisse imaginer le coût que pouvait avoir un tel matériel.

Plus récemment, en 2001, grâce à la miniaturisation et au développement des MEMS [59] (*MicroElectroMechanical System*) la DARPA (*Defense Advanced Research Projects Agency*) a lancé le programme SensIT [60] qui a pour objectif, d'une part, de développer des architectures réseaux basées sur un principe ad hoc, et d'autre part, de traiter les informations relevées afin de disposer des données les plus fiables possible.

BlinkM.nc

```
1:  module BlinkM {
2:    provides {
3:      interface StdControl;
4:    }
5:    uses {
6:      interface Timer;
7:      interface Leds;
8:    }
9:  }
10: implementation {
11:
12:   command result_t StdControl.init() {
13:     call Leds.init();
14:     return SUCCESS;
15:   }
16:
17:   command result_t StdControl.start() {
18:     // Start a repeating timer that fires every 1000ms
19:     return call Timer.start(TIMER_REPEAT, 1000);
20:   }
21:
22:   command result_t StdControl.stop() {
23:     return call Timer.stop();
24:   }
25:
26:   event result_t Timer.fired()
27:   {
28:     call Leds.redToggle();
29:     return SUCCESS;
30:   }
31: }
```

BlinkM.nc

FIG. 3.3 – Source du fichier BlinkM.nc

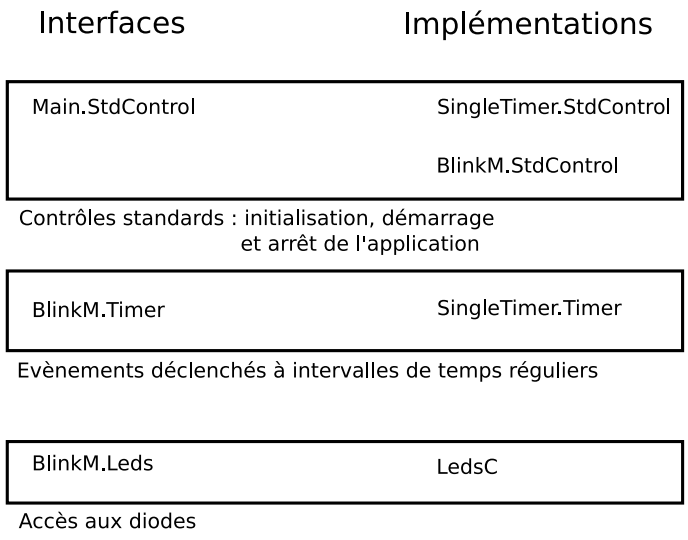


FIG. 3.4 – Redirections des interfaces de TinyOS vers les implémentations utilisées par l'application Blink

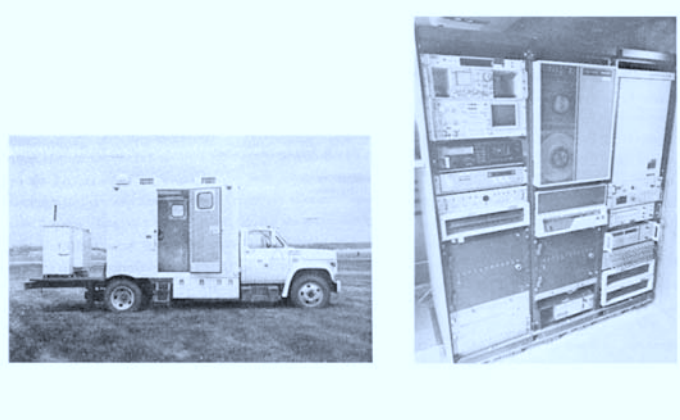


FIG. 3.5 – Exemple de nœud d'un réseau de capteurs destiné au suivi d'avions en 1985

Les réseaux de capteurs suscitent aussi beaucoup d'intérêt dans le domaine civil car ils permettent de créer des systèmes d'observation de phénomènes naturels, à faible coût, et sans incidence forte sur le phénomène observé. De nombreux domaines d'application profitent ainsi de l'apport des réseaux de capteurs, que ce soit en biologie, en climatologie ou en géologie. En effet, les sciences naturelles semblent pleinement exploiter les possibilités des réseaux de capteurs, que ce soit dans le but d'alerter les populations en cas d'éruptions volcaniques [61], d'observer des animaux (comme des zèbres [62]) ou d'assister les agriculteurs en leur apportant une mesure précise des phénomènes qui influent sur le travail de certaines cultures [63].

L'observation des animaux peut être facilitée en réduisant la présence de l'homme et en évitant ainsi son impact sur le comportement animal. L'expérience décrite dans [64] a pour objectif d'observer le comportement d'oiseaux appelés pétrels à l'intérieur de leur terrier sur une île (*Great Duck Island*). Les auteurs s'intéressent plus particulièrement à mesurer la température en présence ou non des adultes dans le nid, notamment lors de la couvée des œufs. Des capteurs sont donc placés à l'intérieur des terriers à observer. L'homme ne dérange ensuite plus les animaux, sa présence étant inutile pour réaliser les mesures. Le système présenté est basé sur une architecture hiérarchique constituée de trois éléments distincts :

1. les capteurs, disposés en plusieurs îlots topologiques ;
2. une passerelle par îlot ;
3. une station de base sur l'île reliée à Internet via une connexion satellite.

L'utilisation des réseaux de capteurs permet aussi d'observer des phénomènes sans risquer de compromettre l'intégrité physique des personnes qui sont chargées de la collecte d'information, comme dans la détection d'éruption volcanique décrite dans [10].

Huang *et al.* [65] ont développé un système d'assistance pour des secouristes qui a pour objectif de simplifier la localisation de randonneurs ou d'alpinistes. Ce système appelé *CenWits* (figure 3.6) est basé sur l'utilisation de capteurs positionnés sur ces randonneurs et qui enregistrent localement à intervalles de temps réguliers la position GPS de chacun d'entre eux. Cette collecte permet de constituer un historique de points de passage pour chaque randonneur. Lorsque deux randonneurs se rencontrent, ils échangent les données qu'ils ont collectées de façon à les disséminer sur un maximum de nœuds. Lorsqu'un randonneur passe près d'une station de base (AP sur la figure 3.6) qui est connectée à un réseau global comme Internet, le capteur qu'il embarque transmet les enregistrements qu'il contient afin qu'un système centralisé puisse reconstituer ou estimer le trajet de chaque personne ce qui permettra de les localiser en cas de disparition.

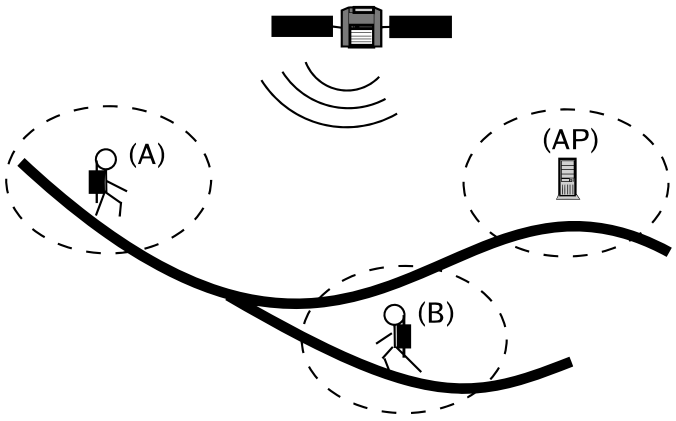


FIG. 3.6 – Représentation de l'infrastructure du système *Cenwits (figure extraite de [65])*

Dans [11], Simon *et al.* proposent un système de localisation d'un tireur isolé (*sniper*) dans un contexte urbain. Cette application permet de reconstituer la trajectoire d'une balle en se basant sur l'enregistrement des dates auxquelles chaque capteur perçoit l'onde de choc produite au passage de la balle grâce à un micro. Après analyse des données de manière centralisée, la position du tireur est estimée. L'aire probable de présence du tireur est représentée par une surface circulaire. L'efficacité de cette application repose sur la précision de l'horloge partagée entre les nœuds : pour atteindre un bon résultat les capteurs se synchronisent en temps en utilisant l'algorithme FTSP [66] qui sera décrit en section 3.4.1 de ce document. Ce système n'utilise pas non plus de configuration automatique pour la localisation des nœuds afin d'éviter d'introduire une incertitude supplémentaire. Les capteurs sont donc positionnés manuellement à des emplacements définis à l'avance.

3.4 Problèmes classiques et solutions existantes

3.4.1 Synchronisation temporelle

Un des problèmes majeurs lors de la prise de mesures dans un réseau de capteurs est la synchronisation temporelle des nœuds. Celle-ci est nécessaire pour pouvoir ordonner dans le temps les échantillons de mesure. On ne peut pas supposer l'existence d'une horloge globale sur laquelle chaque nœud pourrait se synchroniser car cela nécessiterait que chaque nœud puisse se connecter à un point central, hébergeant par exemple un serveur NTP [67]. De plus, une fois synchronisée, l'horloge des capteurs dévie assez fortement, de l'ordre de 40μs par seconde [66], et cela de façon non uniforme.

Le problème de la synchronisation temporelle est ici d'autant plus important que le réseau de capteurs peut être disjoint, c'est-à-dire composé de plusieurs îlots et que nous souhaitons tout de même que les échantillons puissent être replacés dans le temps relativement à une même référence. Nous considérerons deux types d'horloge : une horloge globale (virtuelle) qui est celle sur laquelle nous souhaitons que les noeuds se synchronisent et une horloge locale à chaque nœud qui correspond à son horloge interne. La synchronisation a pour objectif de fixer l'horloge interne d'un nœud sur l'horloge d'un nœud de référence. Dans le cas du nœud de référence l'horloge globale est égale à son horloge interne. Une autre solution consiste à enregistrer le décalage entre l'horloge interne de tout nœud et celle du nœud de référence.

Maroti *et al.* identifient dans [66] les paramètres à prendre en compte lorsqu'on veut réaliser une synchronisation par envoi de messages :

1. **Temps d'envoi**
   Le temps d'envoi est la durée nécessaire pour assembler le message et faire une requête à la couche MAC. Ce temps est de l'ordre d'une centaine de millisecondes.

2. **Temps d'accès**
   Le temps d'accès est le délai pour que le canal de transmission devienne disponible et que la transmission puisse commencer. Il peut varier de quelques millisecondes à quelques secondes en fonction de l'occupation du réseau.

3. **Temps de transmission**
   Ce temps est la durée nécessaire à l'émetteur pour transmettre son message sur le canal. Il est de l'ordre de quelques dizaines de millisecondes.

4. **Temps de propagation**
   Le temps de propagation est le temps mis par l'onde radio pour se déplacer de l'émetteur jusqu'au récepteur. Il est inférieur à une microseconde si l'on considère une plate-forme de mica2 dont la portée est inférieure à 300m.

5. **Temps de réception**

Le temps de réception est l'équivalent pour le récepteur du temps de transmission de l'émetteur (cf fig 3.7). Il correspond à la récupération du message sur le canal.

6. **Temps de prise en compte du message**
   Le temps de prise en compte du message est le délai entre la fin de la réception du message et la notification de son arrivée.

7. **Temps de récupération de l'interruption radio**
   Le temps de récupération de l'interruption radio est le délai entre le moment où la puce radio lève une interruption et le moment où le microcontrôleur la traite. Il est de l'ordre de 5μs sur la plate-forme Mica2.

8. **Temps d'encodage**
   Le temps d'encodage est le temps que met la puce radio pour convertir le message numérique en onde radio. Il est de l'ordre d'une centaine de microsecondes.

9. **Temps de décodage**
   Le temps de décodage est la durée nécessaire pour que la puce radio (côté récepteur) reconstitue le message binaire à partir de l'onde radio. Il est de l'ordre d'une centaine de microsecondes.

10. **Temps d'alignement d'octet**
    L'alignement sur le premier octet du message est nécessaire car certaines puces radio ne sont pas capables de capturer le début d'un message dans un flux de données. Il faut donc que le récepteur détecte le début du message en faisant un décalage de bits pour s'aligner sur l'octet de début de message. Le temps nécessaire à la réalisation de cette opération peut être calculé en fonction du décalage effectif et de la vitesse de la transmission radio. Il varie de 0 μs pour un décalage de 0 bit à 365μs pour un décalage de 7 bits.

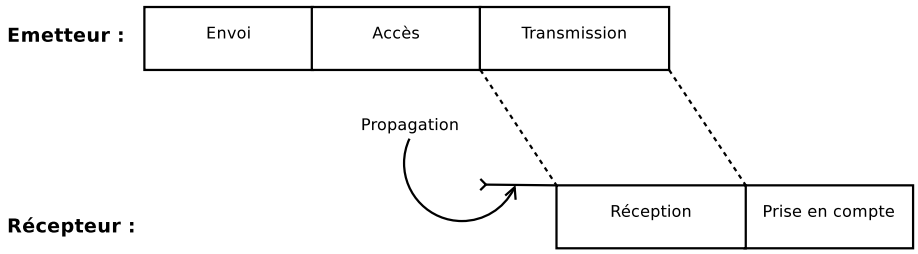


FIG. 3.7 – Décomposition du temps nécessaire à l'envoi d'un paquet (extrait et traduit de [66])

Certains de ces paramètres sont déterministes comme le temps de propagation ou les temps d'encodage et de décodage, alors que d'autres sont indéterministes comme le temps de récupération des interruptions ou le temps d'accès au canal de communication. L'algorithme décrit dans la section 3.4.1 permet de réduire le nombre de paramètres à prendre en compte grâce à l'utilisation d'un étiquetage temporel réalisé au niveau de la couche MAC. On enregistre la valeur de

l'horloge au moment du passage par la couche MAC ce qui permet de ne pas devoir considérer la durée des traitements effectués par les couches supérieures lors du calcul de la valeur de l'horloge. L'algorithme FTSP [66] permet aussi d'estimer le temps nécessaire à la chaîne complète de transmission d'un message entre deux nœuds et ainsi d'obtenir une synchronisation avec une précision de l'ordre de la microseconde.

L'algorithme *Flooding Time Synchronization Protocol* (FTSP)

Dans nos travaux nous exploitons l'algorithme FTSP [66, 68] (*Flooding Time Synchronization Protocol*) qui permet de synchroniser tous les nœuds d'une même composante connexe d'un réseau de capteurs sur l'horloge interne d'un de ses membres. L'objectif de cet algorithme est d'obtenir une synchronisation d'horloge avec une précision de l'ordre de la microseconde, qui supporte le passage à l'échelle sur un réseau de plusieurs centaines de nœuds, et qui soit robuste aux changements topologiques et aux pannes. FTSP est basé sur l'utilisation de marqueurs temporels (un champ dédié du message de synchronisation) dans lequel on stocke la valeur de l'horloge locale au moment du traitement des messages par la couche MAC.

Dans FTSP chaque nœud du réseau se synchronise sur une valeur d'horloge globale égale à la valeur de l'horloge interne d'un nœud de référence. Le nœud de référence sur lequel se fait la synchronisation est appelé racine. L'élection de la racine est basée sur l'hypothèse que chaque membre du réseau possède un identifiant unique (sous forme d'un entier). L'élection de la racine élit le nœud ayant le plus petit identifiant.

Au départ, chaque nœud se considère comme une racine de synchronisation. Il envoie périodiquement des paquets contenant son horloge locale et l'horloge qu'il considère globale, initialement son horloge locale. Ces messages sont de la forme présentée figure 3.8. Lorsqu'un nœud reçoit un message de synchronisation, dont l'horloge de référence (Global time) provient d'un nœud qui a un plus petit identifiant (Root ID) que celui sur lequel il est synchronisé, alors il considère ce nœud comme sa nouvelle racine et synchronise son horloge en fonction du message reçu.

Lors de la réception du message, le nœud enregistre sa propre horloge interne au moment du traitement du paquet au niveau de la couche MAC pour éviter au maximum les incertitudes introduites par les couches supérieures. Il peut alors calculer, au temps de propagation près et en fonction d'une estimation du temps de transmission, le décalage de son horloge interne avec celle de l'émetteur et mettre à jour son horloge globale.

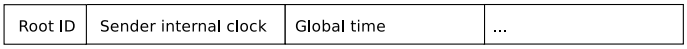


FIG. 3.8 – Contenu d'un paquet de synchronisation de l'algorithme FTSP

La figure 3.9 présente un scénario de synchronisation utilisant l'algorithme FTSP dans lequel le nœud élu est le nœud portant l'identifiant 1 et où tous les nœuds se synchronisent sur cette racine unique. Lors de la première étape (fig 3.9(a)), les nœuds 1 et 3 envoient un message de

synchronisation reçu respectivement par les nœuds 2 et 5. Ceux-ci se synchronisent donc sur les racines 1 et 3. Ensuite (fig 3.9(b)), le nœud 2 synchronisé sur la racine 1 envoie un message qui est reçu par les nœuds 4 et 5 qui prennent donc 1 comme racine. De proche en proche tous les nœuds sont finalement synchronisés sur la racine 1 à la fin du scénario (3.9(d)).

Les messages de synchronisation sont émis régulièrement du fait que comme indiqué précédemment les horloges dévient. Lorsqu'une racine disparaît du réseau, même de façon momentanée, chaque nœud qui ne reçoit plus de message de synchronisation devient lui-même racine et recommence à émettre des messages de synchronisation sans modifier son horloge globale. Le processus de détermination de la racine recommence alors et de la même façon le nœud avec le plus petit identifiant dans la composante connexe sera élu.

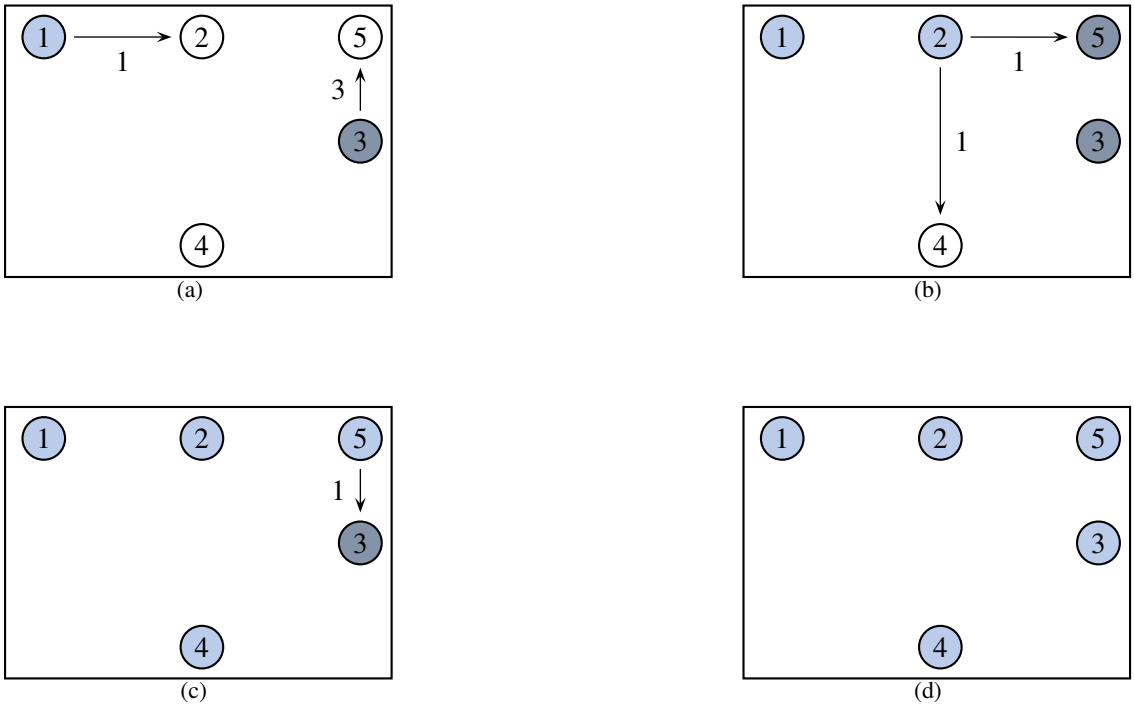


FIG. 3.9 – Exemple de déroulement de l'algorithme FTSP

3.4.2 Localisation

La deuxième information que l'on associe généralement aux valeurs relevées par un réseau de capteurs est la position des capteurs. Cette donnée est souvent cruciale pour l'exploitation des mesures car celles-ci peuvent perdre leur sens en l'absence de référence géographique.

Il existe au moins trois solutions permettant de réaliser ceci :

1. Équiper chaque capteur d'un système de géolocalisation ;

2. Associer manuellement à chaque capteur une position géographique connue avant le déploiement ;
3. Utiliser un algorithme de positionnement basé sur le fait que certains capteurs connaissent leur position par exemple grâce à un GPS embarqué ou via la connaissance de la position de certains autres capteurs, lesquels seront appelés des ancres.

Ce troisième point est le plus intéressant pour notre étude car il permet de positionner les nœuds de façon automatique sans pour autant tous les équiper de systèmes de localisation relativement coûteux. Nous retenons trois solutions parmi les plus répandues pour positionner les capteurs les uns par rapports aux autres :

1. *RSSI* [69](Received Signal Strength Indication)
   La méthode RSSI est une méthode assez simple pour calculer la distance entre deux nœuds dans un réseau de capteurs. Les nœuds envoient des suites de messages en utilisant différentes puissances d'émission et en incluant la valeur de celles-ci dans les paquets. Cela permet au récepteur de connaître la valeur minimale de la puissance d'émission à partir de laquelle il reçoit le signal et ainsi d'estimer assez précisément la distance qui le sépare de l'émetteur.
2. *TDOA* [70] (Time Difference Of Arrival) :
   Cette technique consiste à mesurer la différence de temps que met un paquet à arriver en plusieurs points de positions connues. Cette méthode peut être efficace dès lors que l'onde radio met un temps non négligeable à atteindre les nœuds ciblés. En effet, afin d'évaluer les distances et de mesurer le temps de trajet d'un paquet il est nécessaire que les nœuds soient synchronisés avec une précision supérieure à l'ordre de grandeur du temps de propagation. Alors que notre système de synchronisation a une précision maximale de l'ordre de la microseconde, il est admis que les temps de propagation sont généralement de l'ordre de la centaine de nanosecondes, il est donc impossible d'utiliser cette méthode dans notre contexte. Une méthode ne nécessitant pas de synchronisation d'horloge a été développée et est utilisée dans le système CRICKET [71] du MIT. CRICKET repose sur l'utilisation d'un émetteur à ultrasons en plus du périphérique radio de communication. Le principe consiste à envoyer un message radio à un nœud identifié et à ce qu'il réponde en émettant un signal ultrason. Les ultrasons étant beaucoup plus lents (env. 300 m/s) que les ondes électromagnétiques (env. 300 000 km/s), il devient plus aisé d'estimer avec précision la distance entre les entités. Nous avons écarté cette méthode car nous ne disposons pas de nœuds équipés ni des compétences nécessaires pour adapter notre matériel.
3. *AOA* [72] (Angle Of Arrival) :
   Le procédé AOA consiste à mesurer l'angle d'arrivée des paquets sur un nœud à l'aide d'une antenne complexe qui permet d'associer un signal reçu à une direction. Cette technique permet donc de positionner précisément un nœud par rapport à un autre nœud en l'associant

à une méthode permettant de calculer l'éloignement entre le récepteur et l'émetteur. De plus, en connaissant la position de l'émetteur, le nœud peut être localisé de façon absolue en n'ayant pour information la position d'un seul de ses voisins. La connaissance de la position absolue de l'émetteur peut aussi permettre de se localiser précisément de manière autonome lorsque l'on connaît la position de deux ou trois de ses voisins suivant que l'on possède des informations sur l'orientation ou pas. Les antennes permettant ce type de localisation n'étant pas présentes sur notre matériel (Crossbow Mica2), cette solution a été écartée malgré ses bonnes performances.

Utilisation d'ancres

Connaître la distance entre les différents capteurs ne suffit pas à les positionner de façon absolue. En revanche l'utilisation d'ancres, qui sont des nœuds connaissant leur propre position, est une des techniques les plus courantes permettant de le faire. Un nœud ne connaissant pas sa position mais les distances le séparant d'au moins trois ancres peut ainsi calculer ses coordonnées absolues. Par exemple, les nœuds équipés d'un système de géolocalisation peuvent constituer des ancres. On peut aussi placer certains nœuds sur des points de coordonnées définies à l'avance (et ce sont donc des ancres).

Une solution, MoteTrack [14], développée à Harvard s'appuie sur la méthode RSSI décrite précédemment et l'utilisation d'ancres. Cette solution a notamment la particularité d'utiliser comme ancres les nœuds qui ont calculé leur position. L'utilisation de RSSI est la seule qui soit adaptée à notre matériel et MoteTrack semble être pour le moment la mise en œuvre la plus appropriée même si elle nécessite de nombreuses ancres au départ. Une solution basée sur AOA permettrait, grâce à un meilleur positionnement relatif, de réduire fortement le nombre d'ancres à introduire dans le réseau et serait plus adaptée à un contexte dynamique.

Chapitre 4

Construction d'une vision continue des données collectées

4.1 Description de l'application et spécificité du contexte militaire

Il pourrait dans un premier temps sembler que les réseaux de capteurs ne soient pas concernés par la thématique de ce document qui est l'étude des applications des réseaux mobiles ad hoc. En effet, si la mise en œuvre d'un tel réseau est sans conteste ad hoc, son évolution n'est pas mobile au sens premier. Pourtant si l'on considère le graphe de connectivité du réseau, il peut être caractérisé de la même façon que pour un réseau MANet à savoir par l'apparition ou la disparition de sommets ou d'arêtes à tout moment.

Quand on utilise ce type de matériel, même dans le cadre d'un laboratoire, il apparaît très rapidement que sa fiabilité ne permet pas de considérer un réseau de capteurs comme un réseau statique, du fait des pannes et des problèmes de connexion. En outre, lorsque le matériel est situé dans un environnement hostile comme un terrain ennemi, un champ de mines ou une forêt en feu, les phénomènes de disparition des nœuds ou d'interférences sont inévitables.

Dans un contexte réel, l'objectif du déploiement d'un réseau de capteurs est de collecter des données en différents points d'un champ d'opérations et de les faire parvenir à des postes clients capables de les traiter, bien souvent dans un but d'aide à la décision (feux de forêt). Pourtant lorsque le réseau subit certaines avaries des messages se perdent et la vision du phénomène observée devient discontinue.

Nous définissons dans la suite une architecture permettant de construire une vision continue d'un phénomène lorsque la collecte des données sous-jacentes devient discontinue dans le temps ou dans l'espace. En effet, la suite de relevés effectués peut être discontinue dans le temps car certaines mesures provenant d'un capteur peuvent disparaître à cause de problèmes de transmission. C'est pourquoi, nous introduisons une architecture modulaire permettant d'utiliser des méthodes d'interpolation contextuelles afin de produire une estimation des données manquantes.

4.2 Conception et problèmes algorithmiques

Nous considérons des réseaux de capteurs qui peuvent être constitués de plusieurs composantes connexes disjointes. Les données capturées doivent pouvoir être repositionnées dans le temps de manière unique, c'est-à-dire associées à leur date de relevé. Ceci doit pouvoir être réalisé même si le réseau est partitionné en plusieurs composantes connexes. Nous nous plaçons dans un réseau de capteurs dans lequel certains nœuds sont équipés d'un récepteur GPS qui leur fournit donc une horloge de référence. La valeur de cette horloge est associée aux mesures et peut être interprétée par tout utilisateur externe au système sans qu'il soit nécessairement synchronisé sur cette horloge.

Nous supposons que la densité de nœuds équipés d'un système de géolocalisation est à tout instant au moins de un par composante connexe. De cette façon, chaque composante connexe conserve une référence à l'horloge globale du système de localisation. Nous proposons alors une solution de synchronisation basée sur l'algorithme FTSP [66] présenté précédemment (section 3.4.1 page 36) et qui a été modifié pour profiter de l'apport d'une horloge provenant d'un système de géolocalisation. Les nœuds équipés de GPS sont choisis comme racines de référence, ce qui supprime la phase d'élection. De ce fait on peut avoir plusieurs racines pour la synchronisation dans une même composante connexe. Les nœuds sans système de géolocalisation devront donc choisir une de ces racines.

Les nœuds qui ne sont pas équipés de capteurs GPS se synchronisent de proche en proche sur l'horloge globale fournie par ceux disposant d'un GPS. L'erreur introduite à chaque saut par l'algortihme FTSP, de l'ordre de la microseconde, est ici d'autant plus négligeable que l'horloge fournit par le GPS a une précision à la milliseconde (suffisante pour les objectifs fixés). Nous ajoutons une étiquette dans les paquets de synchronisation pour indiquer le nombre de sauts qui séparent l'émetteur d'un de ces paquets de la racine qui lui sert de référence. Un nœud équipé d'un GPS étiquette donc les paquets de synchronisation qu'il émet avec un entier de valeur 0. Cette étiquette est incrémentée à chaque saut. Toute entité souhaitant se synchroniser choisit parmi ses voisins celui qui est le plus proche d'une racine, car plus un nœud est proche d'une racine moins le décalage induit par le protocole de synchronisation sera grand.

L'algorithme 1 décrit de manière simplifiée la méthode utilisée par chaque nœud pour choisir sa racine et transmettre l'horloge globale de proche en proche. Il utilise des fonctions permettant de faciliter la lecture :

- reception_message() : cette fonction est bloquante et en attente d'un paquet de synchronisation,
- est_synchronise() : cette fonction permet de savoir si le nœud s'est déjà synchronisé sur un horloge,
- synchronise_horloge_sur() : cette fonction permet de fixer l'horloge du nœud courant en fonction de celle contenu dans le paquet de synchronisation

Algorithme 1 Algorithme de synchronisation d'horloge (version simplifiée)

```
while vrai do
  if ( ! Noeud_equipe_GPS && (paquet = reception_message())) then
    // attend jusqu'à réception d'un message
    if ( ! est_synchronise() || (etiquette(paquet) + 1 < ETIQUETTE)) then
      synchronise_horloge_sur(paquet) ;
      ETIQUETTE = etiquette(paquet) + 1 ; //nobre de sauts jusqu'à la racine
    end if
  end if
  envoyer_message_synchronisation(horloge, etiquette) ;
end while
```

– envoyer_message_synchronisation() : cette fonction permet de répandre de proche en proche la nouvelle valeur de l'horloge.

La figure 4.1 illustre un scénario de déroulement de la version modifiée de FTSP qui utilise un système de géolocalisation (ici un GPS). Le réseau de capteurs est composé de cinq nœuds dont deux sont équipés de GPS. A la première étape (fig 4.1(a)) les nœuds équipés de GPS envoient des messages de synchronisation. Lorsque le nœud 2 est synchronisé (fig 4.1(b)), il participe aussi au processus de synchronisation en envoyant lui-même des messages de synchronisation. De proche en proche, les capteurs synchronisent leur horloge sur celle du capteur équipé d'un GPS le plus proche en terme de nombre sauts (fig 4.1(d)).

4.3 Définition et algorithmique du système proposé

Lors de la prise de mesures, à intervalles réguliers, au sein d'un réseau de capteurs, certaines mesures peuvent disparaître et les clients subissent les conséquences de ce manque d'information lors de l'interprétation du phénomène observé. Ces pertes peuvent par exemple être la conséquence d'interférences radio. Certaines mesures peuvent aussi manquer à cause de la défaillance matérielle d'un capteur qui n'aura pas relevé certains échantillons. Afin de limiter les effets négatifs de ce phénomène nous avons proposé et mis en place une architecture générique permettant d'interpoler autant que possible les données manquantes. Ce mécanisme a été testé avec comme grandeur mesurée la variation de température à l'intérieur d'une pièce lors de la création d'un courant d'air (cf. section 4.4).

4.3.1 Organisation du réseau

On peut distinguer deux types de nœuds au sein du réseau de capteurs comme présenté figure 4.2 :

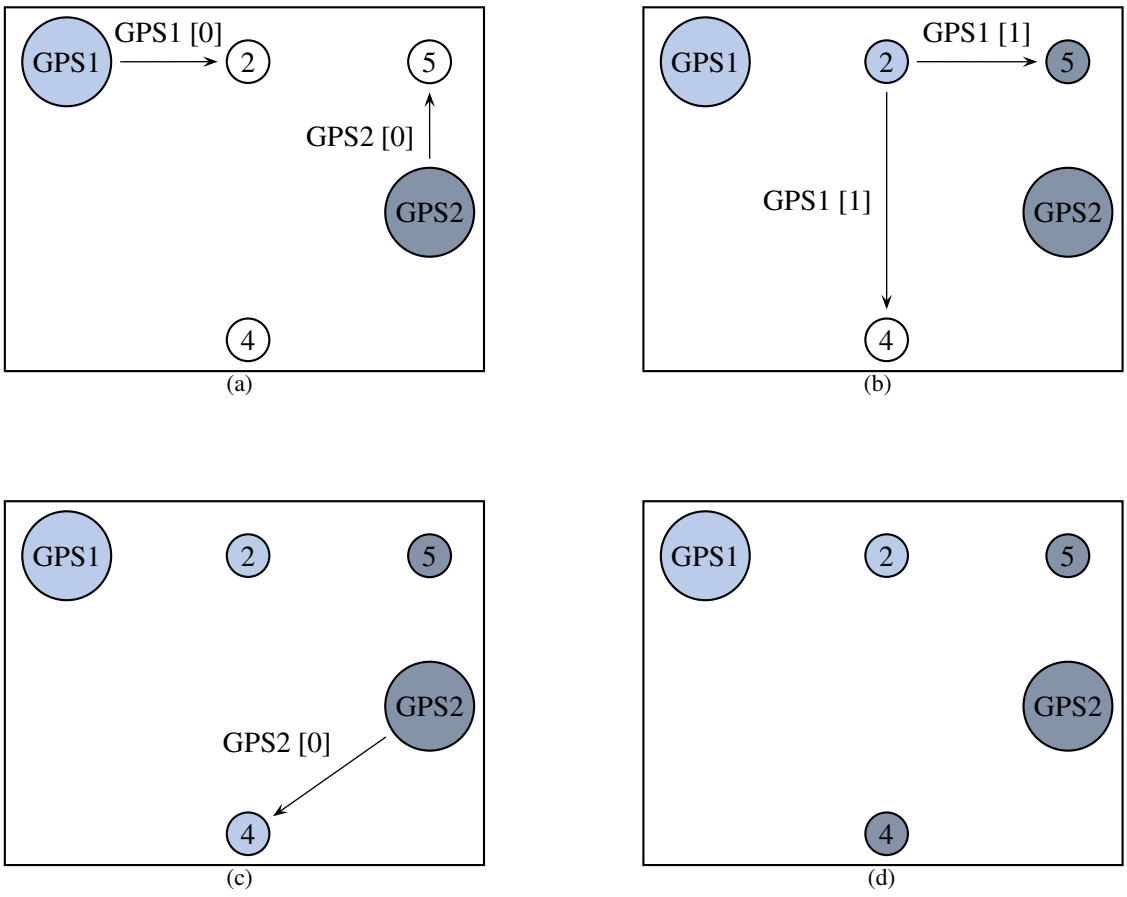


FIG. 4.1 – Exemple de déroulement de l'algorithme FTSP modifié

1. Les nœuds dits « basiques » ont pour rôle de prendre des mesures et de les disséminer dans le réseau. Il s'agit en pratique d'un Mica2 et d'une carte de capteurs.
2. Les nœuds dits « clients » collectent les données qui transitent sur le réseau dans le but de les utiliser localement. Les clients sont composés d'une machine classique (portable ou PDA) connectée via une interface (série ou USB) à un nœud (Mica2). Ce nœud sert d'interface à la machine pour qu'elle puisse communiquer avec le reste du réseau de capteurs en utilisant le protocole adapté. Afin que le nœud installé sur la station de base transmette les paquets qu'il reçoit, nous y installons une application fournie par Crossbow (fabricant des Mica2) qui a pour rôle de répéter les paquets reçus du réseau de capteurs à la fois sur le canal radio et sur un port série du microcontrôleur (relié à la station de travail).

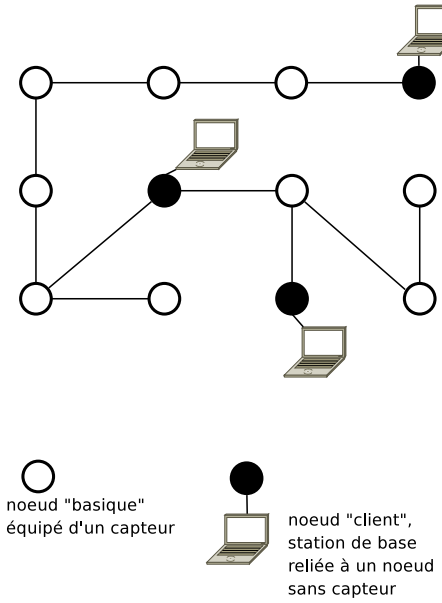


FIG. 4.2 – Organisation des éléments du réseau

4.3.2 Capture, dissémination et perte éventuelle des mesures

Le principe de l'architecture proposée est de fournir une vision globale d'un phénomène physique, mesuré par un réseau de capteurs, en palliant les pertes éventuelles de certains échantillons. Pour supporter un maximum de dynamicité au sein de la topologie, nous ne souhaitons pas nous adresser à un nœud client en particulier ni utiliser une couche de routage. De plus, la fiabilité relativement faible (liée au contexte d'utilisation) des éléments d'un réseau de capteurs ne permet pas de supposer qu'un nœud qui est disponible à un instant donné le sera à l'instant suivant.

L'application installée sur les nœuds « basiques » se compose d'une partie destinée à la prise de mesures et d'une autre partie qui fait suivre les mesures d'un nœud à l'autre. L'algorithme de

capture des données repose sur le fait qu'un nœud basique ne commence ses captures qu'une fois qu'il est synchronisé sur l'horloge de référence (cf. algorithme 2).

Algorithme 2 Capture et envoi périodique de la température par un capteur (version simplifiée)

```
loop
  if est_synchronise() then
    t ← relever_temperature()
    diffuser_temperature(t)
  end if
  attendre(intervalle)
end loop
```

Il nous faut aussi considérer d'autres contraintes inhérentes aux réseaux de capteurs comme la consommation d'énergie ou les interférences radio. Nous nous devons donc de limiter l'émission de paquets de données inutiles. Pour permettre une meilleure configuration nous ajoutons dans notre algorithme de diffusion une variable TTL (*Time To Live*) aux paquets de dissémination de données. D'autre part, chaque nœud ne transmet qu'une seule fois chaque paquet (cf algorithme 3) et enregistre l'identifiant des paquets transmis (dans la limite de la mémoire disponible sur le nœud pour stocker ces identifiants). Cela évite de ré-émettre indéfiniment les paquets et de ce fait limite les interférences qui peuvent être importantes dans un réseau dense. Ce paramètre (TTL) peut être modifié par l'utilisateur en fonction de la connectivité du réseau, de sa taille et du contexte d'utilisation. Nous présentons l'algorithme 3 utilisé dans nos travaux par souci de transparence, celui-ci n'est pas optimal en terme de consommation énergétique. La contribution de cette partie ne porte pas sur cet algorihtme et il pourrait être remplacer par tout algorithme de diffusion.

Bien que notre système de dissémination duplique les paquets sur un maximum d'arêtes du réseau, des messages peuvent tout de même ne pas arriver au nœud client notamment lorsqu'on se place dans un contexte d'utilisation militaire ou hostile. En effet, plusieurs raisons peuvent entraver la bonne dissémination des informations :

- Les communications peuvent subir des interférences provenant soit du réseau lui-même, soit d'autres équipements, par exemple placés dans ce but par un ennemi.
- Un nœud peut venir à manquer d'énergie.
- Un nœud peut être physiquement endommagé ou détruit.

Afin de limiter les conséquences que peut engendrer une perte d'information (comme une mauvaise estimation du phénomène observé), nous plaçons sur les clients une infrastructure que nous avons développée et qui permet d'interpoler des données manquantes.

4.3.3 Architecture du système d'interpolation

Nous avons conçu et développé une architecture permettant d'interpoler les données soit au cours d'une expérience soit lors d'un traitement postérieur à la collecte. Le système repose sur la

Algorithme 3 Dissémination des mesures reçues

```
loop
  message ← message_suivant_dans_la_file_de_reception)
  TTL ← ttl(message);
  if jamais_retransmis(message) then
    if TTL > 0 or TTL = −1 then
      // TTL==-1 l'option TTL n'est pas utilisée
      if TTL ≠ −1 then
        modifier_ttl(message, TTL − −)
      end if
      retransmettre(message)
      ajouter_liste_message_retransmis(message)
    end if
  end if
end loop
```

mise en cache des données collectées afin de les utiliser pour le calcul des valeurs manquantes (interpolation) ou pour l'estimation de la tendance à venir (extrapolation).

La mise en œuvre de cette architecture a été effectuée sur la base de composants pour pouvoir changer facilement de méthode d'interpolation ou d'extrapolation. De cette façon, l'expert dans le domaine d'application ciblé n'a plus qu'à définir une méthode adaptée au contexte applicatif.

L'architecture modulaire du système peut être modélisée comme cela est présenté figure 4.3. L'architecture est composée de cinq couches :

1. Le nœud connecté à la station de base (*Input mote*) a pour rôle de transmettre à la station de base les données qui transitent au sein du réseau de capteurs. En pratique, nous avons installé sur un nœud l'application *TOSBase* fournie avec *TinyOS*. Elle récupère les paquets sur l'interface radio et les retransmet à la station via le port série du microcontrôleur.

2. On place ensuite sur la station de base un module (*Data Server*) qui récupère les données provenant du nœud. Ce module permet d'extraire l'information utile des paquets qui transitent sur le réseau de capteurs.

3. Les données sont ensuite mises en cache (*Buffer*). Elles peuvent aussi être triées en fonction de l'identifiant du capteur qui a effectué les mesures et/ou de la date à laquelle elles ont été collectées (*Ordered Buffer*). Ceci est utile pour certains algorithmes d'interpolation ou d'extrapolation qui nécessitent que les données soient ordonnées.

4. Les échantillons de mesures peuvent ensuite être fournis au module d'interpolation (*Interpolation Module*) ou d'extrapolation (*Extrapolation Module*) qui complète les mesures manquantes.

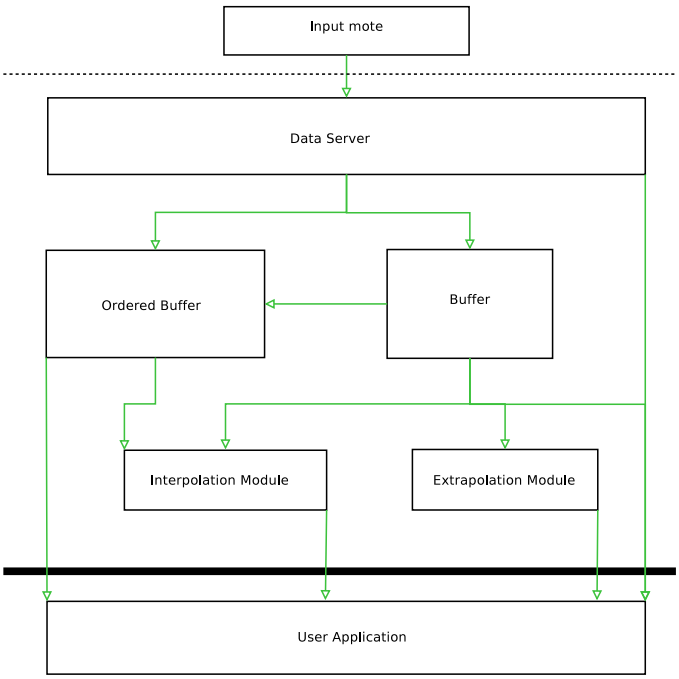


FIG. 4.3 – Modélisation de l'architecture

5. Enfin, les données peuvent être utilisées par l'application finale. A ce niveau les valeurs interpolées ou extrapolées sont différenciées des données réelles à l'aide d'un marqueur (en pratique un bit).

L'organisation de ces composants entre eux s'adapte à l'application finale. Les communications entre ceux-ci sont basées sur un système d'événements qui peuvent être récupérés par plusieurs composants, qu'ils appartiennent à la même couche de l'architecture ou non.

4.4 Résultats de l'évaluation de l'architecture d'interpolation

4.4.1 Capture des mesures

Afin d'évaluer le fonctionnement de notre système, nous avons placé vingt nœuds Mica2 répartis de manière homogène dans une pièce de douze mètres par six et qui comporte deux fenêtres (W1 et W2) comme indiqué sur la figure 4.4. Le but de l'expérience est de tester l'architecture que nous avons proposée en mesurant la variation de la température de la pièce. Durant cette expérimentation nous procédons à l'ouverture simultanée des deux fenêtres qui sont situées à des positions opposées dans la pièce. La prise de mesure a consisté à enregistrer la température en chacun des points où sont placés les capteurs sur une période de temps pendant laquelle les fenêtres W1 et W2 sont ouvertes. Nous observons la distribution de la température au début de l'expérience sur la figure 4.5 et à la fin de l'expérience figure 4.6 (sept minutes plus tard). Les axes représentent

les dimensions de la salle en décimètres.

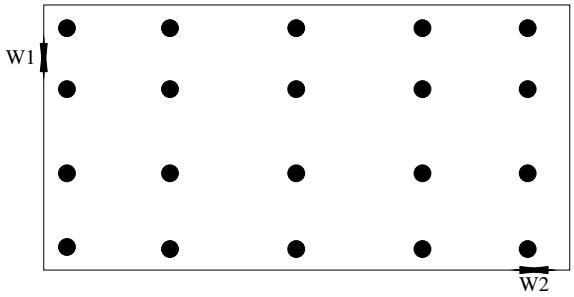


FIG. 4.4 – Plan de la salle d'expérimentation

Au début de l'expérience, on observe une température moyenne d'environ 23°C et un point plus chaud près de la fenêtre de l'angle inférieur droit (*W2*) dû à l'ensoleillement. Après ouverture des fenêtres on observe une diminution globale de la température induite par le courant d'air frais créé entre les deux fenêtres.

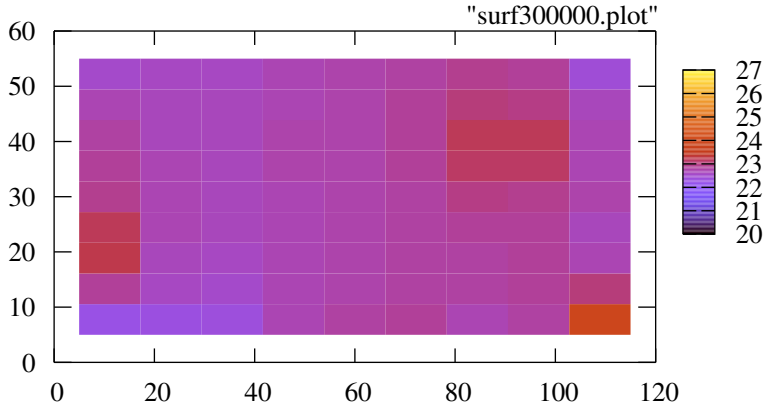


FIG. 4.5 – Température observée dans la salle au début de l'expérience

D'un point de vue global on observe que la température de la pièce est correctement relevée et que sa variation est conforme à ce que l'on pouvait attendre. Le réseau de capteurs nous permet donc de prendre des mesures et le poste client peut les récupérer et les exploiter.

4.4.2 Interpolation des mesures

Après avoir réussi à mesurer un phénomène grâce à un réseau de capteurs nous souhaitons évaluer la pertinence de notre architecture d'interpolation. Le test présenté ci-après se focalise

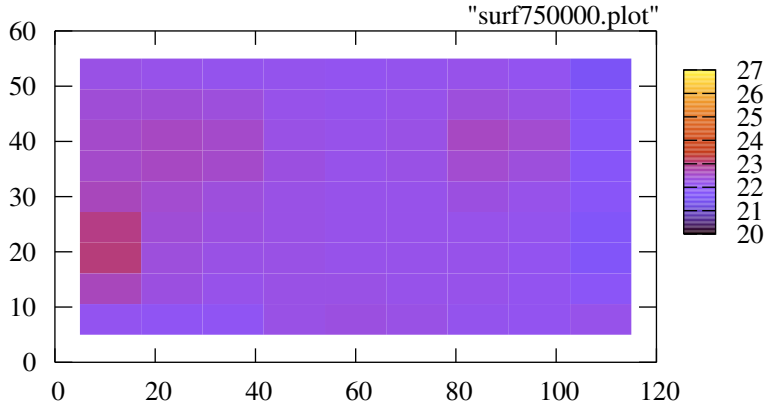


FIG. 4.6 – Température observée dans la salle après ouverture des fenêtres

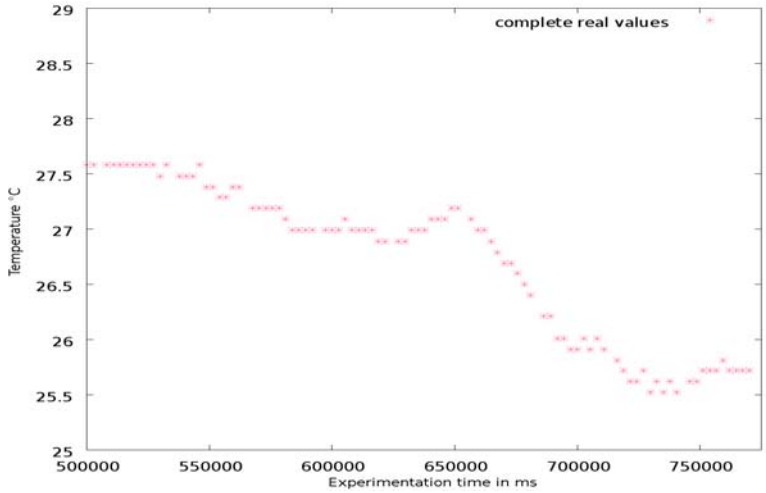
sur les données relevées en un seul point du réseau de capteurs. Pour évaluer l'intérêt ou l'efficacité du système d'interpolation proposé nous avons choisi un point sur lequel il n'y a pas eu de pertes de valeurs au cours de l'expérience. Ensuite, deux tiers des valeurs ont été supprimées de façon homogène. Nous avons alors tenté d'interpoler des valeurs « manquantes » en utilisant une implémentation de l'architecture décrite précédemment.

Les graphiques de la figure 4.7 représentent l'évolution de la température en un point donné au cours du scénario décrit dans la section précédente. Ils sont constitués de mesures prises à intervalles de trois secondes pendant la phase de changement de la température dans la pièce. Le graphique 4.7(a) représente les valeurs réelles collectées par le nœud étudié. Celui-ci est complet dans le sens où il ne manque aucun échantillon. Nous avons ensuite supprimé volontairement des mesures puis utilisé un algorithme d'interpolation polynomial de type *Cubic Spline*[15] afin de reconstituer les données supprimées. Cette méthode consiste à construire un polynôme, à partir des points dont on dipose, dont les dérivées première et seconde sont continues sur chaque intervalle entre deux mesures (afin d'avoir une courbe d'interpollation très progressive) puis à appliquer cette fonction pour les points que l'on considère manquants pour procéder à une interpolation. L'objectif de ce travail n'est pas la mise au point d'une méthode de calcul des mesures manquantes. En effet, la meilleure technique d'interpolation dépend de la nature du phénomène observé et de son évolution dans le temps. De ce fait la définition de méthodes de calcul relève de la compétence d'experts. Les résultats présentés dans ce document montrent l'utilisation d'une architecture générique dont l'intérêt n'est pas lié au phénomène physique ou à la stratégie d'interpolation.

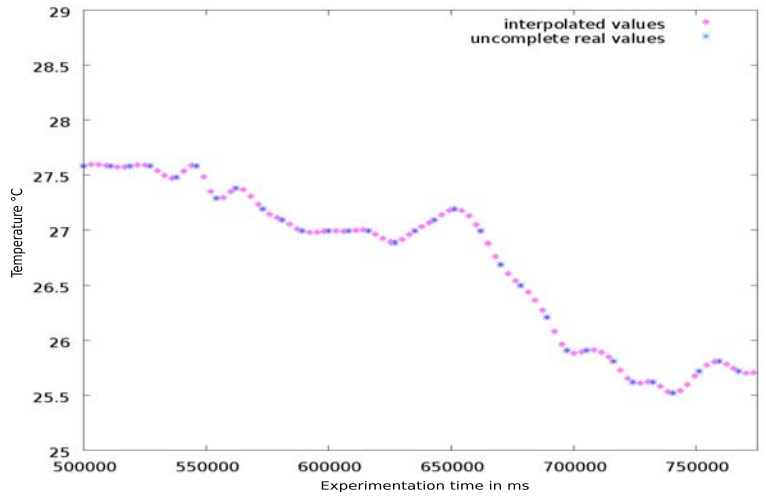
Le graphique 4.7(b) montre les valeurs réelles et les valeurs interpolées. La figure 4.7(c) permet de comparer les résultats de l'interpolation avec les données réelles. Nous observons que seules quelques mesures réelles s'écartent de la courbe reconstruite. Ceci est non significatif en regard

de la précision de nos capteurs qui est de l'ordre de 0,1 °C. Il est évident que la définition de l'algorithme le plus adapté est très dépendante du contexte et peut nécessiter plus de données que les seules informations locales à un nœud. Ceci sera discuté dans la conclusion de cette partie.

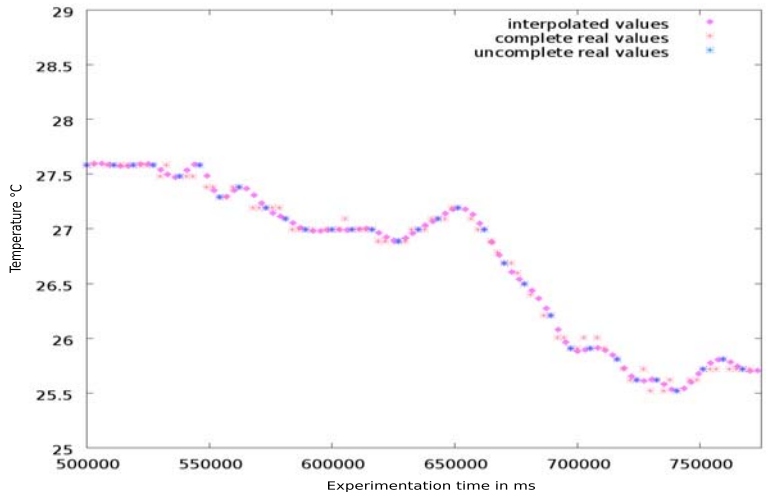
Ces résultats montrent que notre architecture est fonctionnelle et que l'utilisation d'algorithmes d'interpolation est pertinente lorsque certaines valeurs manquent pour procéder à l'analyse d'un phénomène.



(a) Valeurs réelles complètes



(b) Valeurs réelles tronquées et valeurs interpolées



(c) Comparaison entre les valeurs réelles complètes et les valeurs interpolées

FIG. 4.7 – Courbes représentant les résultats des relevés et de l'interpolation

Conclusion

Résumé des contributions

Le travail présenté dans cette partie du document permet de fournir une vision aussi complète que possible d'un phénomène dont le contexte d'observation ne permet pas de récupérer toutes les données élémentaires. Rappelons, en effet, que lors du déploiement du matériel et de la collecte d'information sur un champ d'opération, le système de mesure peut subir des avaries comme des interférences ou des pannes matérielles. Celles-ci peuvent être la cause de la perte de données cruciales dans les prises de décision. Nous avons donc proposé une architecture permettant d'interpoler ou d'extrapoler les données manquantes en fonction de celles qui ont pu être collectées et acheminées correctement.

Ces travaux ont fait l'objet de plusieurs communications sous différentes formes lors de conférences. L'architecture et l'algorithmique ont été publiés lors de la conférence IEEE MILCOM 2007 (*MILtary COMmunication Conference*) [73]. L'expertise acquise tant dans le domaine des réseaux de capteurs, des RFID que de ZigBee (hors du contexte de cette thèse pour ce dernier point) a donné lieu à un tutoriel d'initiation à ces technologies donné lors du workshop WISTP 2008 [74] (*Workshop in Information Security Theory and Practices*).

Enfin, l'implémentation présentée ici est fonctionnelle. Elle offre la possibilité d'expérimenter différentes méthodes d'interpolation ou d'extrapolation car elle permet une intégration rapide de celles-ci au sein de l'architecture. Une application concrète peut ainsi être rapidement mise en œuvre par des utilisateurs, militaires ou civils, qui souhaiteraient mesurer un phénomène spécifique.

Perspectives

Nous avons illustré ce travail et l'utilisation de notre architecture grâce à la méthode d'interpolation du *Cubic Spline*. Cet algorithme générique se base sur la continuité du phénomène pour interpoler des valeurs. Nous pouvons envisager de comparer ses performances avec d'autres algorithmes d'interpolation pour déterminer lequel est le plus adapté à ce type de phénomène. Il serait aussi intéressant d'étudier d'autres phénomènes physiques (par exemple le bruit ambiant)

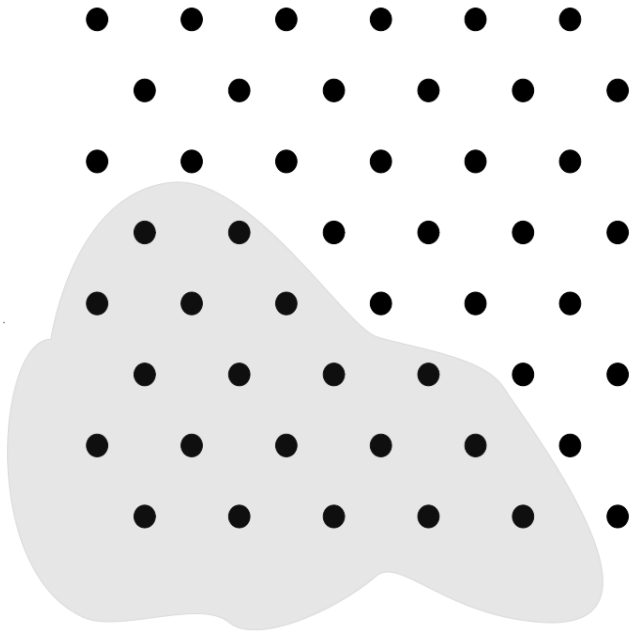


FIG. 4.8 – Phénomène continu dans l'espace (nuage) sur un champ d'opération muni d'un réseau de capteurs

pour évaluer l'efficacité de cet algorithme ou d'autres algorithmes génériques dans des contextes différents.

Enfin, il parait envisageable de reconstituer des données manquantes, même pour des phénomènes peu linéaires en utilisant les échantillons collectés au même instant en d'autres points proches du point concerné. Une telle approche peut permettre d'interpoler des valeurs pour un phénomène qui n'est pas continu dans le temps mais dans l'espace. Par exemple, lors du passage d'un nuage radioactif (fig 4.8), phénomène comportant des variations très rapides, on pourrait vouloir mesurer la radioactivité sur le champ d'opération.

Deuxième partie

Seconde problématique : Travail collaboratif et information dynamique

Table des matières

Introduction au problème **59**

5 Structure et modélisation de la solution proposée **61**
5.1 Structure de données utilisée 61
5.2 Gestion de la cohérence du document 65
5.2.1 États des nœuds de l'arbre 65
5.2.2 Transitions entre les différents états des nœuds 67
5.3 Versions et synchronisation entre utilisateurs 69
5.3.1 Gestions des versions 70
5.3.2 Synchronisation 71
5.4 Conclusion 72

6 Perte de droits, information floue et logique associée **75**
6.1 Problème 75
6.2 Le contexte hiérarchique militaire 76
6.3 Présentation de la solution pour récupérer des régions 76
6.4 Utilisation des cartes à puces et authentification de la hiérarchie 77
6.4.1 Introduction aux cartes à puces 77
6.4.2 Intérêt de l'utilisation des cartes à puces pour la réintroduction de verrous 78
6.5 Introduction de la hiérarchie 79
6.5.1 Prise en compte de la réintroduction de verrous au niveau de la structure des informations 79
6.5.2 Influence de l'introduction de nouveaux verrous sur la fusion de sous-parties 81
6.5.3 Algorithme de régénération de verrous 82
6.6 Information sur une surface 85
6.7 Logique associée aux changements d'états 86
6.7.1 Modification locale de l'état d'une information par constats 86
6.7.2 Transitions simples et informations exclusives 87
6.7.3 Synchronisation 87
6.8 Conclusion 89

7 Simulation **91**
7.1 Introduction 91
7.2 Le Simulateur Madhoc 92

7.3 Deux autres simulateurs existants . . . 93
7.3.1 ns-2 . . . 93
7.3.2 GloMoSim . . . 94
7.4 Les modèles de mobilité . . . 94
7.4.1 Le modèle de mobilité Random WayPoint (RWP) . . . 94
7.4.2 Le modèle de mobilité Human Mobility . . . 96
7.4.3 D'autres modèles de mobilité . . . 96
7.5 Codage de Shaadhoc dans Madhoc . . . 98
7.5.1 Adaptation de Shaadhoc à la discrétisation du temps . . . 98
7.5.2 Simulation du comportement des utilisateurs . . . 100
7.6 Résultats . . . 101
7.6.1 Validation qualitative . . . 101
7.6.2 Validation quantitative . . . 101
7.6.3 Mesures . . . 102
7.7 Conclusion . . . 112

8 Mise en œuvre de Shaadhoc 113
8.1 Architecture logicielle . . . 113
8.2 Découverte du voisinage et des services . . . 115
8.2.1 Découverte de voisinage dans les réseaux mobiles IP (WiFi) . . . 115
8.2.2 Bluetooth et SDP . . . 117
8.3 Protocole de synchronisation . . . 118
8.4 Interface et scénario d'utilisation . . . 119
8.4.1 Verrouillage/Déverrouillage . . . 121
8.4.2 Découpage/Fusion . . . 122
8.5 Conclusion . . . 122

9 Conclusion générale 125

Introduction au problème

La multiplication des technologies de communication sans fil et leur intégration dans de nombreux matériels permet le développement et la diffusion d'applications communautaires. Celles-ci ont généralement pour principe d'utiliser les moyens de calcul et de communication disponibles dans le réseau pour aboutir à un objectif commun.

Parmi ces objectifs on trouve notamment le partage de données qui est un des principaux thèmes de recherches dans le domaine des réseaux MANets [5]. Dans un contexte aussi instable, la diffusion d'informations est une tache complexe. Dans ce même contexte si l'on souhaite éditer un document partagé entre plusieurs utilisateurs la difficulté est encore plus grande à cause de l'évolution constante des données. Le problème majeur vient du fait que l'on ne peut pas faire de supposition sur la présence des utilisateurs qui participent à la tâche collaborative à un instant donné. De plus, il n'est pas possible d'assurer une vision globale du réseau car dans un contexte mobile on ne peut pas exclure le fait que celui-ci puisse se partitionner en différents îlots de façon temporaire ou permanente.

Nous présentons dans cette partie une méthode pour éditer un document de façon collaborative. Ce document peut être par exemple une carte stratégique partagée par des fantassins (cf. figure 4.9 page suivante). La grande difficulté est d'assurer la cohérence des données qui sont modifiées de façon distribuée. La solution que nous proposons se base sur le découpage du document en plusieurs parties. Ceci permet aux utilisateurs de modifier ces différentes parties en parallèle. Nous souhaitons aussi pouvoir fournir à chaque participant la meilleure vision globale possible du document modifié au fur et à mesure qu'il rencontre d'autres entités mobiles.

La méthode que nous proposons pour gérer la cohérence des données repose sur l'association d'un état qui peut être assimilé à un verrou à chaque sous-partie. Nous présentons chapitre 5 les différents états qui peuvent être associés à une partie ainsi que les transitions qui les lient. Afin de permettre de constituer des sous-parties du document global et de pouvoir le ré-assembler nous lui avons fourni une structuration hiérarchique. Celle-ci se présente sous la forme d'un arbre sur lequel nous définissons les opérations de découpage et de fusion de parties.

Au fur et à mesure que les utilisateurs se rencontrent, ils peuvent mettre à jour leurs versions du document et se transmettre les verrous sur certaines parties de la carte ; nous appelons cette opération synchronisation (de versions).

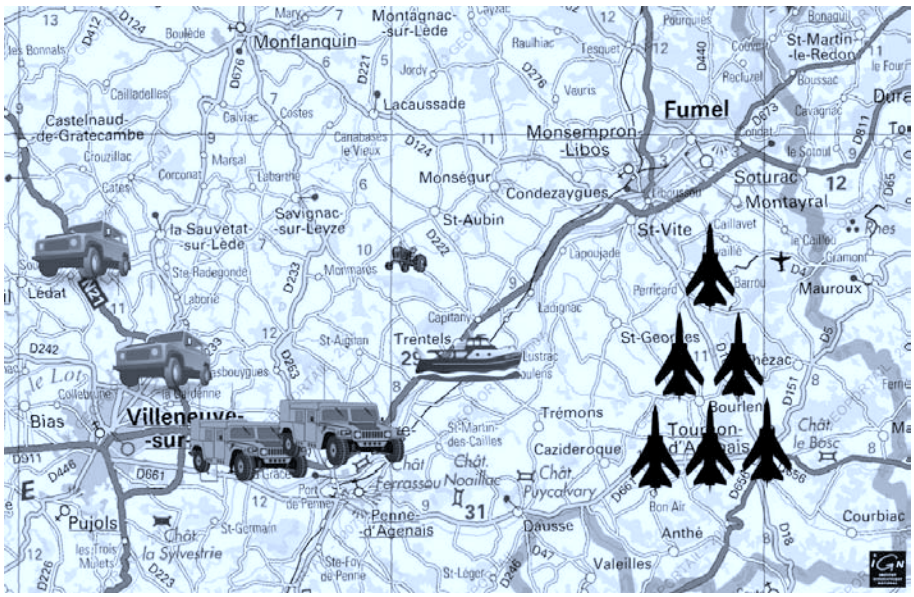


FIG. 4.9 – Exemple de carte stratégique

Un problème se pose cependant lorsqu'une entité qui dispose d'un verrou sur une partie de la carte vient à disparaître définitivement. Ce verrou est alors perdu et cette partie du document n'est donc plus modifiable. Néanmoins, le contexte d'application militaire permet ici de différencier certains nœuds qui disposent d'un pouvoir supplémentaire et donc de droits supplémentaires. La solution à la perte de verrous est la réintroduction par des supérieurs hiérarchiques de verrous sur des parties considérées comme perdues. Nous proposons aussi une méthode pour gérer la cohérence lorsqu'un morceau du document considéré comme perdu à tort réapparaît dans le système.

Ces travaux ont conduit au développement de l'application Shaadhoc que nous avons amenée à un stade pré-industriel grâce au financement d'un contrat d'ingénieur obtenu dans le cadre de la labellisation Carnot du LaBRI. Ce travail a aussi donné lieu à deux publications. La première présente une version préliminaire du système d'édition, plus statique que celle présentée dans ce document et qui utilise une notion de groupe d'entités mobiles (workshop MobiWac 2006 [75]). La version présentée dans ce document, mais sans gestion des parties potentiellement perdues, a fait l'objet d'une communication lors de la conférence MILCOM 2006[76].

Chapitre 5

Structure et modélisation de la solution proposée

L'objectif de notre système est de permettre l'édition collaborative d'un document tel qu'une carte stratégique. Chaque utilisateur dispose d'une copie locale du document dont il peut modifier certaines parties sur lesquelles il possède ce droit. La copie locale se propage de proche en proche sur les terminaux des utilisateurs souhaitant participer à la tâche collaborative. Pour cela nous mettons en place un mécanisme de découpage du document sous une forme arborescente. Nous définissons aussi la notion d'état pour chaque nœud de l'arbre afin de gérer les droits en écriture sur les différentes parties. Enfin les modifications apportées sur le document sont distribuées dans le réseau via un mécanisme de synchronisation qui repose sur l'utilisation de numéros de version.

5.1 Structure de données utilisée

La propriété majeure de notre méthode est d'assurer qu'à tout moment l'ensemble des copies locales du document présentes sur chacun des nœuds converge, au fur et à mesure des rencontres entre utilisateurs, vers une version unique et ce de façon déterministe. Cela signifie que si aucun utilisateur ne modifie sa copie locale du document, alors, au bout d'un certains temps grâce au processus de synchronisation défini section 5.3.2 page 71, tous les utilisateurs auront la même vision du document global.

Cet automatisme n'est généralement pas assuré par les systèmes de gestion de travail collaboratif tels que CVS [12] ou Subversion [13] qui dans certains cas demandent l'intervention de l'utilisateur pour résoudre des conflits. Ces systèmes reposent sur un serveur central pour gérer les différentes versions des documents et il est donc simple de détecter et de traiter un conflit. Dans un réseau MANet sans serveur central, plusieurs utilisateurs pourraient tenter de résoudre des conflits simultanément sans que l'on puisse garantir que la façon de résoudre le conflit soit la même dans tout le réseau. De ce fait on ne pourrait pas garantir que la résolution de conflits fasse converger

le document global vers une version unique. Ainsi, nous avons choisi de proposer un mécanisme afin d'éviter tout conflit et ainsi d'éviter aux utilisateurs d'avoir à intervenir pour les résoudre.

La méthode de partage utilisée par le système décrit ici consiste à découper le document de façon dynamique en plusieurs parties. Chacune de ces parties ne pourra être modifiée que par un seul utilisateur à la fois (fig. 5.1). De cette façon les écritures concurrentes sur une partie sont impossibles. Chaque fraction du document peut être découpée à nouveau afin que chaque utilisateur ne verrouille que le fragment minimal qu'il est susceptible de modifier.

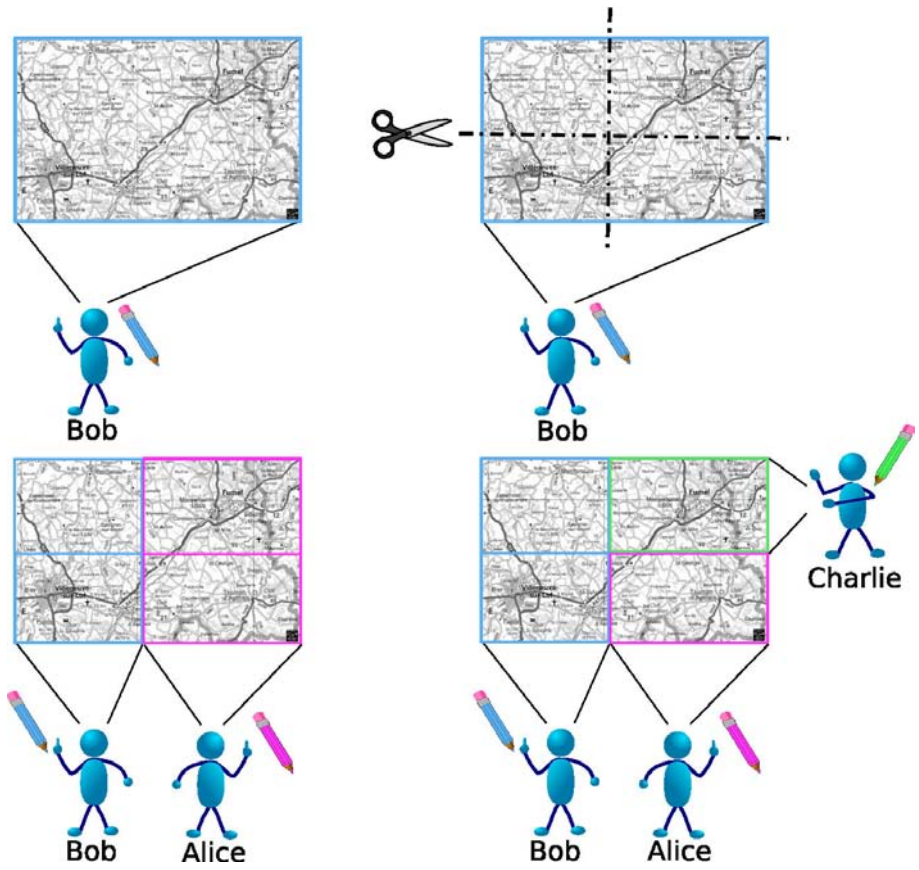


FIG. 5.1 – Principe de base de l'évolution d'un document

Nous avons choisi de représenter le document sous la forme d'un arbre. Cette structure de données permet de maintenir une vision de la hiérarchie du découpage du document. Chaque partie est représentée par une feuille de l'arbre. Lorsqu' une partie (feuille) est découpée, la zone qu'elle représente est remplacée par un sous-arbre composé d'un nœud interne et de feuilles qui contiennent l'information portée par les sous-parties nouvellement crées. La figure 5.2 page ci-contre donne une représentation d'un document en deux dimensions (une carte) découpé en quatre sous-parties. Il paraît naturel qu'un document de dimension n puisse être découpé en 2^n sous-parties. Par exemple, un texte est un document en une dimension qui peut être scindé selon la

position d'un curseur, une carte est un document en deux dimensions fractionnable selon une position de même pour une image 3D qui est un document en trois dimensions. L'existence de documents de dimensions supérieures restera un concept abstrait pour notre étude. La racine du document qui symbolise le document global est en fait vide d'information, l'information effective étant portée par ses sous-arbres. Nous avons choisi dans cette partie d'illustrer nos propos par des figures montrant un document sous la forme d'un document en deux dimensions comme une carte stratégique portée par un arbre quaternaire. Les raisonnements présentés sont généralisables à tout document découpable sous la forme d'un arbre n-aire. C'est notamment le cas dans le chapitre 7 (Simulation) dans lequel nous prenons l'exemple d'un texte porté par un arbre binaire.

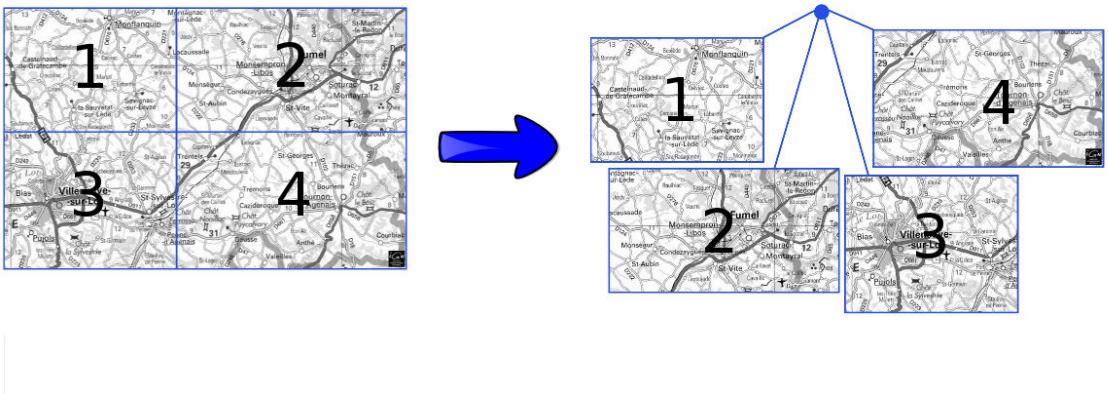


FIG. 5.2 – Représentation sous forme d'arbre d'une carte stratégique

Deux opérations ont été définies pour modifier la structure de l'arbre : le découpage et la fusion. Elles sont illustrées figure 5.3 page suivante.

Découpage. Le découpage est l'opération qui consiste à décomposer une feuille de l'arbre en un sous-arbre de hauteur 1. Les informations portées par la feuille initiale sont réparties entre les nouvelles feuilles en respectant les choix de découpage.

Fusion. La fusion est l'opération inverse du découpage. Elle permet de recompacter un sous-arbre de hauteur 1 en une seule et unique feuille. Les informations portées par l'ensemble des sous-parties sont regroupées pour être associées à la feuille résultant de la fusion.

L'utilisation d'un arbre permet aussi de gagner en efficacité lorsque deux nœuds du réseau qui se rencontrent et synchronisent leurs versions du document pour les mettre à jour (voir section 5.3.2 page 71). On peut, par exemple, faire des mises à jour sur un sous-arbre complet au lieu de mettre à jour une à une les parties qui le composent. Pour cela il suffit de remplacer tout ce sous-arbre et on évite ainsi de le parcourir.

Lorsque des entités mobiles se rencontrent, elles s'échangent des parties du document qui correspondent à des sous-arbres (ou des nœuds le cas échéant). Pour pouvoir positionner n'importe

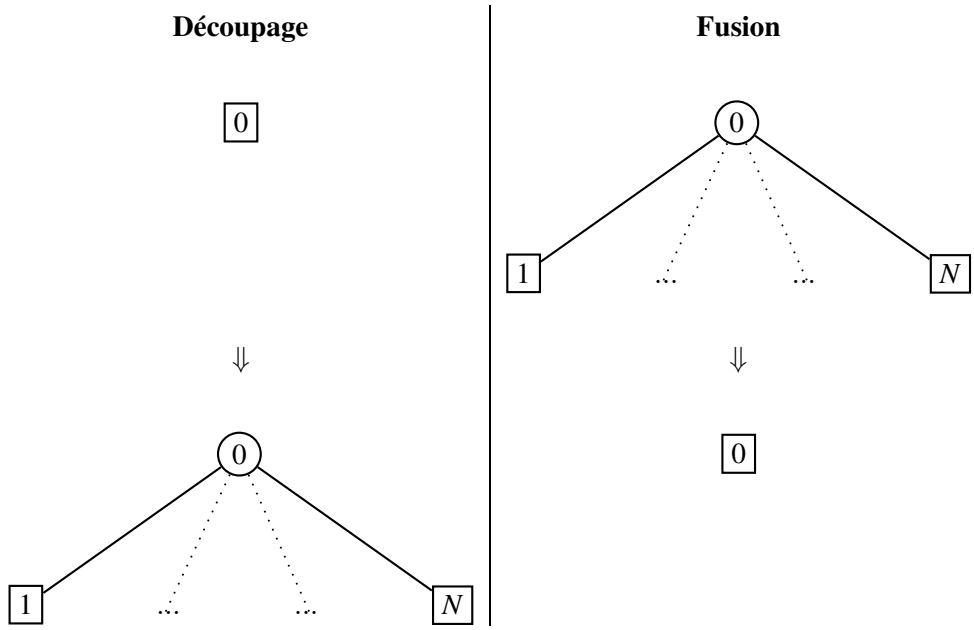


FIG. 5.3 – Représentation des opérations de découpage et de fusion de parties d'un document

quelle partie de l'arbre qui est reçue via le réseau au sein de la structure arborescente portée par l'entité cible, chaque nœud de l'arbre doit être identifié de façon unique. On numérote tout d'abord les fils de chaque sommet interne en fonction de leur position dans l'arbre (fig. 5.4). Ce

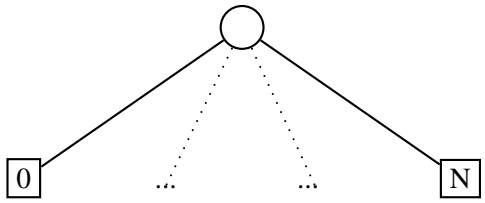


FIG. 5.4 – Nommage des fils d'un nœud interne de l'arbre

raisonnement permet d'associer un entier unique à chaque nœud de l'arbre grâce à la définition d'une fonction de nommage.
$f_{id} : N \rightarrow \mathbb{N}(Nensembledesnudsdel'arbre)$:

$$\begin{cases} f_{id}(n) = 0 \text{ si } n \text{ est la racine de l'arbre} \\ f_{id}(n) = f_{id}(p(n)).id_{loc}(n) \text{ avec } p(n) \text{ le nœud père du nœud } n \\ \qquad id_{loc}(n) \text{ la position de } n \text{ en tant que fils de } p(n) \\ \qquad . \text{ étant l'opération de concaténation} \end{cases}$$

On applique cette fonction uniquement lors de la création d'un nœud dans l'arbre, c'est-à-dire lors du découpage du sommet père du nœud crée. Ce calcul n'implique donc pas de parcours en profondeur complet de l'arbre pour nommer une feuille.

Disposer d'un identifiant unique pour chaque partie du document et aussi très utile lors des échanges entre les différents nœuds du réseau. On peut ainsi désigner un nœud de l'arbre de façon unique pour toutes les entités du réseau. La composition de ce nombre permet à n'importe quelle entité de connaître l'emplacement correspondant dans l'arbre, même si le découpage du document porté par cet utilisateur est différent. Nous verrons section 5.3.2 page 71 que la désignation unique des parties du document permet aussi d'optimiser les coûts de communication en différenciant la description d'un document des données qu'il contient.

La figure 5.5(a) page suivante représente un document (qui pourrait être une carte stratégique), découpé en plusieurs parties. Chaque partie porte l'identifiant qui lui est associé. La figure 5.5(b) représente l'arbre correspondant. Nous avons choisi dans le cas des cartes stratégiques d'utiliser un arbre quaternaire qui permet de découper directement une carte dans ses deux dimensions. Ceci est notamment utilisé en algorithmique géométrique sous le nom de *QuadTree* [77]. Néanmoins, le système que nous décrivons dans cette partie fonctionne avec n'importe qu'elle structure régulière d'arbre, par exemple binaire. Les développements logiciels qui découlent de ce travail sont d'ailleurs paramétrables en ce sens.

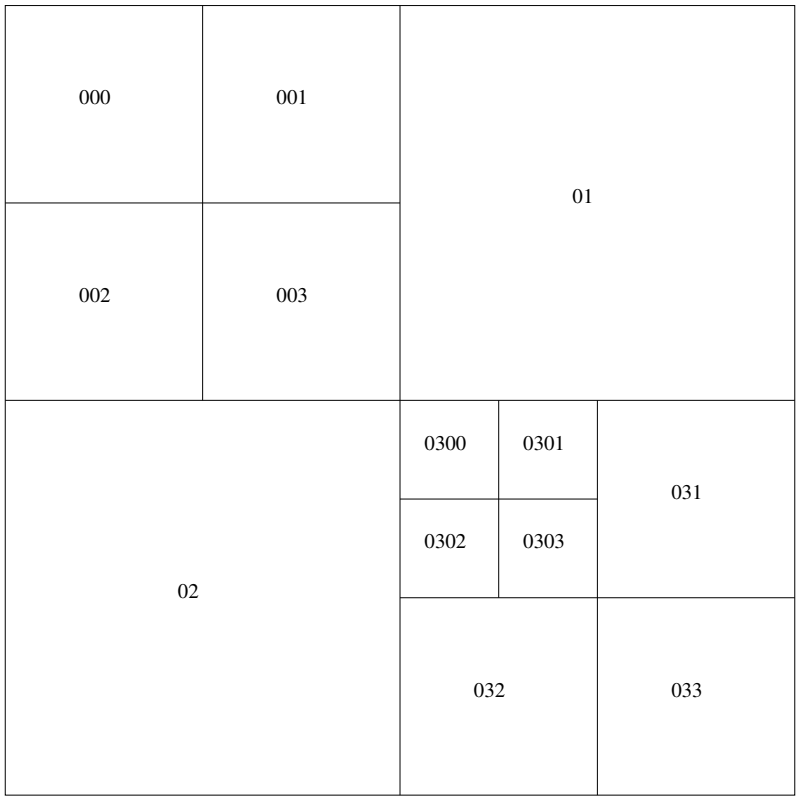
5.2 Gestion de la cohérence du document

5.2.1 États des nœuds de l'arbre

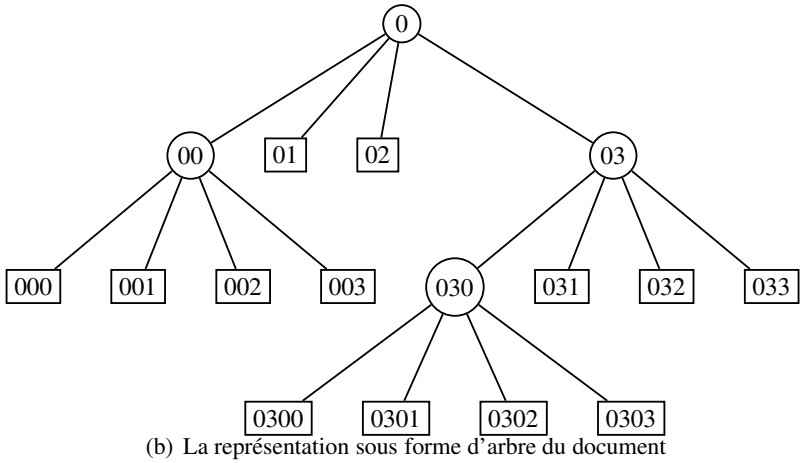
Le document qui fait l'objet du partage est modifié de façon totalement décentralisée par les différentes entités mobiles du réseau. Lorsque ces entités se rencontrent elles doivent pouvoir actualiser leurs versions. Ces mises à jour doivent faire tendre les différents documents vers une version unique. Elle représente le document le plus récent possible à un instant et est appelée **version globale virtuelle**. Elle est composée de toutes les parties du document dans l'état de leur dernière modification. Cette vision doit pouvoir être construite grâce à un procédé déterministe sans utiliser de système de centralisation.

Pour cela nous souhaitons éviter toute divergence de version et ainsi nous proposons de restreindre les modifications sur le document à un seul utilisateur par partie du document à un instant donné. Le travail sur le document peut donc être réparti sans pour autant que la cohérence de celui-ci ne soit remise en cause. Cette restriction de modification, c'est-à-dire cette écriture exclusive, est gérée grâce à la définition d'états sur les différents nœuds internes et feuilles de l'arbre sous-jacent. On compte cinq états :

- pour les nœuds internes :
  - $\boxed{NODE}$: aucune donnée modifiable n'est associée à ces nœuds ;
- pour les feuilles : :
  - $\boxed{CLOSED}$: L'entité mobile dispose d'un « verrou » sur cette partie et peut la modifier,
  - $\boxed{OPENED}$: L'utilisateur conserve le verrou sur cette partie du document mais n'écrit pas sur cette partie et est prêt à céder le verrou à quiconque le désire,



(a) Un exemple de découpage d'un document de dimension 2



(b) La représentation sous forme d'arbre du document

FIG. 5.5 – Un exemple de document sous forme de carte et son arbre associé

– *REQUESTED* : L'entité ne dispose pas des droits en écriture mais déclare vouloir obtenir le verrou sur cette partie du document (afin de pouvoir y apporter des modifications),
– *UNDESIRED* : L'utilisateur ne dispose pas du verrou, n'a pas les droits en écriture et ne souhaite par les obtenir.

5.2.2 Transitions entre les différents états des nœuds

Dans cette section nous décrivons les transitions entre les états définis section précédente. Rappelons qu'un état est associé à chaque partie du document (i.e. à chaque feuille et nœud interne de l'arbre sous-jacent). Les cas sont énumérés de manière exhaustive. On distingue deux types de transitions : les transitions locales à un nœud et celles qui sont dues à une réaction à un événement qui survient au sein du réseau tel que la rencontre avec une autre entité mobile. La table 5.1 représente ces transitions. Dans cette table, T_1, T_2, T_4 et T_6 sont des transitions locales. Les transitions T_3 et T_5 sont des transitions dues à la rencontre d'une autre entité du réseau qui dispose aussi d'une instance de cette partie du document. Elles sont le résultat du processus de synchronisation (lequel sera décrit en section 5.3.2 page 71).

T_1 :	*CLOS.*	→	*OPEN.*	Libération du verrou (transition locale)
	CLOS.	→	*UNDE.*	**Impossible** (on doit le déverrouiller d'abord)
	CLOS.	→	*REQU.*	**Impossible**
T_2 :	*OPEN.*	→	*CLOS.*	Verrouillage de la partie (transition locale)
T_3 :	*OPEN.*	→	*UNDE.*	Une autre entité du réseau a pris le verrou (événement réseau)
	OPEN.	→	*REQU.*	**Impossible**
	UNDE.	→	*CLOS.*	**Impossible**
	UNDE.	→	*OPEN.*	**Impossible**
T_4 :	*UNDE.*	→	*REQU.*	L'entité veut acquérir le verrou (transition locale)
T_5 :	*REQU.*	→	*CLOS.*	L'entité a obtenu le verrou (événement réseau). Cela évite que le verrou ne soit repris par une autre entité
	REQU.	→	*OPEN.*	**Impossible** (voir T_5)
T_6 :	*REQU.*	→	*UNDE.*	L'entité ne désire plus le verrou sur cette partie (transition locale)

TAB. 5.1 – Table des transition entre les états des nœuds

La figure 5.6 page suivante représente par un graphe les transitions entre les différents états possibles d'une feuille. On remarque que les transitions locales (notées loc) sont symétriques alors que les transitions dues à une synchronisation entre nœuds (notées sync) ne le sont pas. Les

transitions T_3 et T_5 sont réalisées simultanément et de façon complémentaire par deux nœuds du réseau ; elles correspondent à la transmission du verrou d'une entité mobile à une autre. La figure 5.7 représente les changements d'états possibles pour une feuille sur deux entités. Il s'agit d'un produit de synchronisation des automates correspondant. Chaque état $\boxed{A/B}$ représente les états sur chacune des entités A et B. Ce graphe provient de la synchronisation sur les transitions T_3 et T_5 sur deux nœuds qui correspond à un passage de verrou. Les transitions locales sont indépendantes sur chaque nœud. Les compositions de transitions locales (a/b) ne sont pas représentées pour garder une bonne lisibilité. Cependant elles ne sont pas nécessaires car on remarque très rapidement la transitivité de ces opérations (le résultat de $a/-$ puis $-/b$ est équivalent à $-/b$ puis $a/-$ dans tous les cas). Cette figure qui rassemble tous les états atteignables possibles montre clairement que le système ne permet pas que deux entités disposent simultanément du verrou sur une même partie. De plus, toutes les trajectoires du graphe de la figure 5.7 étant infinies, nous vérifions que le système est vivant.

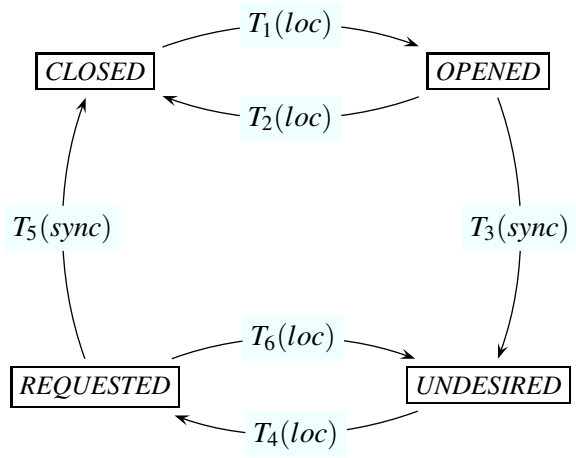


FIG. 5.6 – Graphe des transitions d'état d'une feuille de l'arbre représentant le document partagé

L'introduction d'un état sur les nœuds de l'arbre conduit à des restrictions lors du découpage et de la fusion des parties. En effet, on ne peut découper une partie que si l'on dispose du verrou sur celle-ci. De même, pour fusionner les feuilles d'un sous-arbre, on doit posséder tous les verrous sur ses feuilles (état $\boxed{CLOSED}$). Une seule entité peut ainsi modifier la structure d'un sous-arbre ; elle dispose ensuite du verrou sur la feuille résultante.

Les états que nous avons définis jouent aussi un rôle lors du découpage ou de la fusion de parties du document. Afin de procéder à l'une ou l'autre de ces opérations un utilisateur doit disposer du verrou sur le ou les nœuds de l'arbre associé. Cela signifie que, lors d'un découpage,

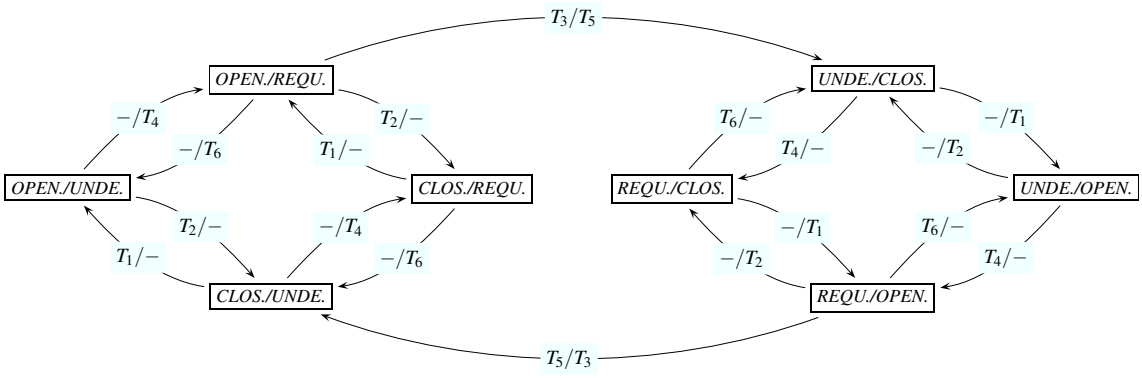


FIG. 5.7 – Graphe des transitions d'états sur deux entités en simultané

la feuille qui sera découpée devra être dans l'état *CLOSED* et de même pour chaque feuille du sous-arbre de hauteur 1 lors d'une fusion. Cette contrainte permet aussi d'éviter que la structure du document ne diverge ; en effet on oblige ainsi à verrouiller toute les parties sur laquelle une opération peut influer.

On pourra noter que la définition et l'utilisation qui est faite des états des nœuds sont aussi des points importants pour garantir la cohérence du document lors de la de synchronisation entre les copies locales de plusieurs utilisateurs (ceci sera détaillé section 5.3.2 page 71).

5.3 Gestion des versions du document et synchronisation entre utilisateurs

Lorsque deux entités du réseau se rencontrent, elles s'échangent des données afin de profiter des changements éventuels apportés par l'autre au document. Lors de ces rencontres, que nous appelons synchronisations, il faut pouvoir ordonner temporellement les différentes parties du document entre les entités afin de déterminer lesquelles doivent être mises à jour. Pour cela nous associons des numéros de version aux nœuds de l'arbre. La comparaison des versions est faite lors de l'exécution du protocole de mise à jour des versions (synchronisation). Ce protocole a été conçu afin de permettre à deux stations qui se rencontrent d'obtenir la version la plus récente possible du document, c'est-à-dire une version composée des parties modifiées en dernier. Ce protocole est aussi garant de la transmission des verrous.

Le protocole de synchronisation doit permettre d'actualiser les versions du document tout en minimisant la quantité de données échangées entre les nœuds. L'optimisation repose sur la définition d'une description synthétique du document et de l'état des différentes parties qui le composent sur chaque nœud.

5.3.1 Gestions des versions

La synchronisation des versions se base sur un numéro de version associé à chaque nœud de l'arbre et sur (la valeur de) son état.

La version locale d'un document dépend de l'ensemble des numéros de version portés par les nœuds de l'arbre qui le représente. La comparaison de deux versions d'un document revient à des comparaisons locales à chaque nœud des arbres sous-jacents. Déterminer la version globalement plus récente du document n'a pas de sens, car ce sont les différentes sous-parties du document qui sont modifiées de façon décentralisée. La notion de version pour un document est donc distribuée sur l'ensemble des sous-arbres qui le composent.

Un numéro de version, entier, est donc porté par chaque nœud de l'arbre. Ce numéro est initialisé à 0 et incrémenté à chaque modification du contenu de la partie du document associée, comme par exemple l'ajout d'un soldat ennemi. Lorsqu'une partie est découpée, son numéro de version n'est pas modifié, par contre ce nombre sera incrémenté lors de la fusion des nœuds fils résultant de ce découpage.

Pour définir parmi deux parties *P1* et *P2* la plus à jour, c'est-à-dire celle qui a été modifiée en dernier, nous utilisons la procédure suivante :

1. Si les numéros de version sont différents alors le bloc avec le numéro de version le plus grand est le plus récent ;
2. Sinon si *P1* (respectivement *P2*) est dans l'état $\boxed{NODE}$ et que *P2* (respectivement *P1*) ne l'est pas, alors *P1* (respectivement *P2*) est le plus à jour ;
3. Sinon, les versions sont équivalentes et aucune mise à jour n'est nécessaire.

Remarque : lors de la comparaison des versions de nœuds internes de l'arbre (état $\boxed{NODE}$), si une version plus récente est identifiée alors cela signifie qu'à partir de cette profondeur tout le sous-arbre, et notamment sa structure, a été modifié et qu'il doit être complètement actualisé.

Justification de la remarque : La version d'un nœud interne n'est modifiée que lors de changements dans la structure de l'arbre. Si deux nœuds internes ont une version différente cela signifie que le sous-arbre de celui qui a la version la plus récente a été fusionné (incrémentation de la version) puis découpé à nouveau (retour à l'état $\boxed{NODE}$). Ainsi, les découpages entre ces deux versions sont différents et la comparaison des sommets situés en aval n'a pas de sens.

Explications de la procédure :

Cas 1 et 3 :

version du nœud de l'entité 1 (N1)	version du nœud de l'entité 2 (N2)
v1, feuille	v2, feuille
v1, nœud interne	v2, nœud interne

$v1 > v2 \Longrightarrow$ **N1 est plus récent que N2**
$v1 = v2 \Longrightarrow$ **N1 est dans la même version que N2**

Cas 2 :

version du nœud de l'entité 1 (N1)	version du nœud de l'entité 2 (N2)
v1, feuille	v2, nœud interne

$v1 > v2 \Longrightarrow$ **N1 est plus récent que N2**
$v1 = v2 \Longrightarrow$ **N2 est plus récent que N1**

Justification du cas 2 : Lorsqu'un nœud est créé, c'est une feuille. Le passage a l'état de nœud interne ne peut qu'être postérieur et donc plus récent. Lors d'une fusion, le numéro de version est incrémenté, il permet donc d'identifier le nœud le plus récent.

5.3.2 Synchronisation

La synchronisation entre deux nœuds du réseau est la procédure qui permet à chacun de ces nœuds d'avoir la version du document la plus récente possible en fonction de sa version et de celle de l'autre entité. Cette procédure permet aussi de faire circuler les verrous entre les entités et notamment de transmettre le verrou d'une entité qui dispose d'une sous-partie dans l'état *OPENED* à une autre qui a cette partie dans l'état *REQUESTED* et ainsi d'appliquer les transitions T_3 et T_5 respectivement (section 5.2.2 page 67).

Pour cela, les entités se synchronisent, au sens réseau du terme (ceci sera détaillé dans la partie simulation, section 7.6.2 page 101). Elles s'envoient alors une description de leur arbre respectif. Il n'est en effet pas nécessaire de transmettre toutes les données, comme les annotations stratégiques (cela interviendra plus tard), pour décider de l'information à mettre à jour. Cette description est construite par un parcours de l'arbre en profondeur et en ordre préfixe, et résulte en une suite de triplets de la forme : (identifiant du nœud de l'arbre, numéro de version, état). Ces informations suffisent à décrire complètement l'arbre.

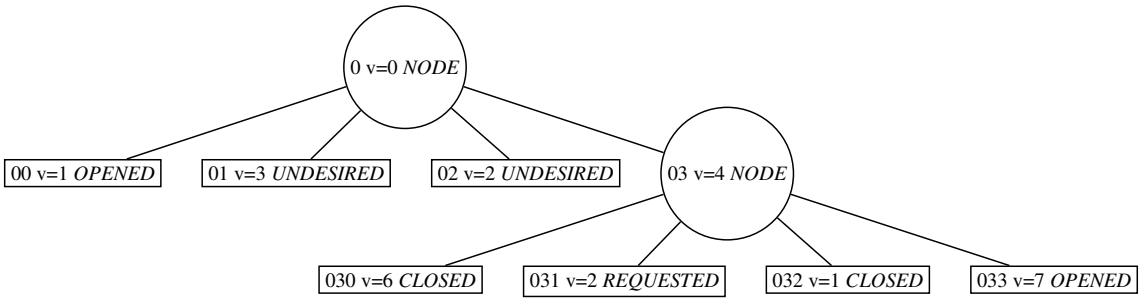


FIG. 5.8 – Exemple d'arbre avec numéros de versions et état des parties

Par exemple l'arbre de la figure 5.8 est décrit de façon suffisante par la chaîne de caractères suivante :
0_0_NODE.

00_1_OPENED.01_3_UNDESIRED.02_2_UNDESIRED.03_4_NODE.
030_6_CLOSED.031_2_REQUESTED.032_1_CLOSED.033_7_OPENED

La transmission de ces descriptions entre deux nœuds leur permet de déterminer les informations effectives qu'ils doivent s'échanger pour que chacun puisse obtenir la meilleure version possible du document. Cela permet aussi de déterminer quels verrous pourront circuler, c'est-à-dire ceux pour lesquels il est possible d'appliquer les transitions T_3 et T_5 de la table des transitions (cf table 5.1 page 67).

La figure 5.9 page suivante représente les arbres de deux entités mobiles A et B avant et après synchronisation. Pour chaque nœud de l'arbre on voit son identifiant, son numéro de version et son état. Rappelons que lorsqu'une entité a une version plus récente d'une partie, elle n'a pas besoin de l'actualiser. On notera plus particulièrement les nœuds 00 et 031 sur lesquels une transmission de verrou a lieu. Par exemple pour le nœud 00 l'entité A applique la transition T_3 de la table 5.1 page 67 qui correspond à donner le verrou. Simultanément l'entité B met à jour ses informations sur cette partie du document pour passer à la version 2 puis applique la transition T_5 qui consiste à récupérer le verrou sur cette partie. Le processus se déroule de la même façon pour le nœud 031. Les sous-parties 01, 02, 032 et 033 sont simplement mises à jour du point de vue de leur contenu et de leur numéro de version.

5.4 Conclusion

Nous avons présenté dans ce chapitre un système original supportant l'édition collaborative d'un document partagé dans un réseau MANet. Ce système permet de répartir les tâches entre les différents utilisateurs tout en assurant la cohérence permanente du document.

D'un point de vu technique, nous avons présenté un mécanisme de découpage de documents reposant sur l'utilisation d'une structure arborescente et les opérations qui permettent de la modifier. Des états assimilables à des verrous sont associés à chaque sous partie du document. Ce principe présente l'avantage d'assurer de manière permanente la cohérence du document et la possibilité de converger vers un document global unique à partir des différentes copies locales grâce à une synchronisation des différentes versions.

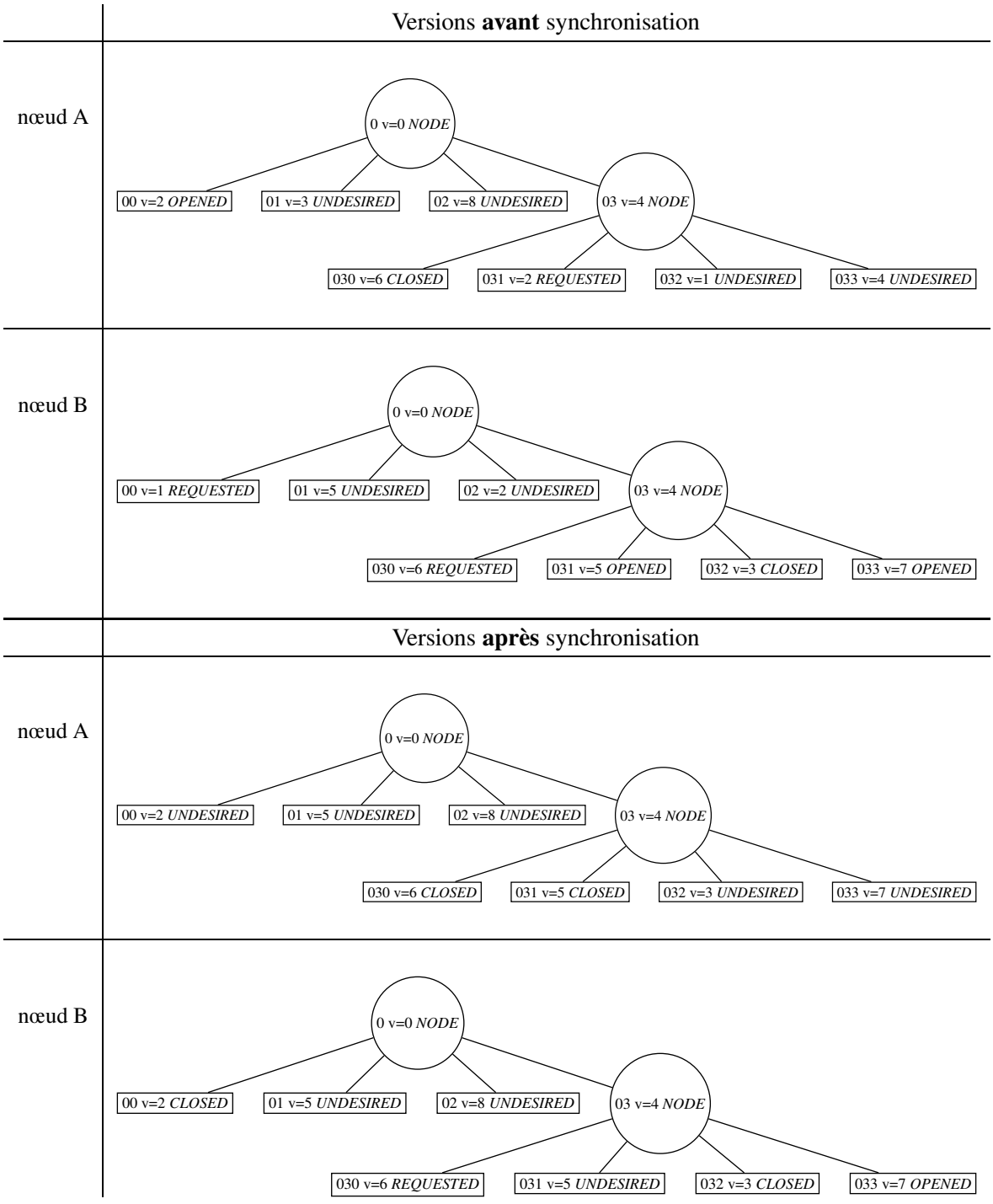


FIG. 5.9 – Exemple de synchronisation entre deux entités mobiles

Chapitre 6

Problème de la perte de verrous sur certaines parties, information floue et logique associée

6.1 Problème

Nous avons présenté chapitre 5 un système supportant l'édition collaborative d'un document partagé entre plusieurs utilisateurs. Celui-ci repose sur le découpage du document concerné en sous-parties auxquelles sont associées des verrous. A chaque instant, le droit de modification d'une partie est attaché à une seule entité du réseau. Si cette entité vient à disparaître, les parties sur lesquelles elle pouvait écrire ne sont plus modifiables, les verrous associés étant perdus. Nous appelons ce phénomène « perte de région ». On pourra remarquer que dans le cadre totalement mobile dans lequel nous nous plaçons, on ne peut pas différencier une disparition momentanée d'une disparition définitive.

Ce problème de perte de verrous est insoluble dans un cadre totalement décentralisé. En effet, aucune entité du réseau ne peut savoir ni décider si le verrou est encore présent, car chaque nœud ne dispose que d'une vision locale. Par contre, ce problème est décidable dans un contexte où le système est supervisé par une autorité. Si une telle autorité existe, elle peut alors régénérer des verrous qu'elle estime perdus et ainsi redonner une vie aux régions perdues.

Notre solution peut être spécialisée dans cette optique car le contexte dans lequel nous nous plaçons est militaire et hiérarchisé. Nous présenterons dans cette partie comment utiliser ce contexte hiérarchique pour permettre la génération de nouveaux verrous.

6.2 Le contexte hiérarchique militaire

Tous les utilisateurs du système, qui n'ont qu'une vision locale de la distribution des verrous, ne doivent pas pouvoir en générer de nouveaux car ceci remettrait en question la cohérence des données ; l'existence et l'unicité d'une version que l'on peut qualifier de plus récente seraient alors compromises.

On souhaite pourtant pallier la perte de verrous en introduisant de nouvelles instances de ceux qui sont perdus définitivement. La solution proposée se base sur la définition d'une unique entité qui seule pourra en générer de nouveaux. Il est évident que le contexte d'utilisation sera essentiel pour pouvoir définir cette entité.

Dans le cas général des réseaux MANets on ne peut pas, *a priori*, différencier certains utilisateurs pour les autoriser à accomplir certaines tâches critiques et ainsi leur attribuer des droits supplémentaires et permanents. En revanche, le contexte d'application militaire dans lequel nous nous plaçons ici est particulier. Les utilisateurs appartiennent à un groupe hiérarchisé dans lequel certaines entités possèdent des droits supérieurs aux autres.

6.3 Présentation de la solution pour récupérer des régions

Dans le contexte dans lequel nous nous plaçons, nous faisons donc l'hypothèse (cf section 6.2) que les verrous peuvent être générés par une autorité unique et commune à tout le système. En effet, si certains membres du réseau peuvent régulièrement se synchroniser avec un supérieur hiérarchique, alors celui-ci pourra introduire de nouveaux verrous lors de ces synchronisations. Nous faisons l'hypothèse que certains utilisateurs possèdent un dispositif qui leur permet de communiquer avec la hiérarchie de façon (au moins) discontinue. Certains soldats peuvent par exemple être équipés de moyen de communication à plus longue distance que celui utilisé couramment entre les fantassins, leur permettant d'établir des liaisons avec un poste de commandement. Ces connexions permettent à l'autorité d'estimer le temps entre deux mises à jour d'une même partie et donc de suspecter la disparition de certains verrous. Si cette autorité s'aperçoit que la version d'une partie n'évolue pas pendant un temps (considéré comme) trop long, alors il peut être décidé de réintroduire un nouveau verrou.

Dès que l'on autorise certains nœuds à introduire de nouveaux verrous dans le système, alors la cohérence de celui-ci peut être compromise. En effet, un verrou peut avoir été introduit à tort lorsque l'entité qui possède la version actuelle n'a disparu que temporairement. En conséquence, nous proposons une méthode pour gérer le cas où il existe plusieurs verrous sur une même partie du document qui permet de converger progressivement vers la situation classique d'un seul verrou par partie. Enfin, dans le contexte militaire dans lequel nous nous plaçons, nous ne pouvons écarter les problèmes liés à la sécurité et notamment à la certification des verrous afin de s'assurer que personne n'introduise de verrou sans y être autorisé. La solution choisie pour assurer la sécurité

du système est basée sur l'utilisation de cartes à puce.

6.4 Utilisation des cartes à puces et authentification de la hiérarchie

6.4.1 Introduction aux cartes à puces

Dans cette section nous ne considérons que les cartes à puce à microprocesseur [45]. Nous ne faisons donc pas état des cartes à mémoire qui ne possèdent pas de capacité de calcul ou des cartes à logique câblée destinées à des applications spécifiques.

La puce embarquée sur une carte comporte un processeur dont la fréquence varie entre 4,77 MHz et 100 MHz pour les plus performantes ; cette puce est aussi équipée d'un coprocesseur cryptographique auquel est associé un générateur de nombres aléatoires. La mémoire dont dispose une carte à puce est de trois types différents :

- ROM (*Read Only Memory*) : cette mémoire n'est pas modifiable et contient le plus souvent le système d'exploitation de la carte et éventuellement certaines applications. Elle est « gravée » lors de la fabrication de la carte et conserve son contenu pendant tout le cycle de vie de la puce.
- RAM (*Random Access Memory*) : l'analogie peut être faite avec la RAM d'un ordinateur classique ; cette mémoire est à accès rapide et son contenu disparaît lorsque la carte n'est plus alimentée.
- EEPROM (*Electrical Erasable Read Only Memory*) : cette mémoire conserve son contenu lorsque la carte n'est pas alimentée ; elle est plus lente que la RAM.

La communication (avec ou sans contact) entre la carte et l'application sur la station de travail s'effectue à travers un terminal appelé CAD (*Card Acceptance Device*). Les échanges se font sur le modèle commande/réponse. Le protocole utilisé est le protocole APDU [45] (*Application Protocol Data Unit*). On n'accède donc pas directement aux informations contenues dans la puce mais c'est l'application fonctionnant sur la carte qui répond à des requêtes. Ce mécanisme assure une partie de la sécurité logicielle.

La technologie Java Card [78] est une solution disponible dans le domaine des cartes à puce. Java Card repose comme Java sur l'utilisation d'une machine virtuelle et c'est aujourd'hui le système d'exploitation le plus utilisé. Ces cartes embarquent une partie de la machine virtuelle JCVM (*Java Card Virtual Machine*) qui est l'interpréteur de byte code. Le convertisseur qui constitue la seconde partie de la machine virtuelle, est placé sur la machine hôte. Il transforme l'ensemble des fichiers *.class* nécessaires à l'application générés par le compilateur en un unique fichier *.cap* directement installable sur la carte. Une application installée sur une carte à puce de type Java Card se nomme une *applet*.

L'équipe SOD du LaBRI ayant une forte compétence en Java Card, cette solution est au cœur de la mise en œuvre choisie pour implémenter cette évolution de notre système.

6.4.2 Intérêt de l'utilisation des cartes à puces pour la réintroduction de verrous

De par leur conception, les cartes à puces sont des systèmes sécurisés. Des mesures matérielles et logicielles y sont mises en place pour contrer les attaques éventuelles. Bien qu'on ne puisse pas considérer que les cartes à puces soient invulnérables, elles sont extrêmement sûres et sont à l'heure actuelle l'un des supports d'exécution les plus fiables. En effet, leur niveau de sécurité matériel et logiciel est certifié par les CESTI (Centres d'Évaluation de la Sécurité des Technologies de l'Information) qui sont chargés d'évaluer la sécurité et de garantir le respect des spécifications. Le niveau de sécurité des cartes à puce et l'impossibilité d'accéder directement à leur mémoire permettent de les utiliser pour conserver des clés de chiffrement (secrètes ou privées). De plus, leurs capacités cryptographiques permettent d'exécuter des opérations utilisant ces clés directement dans la carte.

Nous proposons ainsi d'utiliser des cartes à puce pour assurer la sécurité de notre système. Outre la sécurité des communications, elles rendent possible l'authentification des utilisateurs et la certification des données (ce qui sera expliqué plus tard). La cryptographie repose sur l'utilisation de clés qui ne doivent être possédées que par les utilisateurs auxquelles elles sont destinées. La distribution de ces clés est donc un problème majeur. Nous utilisons les cartes à puce pour stocker les clés, leur distribution relève donc de celle des cartes. Dans le contexte militaire dans lequel nous nous plaçons, les cartes peuvent être configurées et distribuées par la hiérarchie avant le début d'une opération.

Les cartes à puce que nous utilisons contiennent :

1. une identité unique pour chaque entité mobile,
2. un couple clé privée/clé publique propre à chaque unité,
3. la ou les clés publiques du ou des officiers (si les verrous sur différentes parties sont générées par des officiers différents),
4. un ensemble de clés secrètes partagées par le groupe. Ces clés peuvent être utilisées pour chiffrer par exemple les communications sans fil (donc reçues par tous les nœuds à proximité, amis ou ennemis).

Chaque entité mobile est équipée d'une carte configurée comme indiqué précédemment et peut ainsi vérifier l'authenticité d'un verrou. En effet, le stockage de ces informations permet à l'autorité qui peut générer des verrous de les certifier (les signer avec sa clé privé). Les certificats résultants sont vérifiables en utilisant les clés publiques associées qui sont dans la carte.

Les clés secrètes partagées peuvent permettre aux entités mobiles de stocker sur leur périphérique leur copie locale du document et/ou de communiquer de façon sécurisée en utilisant au travers de la carte à puce un algorithme symétrique de chiffrement.

La sécurité repose sur le fait que les accès aux données présentes sur la carte sont contrôlés par l'application qui y est installée. En outre, l'utilisation de la carte est conditionnée par son déblocage grâce à un code PIN (*Personal Identification Number*). Le seul moyen de passer outre

ce mécanisme serait d'avoir accès directement au périphérique d'un utilisateur après que celui-ci se soit authentifié en composant son code PIN.

6.5 Conséquence de l'introduction de la hiérarchie et de la régénération des verrous dans le système global

Lorsque la hiérarchie considère que le verrou sur une partie a disparu du système, ce qui peut être interprété comme la perte d'un utilisateur, elle introduit un nouveau verrou sur cette partie.

Cependant dans un contexte MANet, on ne peut ni connaître l'identité de l'entité mobile qui disposait du verrou perdu ni garantir que celle-ci ne réapparaîtra pas plus tard dans le système. Cette entité peut donc revenir avec un verrou désormais obsolète et on se retrouve alors dans la situation où plusieurs entités peuvent modifier la même partie du document. Ce problème doit donc être pris en compte lors de la génération des nouveaux verrous afin de garantir que l'on pourra toujours converger vers une version globale virtuelle unique.

Il nous faut donc adapter le système pour différencier les différentes versions d'un verrou. Nous utilisons un identifiant pour chaque verrou qui est associé non seulement à la sous-partie correspondante mais aussi à chaque information ajoutée sous son contrôle. De ce fait on pourra identifier les informations présentes lors de la création du verrou le plus récent ou ajoutées avec celui-ci.

La synchronisation entre une version modifiée avec un verrou récent et une version modifiée avec un verrou obsolète doit aboutir à une version cohérente. Pour cela une solution simple serait que les informations qui ont été notées avec l'ancien verrou et après la génération du nouveau verrou soient ignorées. Pourtant, ces informations peuvent contenir des données stratégiques essentielles à la sécurité des personnels participant à l'opération. Afin de les différencier des informations de la version la plus récente, plutôt que de les effacer, elles seront notées comme **floues**. Par exemple cela reviendra à se souvenir qu'il y a peut-être un tank à un certain endroit plutôt que d'ignorer cette information ; il s'agit d'un « principe de précaution ». Ensuite, un utilisateur disposant du bon verrou sur cette partie pourra soit confirmer cette information et la faire apparaître comme toute autre information à jour, soit l'infirmer et l'effacer.

6.5.1 Prise en compte de la réintroduction de verrous au niveau de la structure des informations

La génération de nouveaux verrous induit le fait qu'une partie de la carte peut éventuellement être modifiée par plusieurs utilisateurs de façon concurrente, dans le cas où le verrou initial aurait été estimé comme perdu définitivement à tort.

Pour que le processus de synchronisation aboutisse à une version du document qui converge, il faut donc pouvoir déterminer pour chaque annotation si elle a été réalisée avec un nouveau

verrou ou un ancien. Pour cela nous ajoutons des informations aux annotations et notamment la version du verrou avec laquelle a été faite la dernière modification. Une annotation devient donc un quadruplet de la forme (clé, sens, état, Id_verrou).

Identifiant d'une information : le champ clé

La clé permet d'identifier une annotation de façon unique. Chaque fois qu'une information est ajoutée au document, comme la position d'un tank dans le cas de Shaadhoc, on lui associe un identifiant numérique entier. L'unicité de cette clé peut, par exemple, être obtenue par l'emploi d'une fonction de hachage [79] utilisée sur le contenu de l'information combiné avec un générateur de nombres aléatoires. Cet identifiant n'est aucunement lié à la signification de cette information, il est simplement utilisé pour faire référence à la représentation logique de celle-ci.

Sens d'une information

Le champ « sens » contient des données contextuelles qui permettent d'interpréter l'annotation. Dans le cas d'une carte stratégique ce champ peut contenir des renseignements sur les coordonnées de l'objet observé, les informations nécessaires pour le représenter et même des commentaires. Par ailleurs, l'application finale pourra utiliser ce champ pour exprimer des relations entre certains types d'information. Par exemple, dire qu'une région est vide (cf. section 6.6 page 85) est incompatible avec le fait qu'il y ait un soldat ennemi dans celle-ci. Ces commentaires peuvent être utilisés par l'application finale pour définir une logique sur les annotations.

Les différents états d' une information

Nous associons un état à chaque annotation portée par le document. Lorsqu'une information est placée sur le document elle est simplement notée dans l'état présent (P). Une information modifiée avec un verrou obsolète sera notée comme floue (?) lors de la synchronisation avec une version valide. Un état permet aussi de marquer une annotation comme effacée (E). Les différents états dont on dispose sont donc :

- P : Présent ;
- E : Effacé ;
- ? : Flou.

Information floue. Si de nouveaux verrous ont été introduits, une donnée est considérée comme floue lorsqu'elle a été ajoutée ou modifiée dans une version divergente du document, c'est-à-dire avec un verrou considéré obsolète. C'est au moment de la synchronisation entre deux entités qu'elle sera notée comme floue.

Information Effacée. L'effacement d'information se produit dans deux cas :

- Lorsqu'une information floue a été identifiée comme invalide par un utilisateur qui dispose du verrou sur cette partie du document, alors elle est notée comme effacée. Ce changement d'état permet de tenir compte de l'observation faite par ce dernier utilisateur, tout en gardant une trace de cette annotation. La conservation de cette donnée dans l'état effacé permet de faire en sorte que le constat d'invalidité se diffuse dans le réseau (voir tableau 6.2 page 90) et donc évite de devoir refaire cette observation sur chaque entité qui dispose de l'information notée floue. Si l'information était supprimée cela entrainerait la conservation de l'état flou lors des synchronisations.
- Le passage dans l'état effacé (plutôt qu'une suppression totale) peut aussi provenir de la volonté de l'application d'archiver toutes les annotations ayant existé à un instant donné. Ceci n'a pas d'autre conséquence sur le comportement de l'application que l'occupation mémoire supplémentaire engendrée.

6.5.2 Influence de l'introduction de nouveaux verrous sur la fusion de sous-parties

Afin d'effectuer une fusion de sous-parties d'un document, une entité doit posséder le verrou sur toutes les feuilles du sous-arbre à fusionner. Lors de cette fusion les différents verrous sur les sous-parties disparaissent pour laisser place à un verrou unique associé au nœud père du sous-arbre fusionné comme sur la figure 6.1. Sur cette figure et de la même façon pour les autres figures de cette section, nous notons pour chaque nœud de l'arbre l'identifiant du verrou qui lui est associé, représenté entre crochets. Dans cet exemple tous les verrous ont l'identifiant [1] qui est celui du verrou initial de chaque nœud de l'arbre (celui attribué au moment de la création de la feuille).

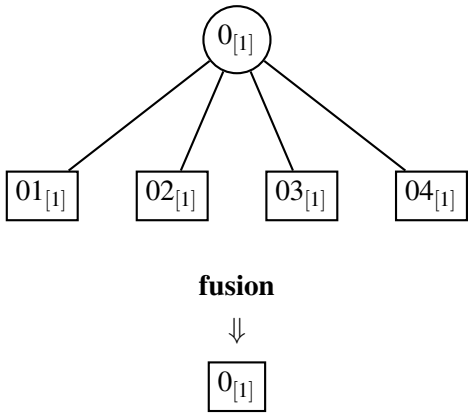


FIG. 6.1 – Exemple de fusion sur l'arbre représentant le document

Lorsqu'un verrou sur une sous-partie du document est considéré comme perdu, un nouveau verrou est généré. On peut voir sur la figure 6.2 page suivante un cas de fusion problématique. La figure 6.2(a) montre la vision du document d'une première entité qui dispose de verrous obsolètes ([1][1][1][1]). Sur l'autre partie, figure 6.2(b), une deuxième entité dispose de verrous valides (ou

au moins plus récents : [2][3][4][2]) sur les feuilles de son arbre. Lors de la fusion les feuilles disparaissent et la seule feuille qui reste correspond au nœud racine de l'arbre. Le verrou porté par cette partie du document a donc l'identifiant correspondant à celui du nœud père qui n'a pas évolué lors des différentes régénération de verrous. Ainsi, une fois qu'une fusion est effectuée, on ne peut plus détecter le fait qu'une des entités disposait de verrous obsolètes. En effet, on observe le même résultat après les fusions, verrou de version [1], sur les figures 6.2(a) et 6.2(b).

Le cas présenté sur la figure 6.2 est un des rares cas qui peut poser problème car pour procéder à une fusion une entité doit posséder les verrous sur tout un sous-arbre de hauteur 1 et un verrou ne peut être possédé que par une seule entité. Pour être dans un cas de ce type il faut donc qu'il existe des verrous en double sur chaque feuille du sous-arbre considéré et que les entités qui les possèdent fusionnent une même partie.

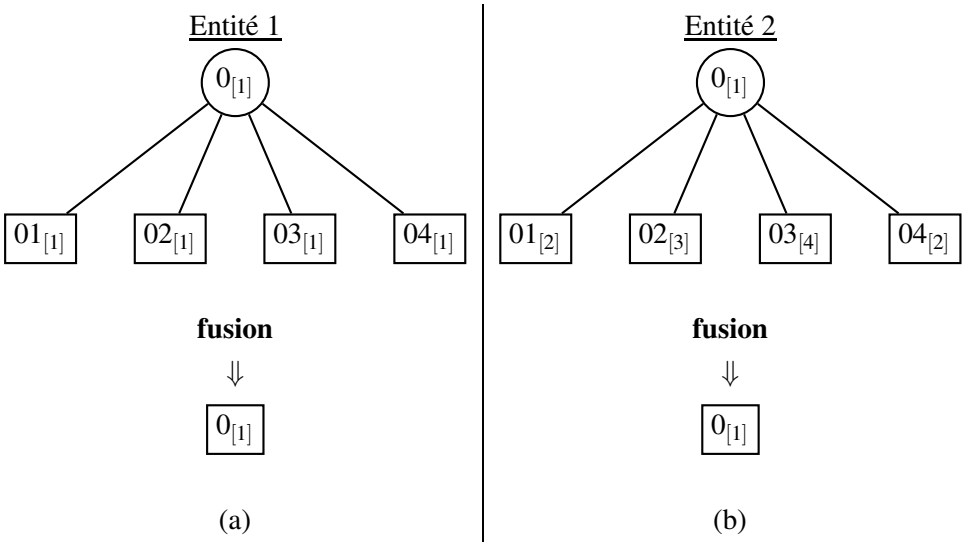


FIG. 6.2 – Problème pouvant survenir lors d'une fusion

6.5.3 Algorithme de régénération de verrous

Afin de résoudre le problème des fusions, la solution proposée consiste à générer, au moment de la création d'un nouveau verrou, une nouvelle version du verrou pour tous les sommets de la branche de l'arbre à laquelle appartient la feuille considérée comme perdue (algorithme décrit dans cette section). Le résultat de cette méthode est présenté figure 6.3 page ci-contre. Une fois la fusion effectuée l'entité 2 conserve une version à jour ([8], numéro de version minimal obtenu dans cette configuration avec l'algorithme) par rapport à l'entité 1 qui reste obsolète ([1]). On peut ainsi toujours différencier un arbre dans lequel on a introduit un nouveau verrou.

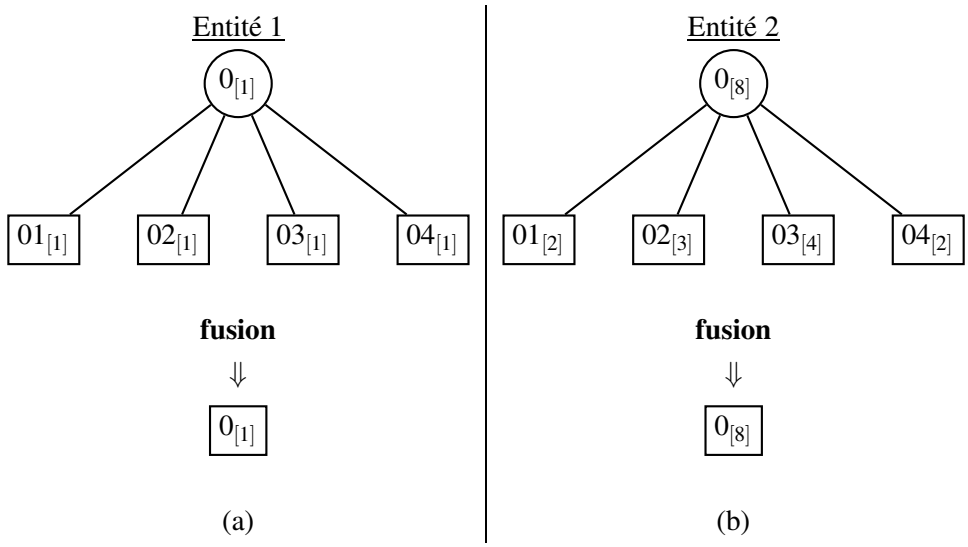


FIG. 6.3 – Solution pour remédier au problème inhérent à la fusion lors de la présence de verrous régénérés

Nous définissons un algorithme adapté à cette solution (algorithme 4) dans lequel :

- t est l'arbre dans lequel le nœud n sur lequel s'applique l'algorithme est inclus,
- *racine_arbre* est la fonction qui associe à un arbre son nœud racine,
- *verrou* est la version du verrou associée au nœud,
- P est la fonction qui associe son père à un nœud.

Algorithme 4 Régénération de verrous

```
reg(n,t){
n.verrou = n.verrou + 1
if n ≠ racine_arbre(t) then
   reg(P(n),t)
end if
}
```

L'exécution de cet algorithme est effectuée par l'autorité habilitée. La figure 6.4 page suivante représente la régénération d'un verrou par une autorité (à gauche sur la figure) après une synchronisation de celle-ci avec une entité du réseau, par exemple un fantassin (à droite). Dans un premier temps le soldat établit une connexion avec sa hiérarchie et procède à une synchronisation pour mettre à jour la copie possédée par la hiérarchie et éventuellement la sienne. A ce moment la hiérarchie peut décider qu'une partie du document aurait dû évoluer depuis le dernier rapport et générer un nouveau verrou grâce à l'algorithme 4 décrit précédemment. Enfin l'autorité n'a plus qu'à « injecter » ce nouveau verrou dans le système.

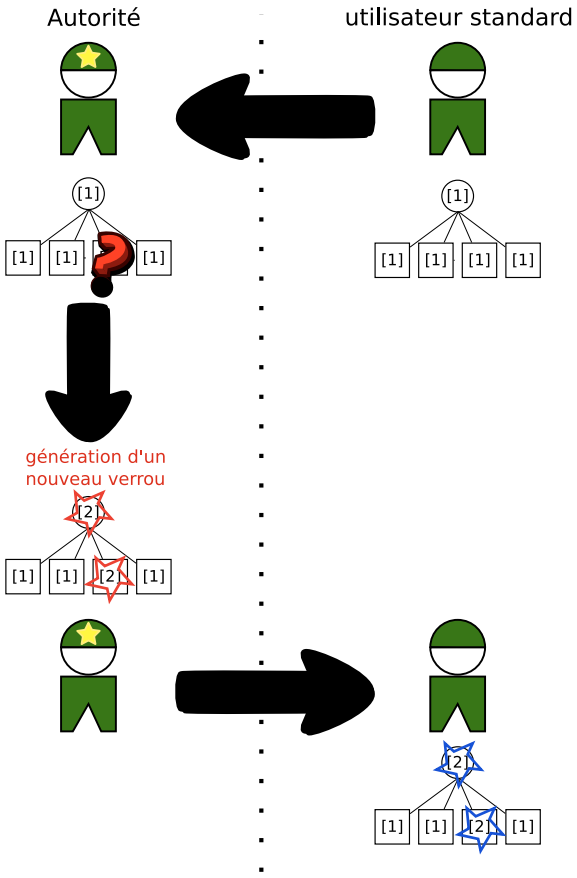


FIG. 6.4 – Régénération d'un verrou par une autorité

6.6 Information sur une surface

Dans cette section, on s'intéresse uniquement au cas particulier du partage d'une carte stratégique car nous allons définir des propriétés sur les informations elles-mêmes. Les propositions faites jusqu'ici sont en effet applicables dans le cas générique de tout document découpable en un arbre n-aire. Les informations considérées sont des annotations sur la carte qui dans un premier temps attachent un objet tel qu'un tank à une position géographique.

Pourtant, dans certains cas, on peut souhaiter annoter toute une zone de la carte, notamment lorsqu'on peut identifier une particularité commune à toute cette zone : zone minée, bâtiment, nature de sol particulière, etc. Il apparaît donc intéressant d'introduire la notion d'information sur une surface.

Une information sur une surface se différencie d'une annotation ponctuelle par le fait qu'elle donne une propriété globale sur toute une zone de la carte. Le sens que l'on donne à une information de surface peut aussi influer sur le sens des autres annotations. Ainsi, lorsque l'on souhaite indiquer sur la carte qu'une zone est vide (Rien à Signaler *RAS*), ce renseignement a une influence sur certaines informations ponctuelles déjà annotées, comme la présence d'un soldat ennemi, et a pour conséquence leur suppression.

Information de zone vide (*RAS*)

On considère un nouveau type d'information pour indiquer qu'après vérification d'une zone de la carte, celle-ci s'avère vide de tout élément pris en compte par l'application. Cette information nous permet donc de différencier les cas où une zone n'a pas été contrôlée du cas où elle l'a été et où aucun élément qui donne habituellement lieu à une annotation n'a été relevé.

Ce renseignement a donc la particularité d'exclure toute présence dans cette zone. Ceci nous conduit à définir la notion d'information exclusive sur une zone qui est un cas particulier de relation entre les informations.

Information exclusive sur une zone

Le renseignement de zone vide est en réalité un cas particulier d'information exclusive sur une zone. Ce caractère exclusif est codé par le champ « sens » de l'annotation. On distingue deux types d'exclusivité qui sont l'exclusivité relative ou absolue. Une **information exclusive absolue** sur une zone n'est compatible avec aucune autre annotation dans cette zone alors qu'une **information exclusive relative** n'interdit la présence que de certains éléments. Nous ferons l'hypothèse qu'une information *RAS* est absolue. Le raisonnement de la suite de ce chapitre se fera sur ce type d'information. Il pourrait aisément être généralisé dans le cas d'une exclusivité relative.

6.7 Logique associée aux changements d'états

6.7.1 Modification locale de l'état d'une information par constats

Le tableau 6.1 montre les modifications qui se produisent sur l'état d'une information sur un nœud particulier qui dispose du verrou approprié en fonction d'un constat qui est fait par l'utilisateur. Un constat est l'observation qui peut être faite par rapport à une annotation sur le document. Ces constats peuvent être de trois sortes :

- V : l'information est vérifiée,
- $\bar{\text{V}}$: l'information n'a plus de sens,
- *RAS* : la zone en question est vide.

Un utilisateur ne peut pas constater une information floue. Une information floue relève de la synchronisation entre une version valide d'une partie du document et une version obsolète.

On remarque que lorsqu'un utilisateur fait le constat qu'une information est présente, alors quel que soit l'état d'une information avant le constat celle-ci passe dans l'état P.

Si l'utilisateur fait le constat de l'invalidité de cette information alors si celle-ci était dans l'état P, il peut choisir de l'effacer totalement (état $\emptyset$) ou bien d'en garder une trace et de la faire passer dans l'état E. Si l'annotation invalide était dans l'état floue (?), elle passe forcément dans l'état effacée (E). Ceci permet d'archiver l'observation d'invalidité tout en supprimant l'annotation de la vision du document (cf. 6.5.1). Cette observation pourra ainsi être diffusée dans le réseau grâce aux synchronisations entre les entités et l'état flou de cette information tendra à disparaître du système (voir section 6.7.3). De plus, lors qu'une information est dans l'état effacé elle ne sera jamais supprimée du système pour assurer la conservation des constats.

Enfin si cet utilisateur souhaite noter qu'une zone est vide alors chaque information qui était présente dans cette zone est susceptible de voir son état modifié (*RAS* + E).

constat / état actuel	V	$\bar{\text{V}}$	Zone Vide (*RAS*)	Commentaires
P	P	$\emptyset$ **ou** E	*RAS* **ou** *RAS* + E	**ou** : choix possible de l'utilisateur de conserver une trace après effacement **+** : superposition de deux informations différentes sur une même zone de la carte
E	P	E	*RAS*+ E	
?	P	E	*RAS*+E	

TAB. 6.1 – Modification locale d'une annotation lors d'un constat

6.7.2 Transitions simples et informations exclusives

Nous présentons dans cette section la logique de base que l'on peut associer aux informations. On notera que l'on peut généraliser l'utilisation des informations de surface dans l'application même lorsqu'on ne considère pas de réintroduction de verrous. Il suffit pour cela de ne pas considérer l'état **flou**. Rappelons qu'une information effacée sera archivée dans cet état. L'indéterminisme des figures présentées dans cette section provient du choix qui peut être fait lors de l'implémentation ou par un utilisateur de supprimer totalement ou de passer à l'état effacé une information jusqu'alors présente dans le système.

Information ponctuelle

La figure 6.5 présente les transitions possibles entre états pour une information ponctuelle. La transition v correspond ici à l'observation qui valide la véracité d'une information sur le terrain.

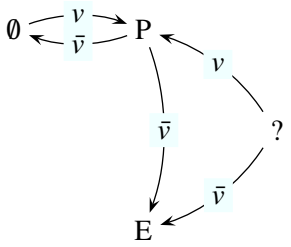


FIG. 6.5 – Transitions des états d'une information ponctuelle

Information de surface Exclusive (RAS)

La figure 6.6 présente les transitions possibles entre états pour une information de surface exclusive. La transition $\overline{ras}$ signifie que quelque chose a été observé et qu'on a appliqué une transition v pour une autre information. On observe sur la figure 6.7 que l'ajout d'une information ponctuelle dans une zone marquée (RAS) a pour effet de supprimer cette annotation pour ne conserver que l'information ponctuelle.

6.7.3 Synchronisation

Lorsque deux entités du réseau se synchronisent, chacune peut disposer d'une même information qui a pu être modifiée avec une version différente du verrou. Dans l'objectif de converger vers une version globale virtuelle unique nous avons défini des règles pour gérer les conflits qui peuvent exister. Le tableau 6.2 page 90 représente les modifications qui peuvent survenir sur l'état associé à une annotation lors d'une synchronisation. L'entité avec le verrou 1 dispose de la version

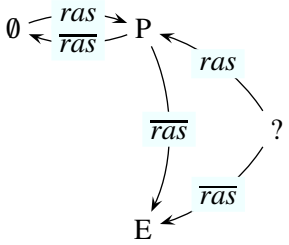


FIG. 6.6 – Transitions des états d'une information de surface exclusive (RAS)

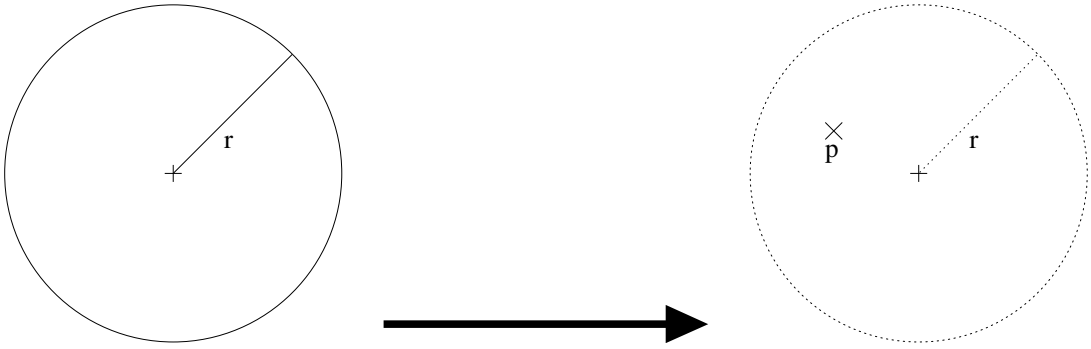


FIG. 6.7 – Insertion d'une information ponctuelle dans une zone marquée RAS

considérée comme obsolète ; la version 2 du verrou est considérée comme la version valide ou au moins plus récente que le verrou 1.

L'ensemble de ces transitions assure les propriétés suivantes :

- une information dans l'état effacé reste dans cet état et n'est jamais supprimée ;
- les constats effectués avec le verrou 2 sont considérés comme les plus récents même si cela peut être faux d'un point de vue chronologique, car c'est ce verrou qui est considéré comme valide par la hiérarchie ;
- on applique un principe de précaution, c'est-à-dire qu'à défaut de constat ($\emptyset$) avec le verrou 2, on conserve le maximum d'information de la version 1. Le seul changement qui est effectué intervient lorsqu'une information marquée P avec le verrou 1 n'existe pas dans l'autre version ($\emptyset$). Cette information est alors notée comme floue.

Dans le tableau chaque annotation est représentée sous la forme : $< Etat >_{<identifiant>}$. L'identifiant est la clé définie précédemment et il est de la forme $_{i1}$ pour l'information portant l'identifiant 1. La plupart des lignes du tableau ne prennent en compte qu'une seule information (un seul identifiant), sauf dans le cas où dans une même zone se trouvent une information ponctuelle et une information exclusive. Rappelons aussi que l'on souhaite assurer au mieux un principe de précaution en fonction des observations qui ont pu être faites et nous acceptons aussi de considérer une information peut-être invalide tout en spécifiant ce fait plutôt que d'ignorer un fait qui pourrait

avoir des conséquences graves. On préférera par exemple connaître la présence incertaine d'un tireur isolé ennemi, plutôt que de ne pas posséder cet avertissement. Nous rappelons que le verrou 2 est considéré comme valide par la hiérarchie, et que le verrou 1 est obsolète.

6.8 Conclusion

Dans ce chapitre nous avons proposé une solution pour pallier la perte de verrous qui peut être occasionnée par la disparition de certains utilisateurs. Notre méthode repose sur la prise en compte des spécificités du contexte d'utilisation de l'application, ici la hiérarchie militaire, qui dote certains utilisateurs de droits supplémentaires. Ainsi on peut choisir des nœuds spécifiques du système qui peuvent régénérer des verrous. Nous avons aussi défini la notion d'information **floue** qui permet de prendre en compte les informations résiduelles de versions invalides, ce qui garantit un principe de précaution. Enfin nous avons proposé d'assurer la sécurité d'un tel système grâce à l'emploi de cartes à puce. Ce dernier point fera l'objet de travaux futurs.

État de la donnée sur le nœud avec le verrou 1	État de la donnée sur le nœud avec le verrou 2	État de la donnée après synchronisation
P_{i1}	P_{i1}	P_{i1}
P_{i1}	E_{i1}	E_{i1}
P_{i1}	$?_{i1}$	$?_{i1}$
$\mathbf{P}_{i1}$	$\emptyset$	$\mathbf{?}_{i1}$
$\mathbf{P}_{i1}$	$\boldsymbol{RAS}_{i2}$	$\boldsymbol{RAS}_{i2}$ **+** $\mathbf{E}_{i1}$
E_{i1}	P_{i1}	P_{i1}
E_{i1}	E_{i1}	E_{i1}
E_{i1}	$?_{i1}$	$?_{i1}$
E_{i1}	$\emptyset$	E_{i1}
E_{i1}	RAS_{i2}	RAS_{i2} + E_{i1}
$?_{i1}$	P_{i1}	P_{i1}
$?_{i1}$	E_{i1}	E_{i1}
$?_{i1}$	$?_{i1}$	$?_{i1}$
$?_{i1}$	$\emptyset$	$?_{i1}$
$\mathbf{?}_{i1}$	$\boldsymbol{RAS}_{i2}$	$\boldsymbol{RAS}_{i2}$ **+** $\mathbf{E}_{i1}$
$\emptyset$	P_{i1}	P_{i1}
$\emptyset$	E_{i1}	E_{i1}
$\emptyset$	$?_{i1}$	$?_{i1}$
$\emptyset$	$\emptyset$	$\emptyset$
$\emptyset$	RAS_{i1}	RAS_{i1}
RAS_{i1}	P_{i2}	P_{i2}
RAS_{i1}	E_{i2}	$\mathbf{?}_{i1}$* + E_{i2}
RAS_{i1}	$?_{i2}$	$\mathbf{?}_{i1}$* + $?_{i2}$
$\boldsymbol{RAS}_{i1}$	$\emptyset$	$\mathbf{?}_{i1}$*
RAS_{i1}	RAS_{i1}	RAS_{i1}

*Ces points d'interrogation symbolisent comme dans le reste du tableau une information floue mais dans ce cas elle porte sur la totalité d'une zone RAS.

TAB. 6.2 – Synchronisation des états des données entre deux nœuds

Chapitre 7

Simulation

7.1 Introduction

Le système décrit précédemment qui a donné lieu à l'application de partage de cartes stratégiques Shaadhoc, a été implémentée au sein du simulateur Madhoc [80] pour évaluer son fonctionnement sur un réseau mobile composé d'un nombre de nœuds assez important.

L'utilisation de simulateurs de réseaux s'explique de par la difficulté de la mise en place de tests réels de grande ampleur, c'est-à-dire mettant en œuvre plus d'une dizaine de terminaux. La première difficulté est de réunir assez de nœuds (de l'ordre d'une centaine). Il faut pour cela obtenir des moyens financiers très important (de l'ordre de 50000 euros). De plus, si l'objectif de réunir une centaine d'entités mobiles était atteint, il resterait encore à rassembler un panel de « testeurs » équipés d'une station d'expérimentation. Il est aussi nécessaire d'équiper les utilisateurs d'un système permettant de tracer leurs déplacements pour exploiter les données relatives à l'application étudiée, en fonction de leur comportement au sein du champ d'opération. On pourra toutefois noter qu'une plate-forme de ce type sera mise en œuvre dans notre équipe de recherche à la rentrée 2008.

La simulation permet aussi de reproduire une expérience dans des conditions identiques afin d'extraire des statistiques. Par exemple, on peut maîtriser et reproduire certains paramètres du comportement des utilisateurs simulés, comme leur vitesse de déplacement ou leur probabilité de présence dans certains lieux spécifiques.

Enfin, l'utilisation d'un simulateur est inévitable si on considère le besoin de collecter des informations globales. En effet, lors de tests réels sur des réseaux MANets on ne peut pas facilement obtenir d'information globale sur le comportement d'une application. Les essais en simulation en revanche le permettent.

Le simulateur Madhoc que nous avons utilisé a été développé dans le cadre de la thèse de Luc Hogie [81] laquelle a notamment fait l'objet d'une collaboration européenne entre l'université du Havre sous la tutelle de Frédéric Guinand et de l'université du Luxembourg sous la tutelle de

Pascal Bouvry. La participation de l'équipe de Frédéric Guinand au projet ANR SARAH au sein duquel cette thèse s'inscrit a aussi motivé notre choix de Madhoc pour réaliser nos simulations, en plus de l'intérêt de l'outil en tant que tel.

7.2 Le Simulateur Madhoc

Madhoc est un simulateur de réseaux ad hoc et mobiles ad hoc (MANet). Il permet de tester des algorithmes ou des applications sur un grand nombre de nœuds tout en faisant varier différents paramètres comme :

- le nombre de nœuds simulés,
- la dimension de l'aire de simulation,
- le modèle de mobilité (Random Waypoint [82], Human Mobility [81]),
- les types de connexion utilisés (IEEE802.11b/g [1] ou Bluetooth [2]),
- la composition du réseau (PC portables, PDAs, smartphones).

On peut aussi intervenir pour préciser certaines grandeur relatives au modèle de mobilité comme les vitesses auxquelles les stations peuvent se déplacer ou leur portée radio. D'un point de vue réseau, Madhoc considère la couche MAC (Medium Access Control) et la couche physique en prenant en compte les collisions de paquets et les interférences radio. L'existence d'un lien de communication entre deux stations, c'est-à-dire la possibilité qu'elles s'échangent des messages, ne dépend ainsi pas uniquement de la distance euclidienne qui les sépare et de leur portée radio, mais aussi des phénomènes physiques sous-jacents. Comme beaucoup de simulateurs, Madhoc utilise la notion de temps discret qui consiste en une segmentation du temps d'exécution en pas. Ainsi il dissocie l'exécution du code simulé qui a lieu pendant un pas de temps, des calculs liés à la mobilité des nœuds et au passage des messages entre stations qui sont effectués entre ces pas.

On procède comme suit pour simuler une exécution avec Madhoc. Il faut dans un premier temps paramétrer le modèle de mobilité qui va être utilisé. Dans un deuxième temps, il faut développer le module applicatif qui va rendre possible la simulation d'une application spécifique. Madhoc est implémenté en Java et l'application ou l'algorithme simulé doit aussi être implémenté dans ce langage. Pour cela il est nécessaire d'étendre une classe Java spécifique et d'en surcharger la méthode principale, laquelle sera appelée à chaque fois que l'ordonnanceur permettra à une station (un terminal) d'exécuter du code. Madhoc permet de visualiser le champ d'opération simulé ainsi que le graphe de connectivité des nœuds du réseau (cf. figure 7.1). Cette interface permet de situer les stations et de suivre l'évolution de leur connectivité au cours de la simulation.

Alors que la plupart des simulateurs s'attachent à reproduire par le calcul finement les phénomènes liés aux couches physiques, Madhoc est basé sur l'utilisation de statistiques pour simuler (moins précisément) les couches basses. De ce fait, l'utilisateur dispose de plus de ressources de calcul sur la plate-forme de simulation pour exécuter des applications qu'avec l'emploi des simulateurs classiques. En effet, si des algorithmes de bas niveau peuvent être simulés avec peu de

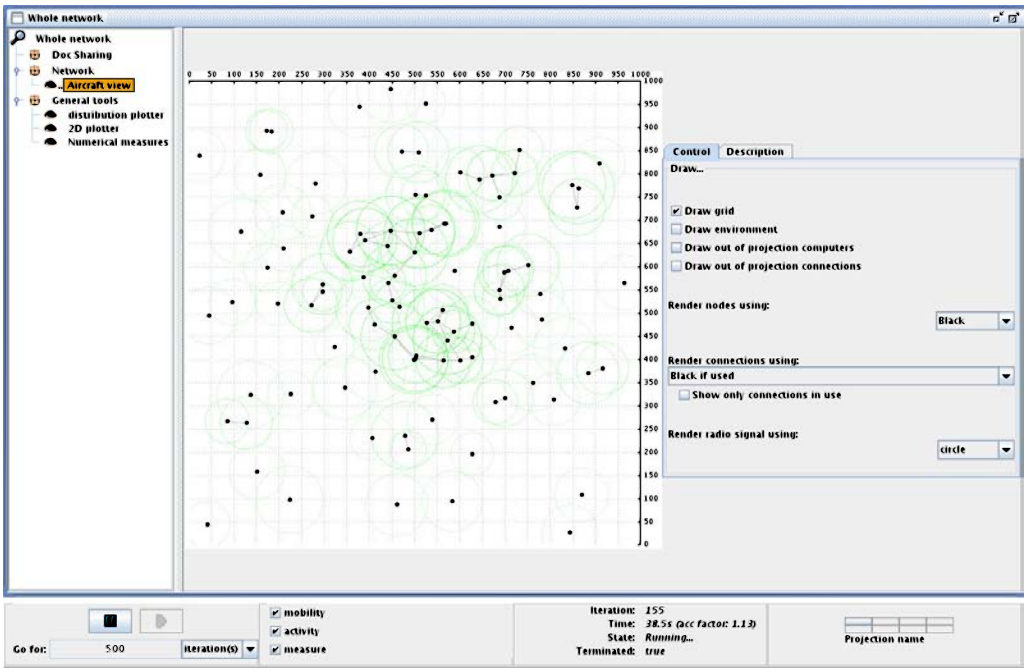


FIG. 7.1 – Interface graphique du simulateur Madhoc

ressources, il n'en est pas de même pour des applications de très haut niveau comme celles qui nous intéressent. Ainsi, nous acceptons d'avoir des résultats plus approximatifs pour la simulation de la couche physique pour pouvoir simuler notre application dans de bonnes conditions.

7.3 Deux autres simulateurs existants

Nous décrivons dans cette partie d'autres simulateurs de réseaux. L'objectif n'est pas ici de dresser une liste exhaustive mais plutôt de donner un aperçu des simulateurs les plus utilisés. Le lecteur est invité à se référer à la section 4.2.1 de la thèse de Luc Hogie [81] pour une étude comparative plus complète.

7.3.1 ns-2

ns-2 [83] est le plus connu et le plus utilisé des simulateurs de réseaux. Il est développé à l'ISI (*Information Sciences Institute*) en Californie (*Marina del Rey*). ns-2 est basé sur l'utilisation d'événements discrets et a été développé en premier lieu pour simuler des réseaux filaires. Les supports de WiFi et de Bluetooth n'ont été introduits que plus tard.

Ce simulateur est implémenté en C++ et se configure via des scripts en OTCL (langage dédié basé sur TCL développé au MIT). ns-2 s'attache à simuler précisément les couches physiques du réseau. Cette précision nécessite beaucoup de ressources de calcul et en fait un simulateur destiné

à des protocoles de bas niveau car il laisse très peu de ressources pour simuler des applications de haut niveau.

7.3.2 GloMoSim

GloMoSim [84] est un simulateur développé à l'université de Californie (UCLA) à Los Angeles. Il est écrit en Parsec, langage dédié basé sur le C développé à UCLA, et les protocoles à simuler doivent aussi être écrit dans ce langage. GloMoSim respecte le modèle OSI et associe un objet à chaque couche. Ce simulateur a la particularité d'exploiter les capacités des architectures SMP (*Shared Memory Processing*), la technique utilisée pour cela consistant à répartir les nœuds du réseau de façon homogène sur les processeurs.

Ce simulateur sert de base à QualNet [85] qui est un simulateur commercial.

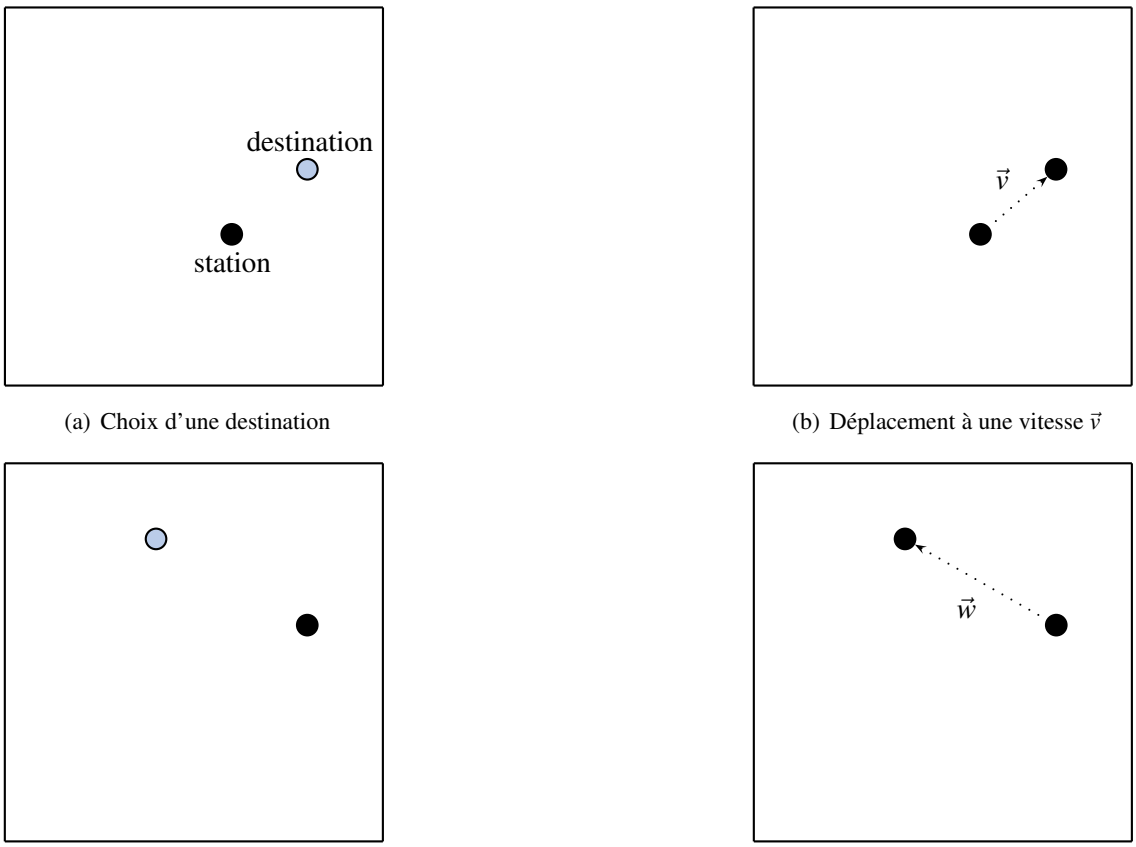
7.4 Les modèles de mobilité

L'évolution de la connectivité entre les nœuds du réseau est la caractéristique qui différencie le plus un réseau MANet d'un réseau statique. Cette évolution est généralement liée aux déplacements physiques des stations. Ils doivent donc être pris en compte pour que l'algorithme simulé évolue dans des conditions similaires à ce que l'on pourrait observer dans un contexte réel. Pour cela il existe différents modèles de mobilité qui définissent les changements des positions des nœuds du réseau. Ces modèles ont pour objectif de régir les déplacements des stations et de permettre de recréer les mêmes conditions expérimentales plusieurs fois.

7.4.1 Le modèle de mobilité Random WayPoint (RWP)

Le modèle *Random Waypoint* [82] est le plus connu des modèles de mobilité. Il est le standard de fait pour l'évaluation d'algorithmes dans les réseaux ad hoc et mobiles ad hoc. Son statut de standard est dû non seulement à son antériorité par rapport à d'autres modèles mais aussi à son extrême simplicité. L'algorithme qui régit le déplacement des stations dans le modèle *Random Waypoint* se base sur l'hypothèse que l'ensemble des stations évolue dans une aire de simulation bornée. Chaque station choisit (sans contrainte) une destination à l'intérieur de l'aire de simulation, ce qui se traduit en pratique par le choix aléatoire d'une coordonnée qui sera celle de la destination. Chaque station choisit ensuite une vitesse de déplacement dans l'intervalle des vitesses admises par la simulation. Elle se déplace et lorsque la destination est atteinte, elle s'arrête pendant un certain temps, puis choisit une nouvelle destination. La figure 7.2 page ci-contre représente le déplacement d'une station régi par le *Random Waypoint*.

Malgré son incontestable popularité, il paraît assez évident que cet algorithme ne correspond pas à un déplacement humain réaliste. En effet, dans ce modèle, le déplacement des stations n'est aucunement affecté ni par l'environnement (obstacles, points d'intérêt, ...) ni par la proximité et



(a) Choix d'une destination

(b) Déplacement à une vitesse $\vec{v}$

(c) Après un temps de pause, choix d'une autre destination

(d) Déplacement à une vitesse $\vec{w}$

FIG. 7.2 – Déplacement d'une station obéissant au *Random Waypoint*

les interactions éventuelles avec d'autres terminaux. Cependant, le *Random Waypoint*, même s'il ne correspond pas à des situations réalistes, est souvent utilisé pour comparer des algorithmes. On retrouve notamment l'emploi de ce modèle dans des évaluations comparatives d'algorithmes de diffusion d'information. Pourtant il est clair que l'utilisation de ce modèle de mobilité très générique ne permet pas d'anticiper les performances d'un algorithme dans des situations réelles. Il n'est pas évident non plus que la comparaison faite lors de la simulation soit très significative.

7.4.2 Le modèle de mobilité Human Mobility

Bien que le *Random Waypoint* soit reconnu par la communauté comme modèle de mobilité de référence pour réaliser des mesures comparatives, d'autres modèles plus proches de la réalité ont été définis ces dernières années.

Le modèle de mobilité humaine, présenté dans la thèse de Luc Hogie [81], est un exemple de ces résultats. Ce modèle générique représente le mouvement d'une population dans une ville. Il prend en compte différentes voies de communication et des points d'intérêt, comme des commerces (cf figure 7.3 page suivante). La population (les utilisateurs) se déplace de point d'intérêt en point d'intérêt avec une certaine probabilité d'aller vers celui le plus proche du dernier visité. On définit différents intervalles de vitesse de déplacement pour l'intérieur de ces zones et entre celles-ci.

Nous avons utilisé ce modèle lors de nos simulation avec Madhoc. Ce choix a été guidé par la volonté d'avoir une vision plus réaliste que celle obtenue avec le *Random Waypoint*. Cette décision a aussi été influencée par la volonté d'utiliser un modèle plus proche du déplacement des militaires sur le terrain, lesquels ont des points de passage de référence.

7.4.3 D'autres modèles de mobilité

On trouve de nombreux autres modèles de déplacements aléatoires. Pour certains d'entre eux la priorité est d'obtenir des déplacements les plus imprévisibles possibles. Parmi ceux-ci on peut citer le *Random Walk* [86]. Ce modèle pousse le raisonnement jusqu'à choisir une direction aléatoirement à chaque pas de calcul. Le résultat donne généralement un système avec un certain immobilisme dû à la probabilité de choisir une direction opposée à celle choisie précédemment. Une variante de cet algorithme appelée *Random Direction* [86] permet aux stations de choisir aléatoirement un vecteur vitesse puis d'atteindre une limite de l'aire de simulation à vitesse constante avant de choisir aléatoirement un autre vecteur.

Nous décrivons maintenant d'autres modèles que l'on retrouve régulièrement dans la littérature (notamment dans l'article de synthèse [86] auquel le lecteur pourra se référer) mais qui restent moins populaires que ceux cités précédemment. Les modèles de types urbains sont une famille de modèles de mobilité qui sont destinés à la représentation des déplacements au sein d'une ville. Le modèle *City Section* représente le déplacement d'une population dans les rues d'une ville. Il

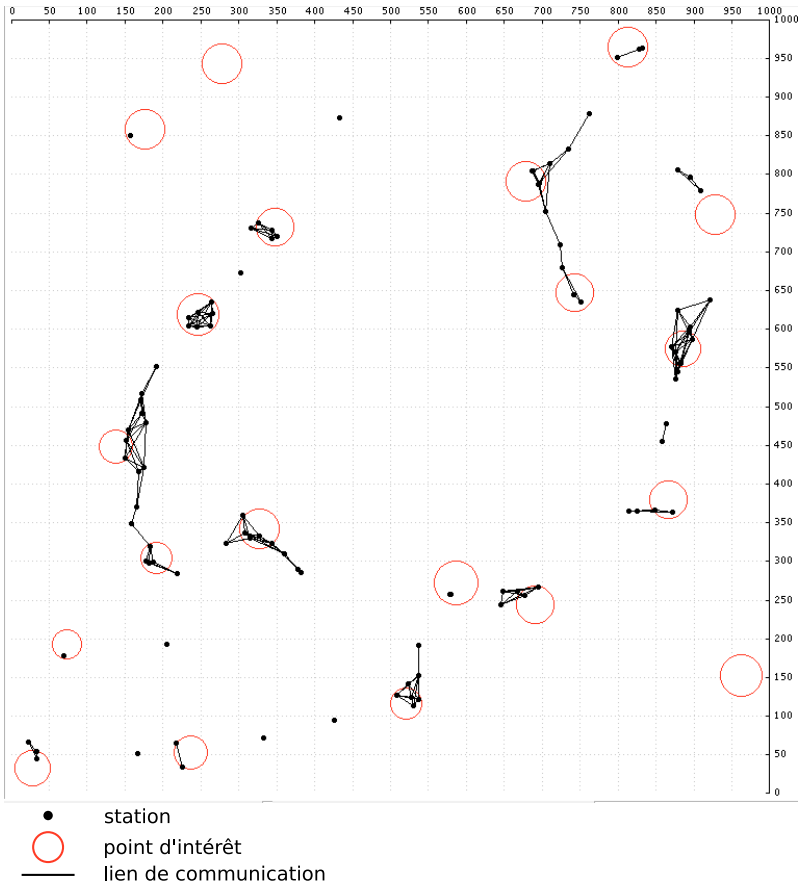


FIG. 7.3 – Simulation du modèle de mobilité humain dans MadHoc

est proche du *Random Waypoint* d'un point de vue algorithmique puisque chaque entité est placée sur un point d'une rue, choisit une destination dans le réseau routier puis s'y rend en respectant des vitesses associées aux types de rue. Dans le modèle *Manhattan* on se place dans un contexte très proche du modèle précédent. La différence est que la disposition des rues est semblable à celle des villes américaines, c'est à dire qu'elle peuvent être représentées sur une grille. A chaque intersection on a une probabilité de 0,5 de continuer tout droit et de 0,25 de tourner d'un côté ou de l'autre.

Les modèles de mobilité pour les groupes sont destinés à la représentation du déplacement de plusieurs stations simultanément et définissent des corrélations entre leurs mouvements respectifs. Le modèle *Nomadic Community* est comparable au *Random Waypoint* mais appliqué à des groupes. Le *Reference Point Group Mobility* (RPGM) associe un déplacement aléatoire au groupe et à chaque entité qui le compose. On définit un centre logique à chaque groupe pour lequel on calcule un déplacement. Puis chaque station choisit son propre déplacement aléatoirement comme pour le *Random Waypoint*. On définit ainsi un vecteur vitesse pour le centre du groupe et un autre vecteur vitesse (de norme plus petite) pour chaque station du groupe. Les entités se déplacent finalement suivant la somme de ces deux vecteurs.

7.5 Codage de Shaadhoc dans Madhoc

Si les simulateurs de réseaux sont bien adaptés au développement de nouveaux protocoles de bas niveau, il ne sont généralement pas conçus pour évaluer des applications réelles. En effet, les simulateurs fonctionnant en temps discret, les applications doivent s'adapter à ce modèle. Les applications classiques fonctionnent normalement en temps continu et le découpage des tâches qu'elles accomplissent en pas de calcul n'est pas toujours trivial. Il faut donc bien souvent réimplémenter chaque programme spécifiquement pour un simulateur donné. De plus, dans le cas de la simulation d'applications réelles qui n'ont d'intérêt que s'il y a une intervention régulière des utilisateurs, le simulateur doit se substituer à ces interventions externes. Nous détaillerons donc dans cette section comment l'application de partage de document décrite chapitre 5 a été adaptée à la discrétisation du temps et comment le comportement des utilisateurs a été simulé.

7.5.1 Adaptation de Shaadhoc à la discrétisation du temps

Au cours de la simulation, chaque station peut réaliser deux types d'actions qui sont soit de modifier sa version du document soit de choisir un voisin et de se synchroniser avec lui (au sens de l'algorithmique du 5.3.2 page 71). La première action (qui sera détaillée dans la section suivante) relève du comportement de l'utilisateur. La deuxième action, la synchronisation, utilise des communications réseau. Madhoc fonctionnant en temps discret, l'opération de synchronisation des versions doit donc être décomposée en plusieurs phases. Avant que deux nœuds puissent se synchroniser, ceux-ci doivent s'être choisis comme partenaires. Ce choix est appelé rendez-vous. Ce

mécanisme doit lui aussi être décomposé en phases. Nous présentons ci-dessous la décomposition que nous avons réalisée pour notre application :

1. Modification locale éventuelle du document (Simulation du comportement humain)
2. Envoi d'une demande de rendez-vous à un voisin
3. Vérification de la réciprocité de la demande de rendez-vous de la part de ce voisin
4. Envoi du document
5. Réception du document distant
6. Envoi d'un accusé de réception
7. Réception de l'accusé de réception distant
8. Modification du document local

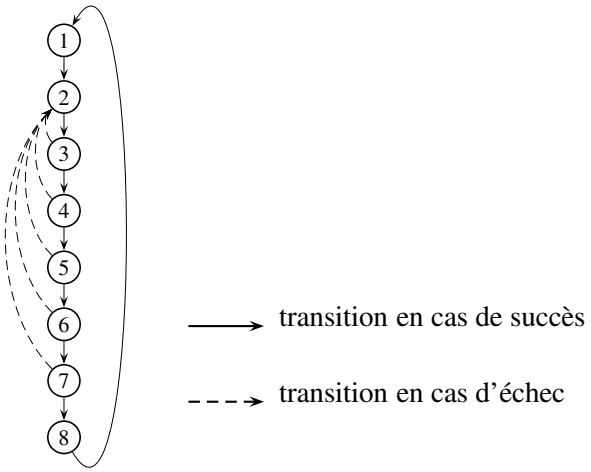


FIG. 7.4 – Transitions conditionnelles entre les étapes de simulation

Le passage d'une étape à la suivante n'a lieu qu'en cas de succès, ce qui est représenté par la figure 7.4. Le découpage présenté n'est pas optimal car certaines étapes peuvent être exécutées pendant le même pas de simulation. La seule contrainte du simulateur est qu'il nécessite un pas de calcul pour que les messages envoyés puissent être reçus. Par exemple une phase nécessitant une réception peut être directement suivie d'une phase procédant à une émission de message. On pourrait donc regrouper certaines opérations pour les effectuer en une seule étape. Seules les étapes 2, 4 et 6 procèdent à l'envoi d'un message ; on peut donc faire les regroupements suivants :

1. étapes 1 et 2
2. étapes 3 et 4
3. étapes 5 et 6
4. étapes 7 et 8

Ces regroupements peuvent être effectués en une seule étape de temps chacun.

La notion de rendez-vous. Un rendez-vous est l'opération qui permet à deux entités du réseau de se synchroniser sur un pas de calcul pour procéder à l'exécution en parallèle d'un algorithme distribué de façon synchrone. Dans notre exemple l'algorithme qui nécessite d'être exécuté de cette façon est la synchronisation (de versions) entre deux nœuds pendant laquelle chaque requête envoyée par une station nécessite une réponse de l'autre station pour se poursuivre.

Après que deux stations se sont synchronisées, elles peuvent exécuter en parallèle chacune des étapes que nous avons décrit précédemment. Les différentes stratégies pour obtenir un rendez-vous avec un voisin seront discutées dans la résultat section 7.6 intitulée résultats.

7.5.2 Simulation du comportement des utilisateurs

La deuxième adaptation importante de l'application concerne l'interface utilisateur. Il n'est en effet pas envisageable de gérer une interface graphique par station dans un contexte où le nombre de stations avoisine la centaine. Le comportement de chaque utilisateur a donc été simulé au sein de la phase qui consiste à modifier la copie locale du document (phase 1 de la figure 7.4). Différentes actions ont été définies en fonction de l'état initial de chaque feuille de l'arbre représentant le document. La simulation du comportement utilisateur consiste alors à attribuer des probabilités d'occurrence à ces actions :

- La station dispose du verrou (état $\boxed{CLOSED}$) sur la feuille de l'arbre, alors on a :
  1. une probabilité P_1 de procéder à un découpage,
  2. une probabilité P_2 de procéder à une modification du document puis de relâcher le verrou pour permettre une meilleure circulation de celui-ci,
  3. une probabilité P_3 de relâcher le verrou ;
- si la station dispose de la feuille de l'arbre dans l'état $\boxed{UNDESIRED}$ alors on a une probabilité P_4 de passer dans l'état $\boxed{REQUESTED}$.

Les valeurs de probabilité que nous avons choisies pour nos tests ont été définies de façon empirique. Nous nous assurons simplement que la somme est inférieure à 1 pour ne pas toujours faire de modification. Les valeurs choisies donnent des évolutions du document assez rapides et permettent aux verrous de circuler correctement.

L'opération de fusion n'est pas traitée au cours de la simulation dans le but d'abaisser le niveau de complexité. De même, le document partagé ici est constitué d'un texte afin d'en simplifier la génération aléatoire et de réduire l'espace mémoire nécessaire pour simuler une centaine de stations. Nous avons présenté le partage d'un document texte basé sur une plate-forme différente lors de MobiWac 2006 [75]. La seule différence qu'il nous faut signaler ici est que dans le cas d'un texte nous utilisons un arbre binaire au lieu d'un arbre quaternaire pour une carte. Rappelons que le système que nous présentons dans cette partie est générique et qu'il peut s'appliquer directement à tout document porté par un arbre n-aire.

7.6 Résultats

L'objectif de la simulation est d'évaluer et d'observer le comportement de notre application sur un très grand nombre de stations. C'est pourquoi les simulations sont incontournables. Les différents modèles de mobilité que l'on peut mettre en œuvre permettent de faire varier le contexte d'exécution et d'observer en quoi le comportement de l'application peut être plus efficace ou différent. Dans un premier temps nous avons procédé à l'évaluation qualitative de l'application qui a essentiellement consisté à vérifier la cohérence des états au cours des différentes exécutions. Après cette évaluation qualitative expérimentale, nous avons évalué quantitativement l'impact de certains paramètres comme les critères de choix des voisins ou l'influence du modèle de mobilité.

7.6.1 Validation qualitative

Dans un premier temps la simulation permet de tester le bon fonctionnement de notre application. Pour cela nous vérifions que chaque synchronisation lorsqu'elle est effectuée, l'est correctement par les deux stations qui ont pris un rendez-vous. Pour cela on génère une matrice d'adjacence contenant le nombre de fois où les entités se sont synchronisées entre elles puis on contrôle en permanence que cette matrice est symétrique. Ainsi on vérifie que si une entité A a appliqué une synchronisation avec une entité B alors la réciproque est vraie. On s'assure aussi qu'une seule entité dispose du verrou sur une même partie du document à la fois.

7.6.2 Validation quantitative

Modèles de mobilité

Nous avons utilisé deux des modèles de mobilité présentés précédemment, à savoir *Random Waypoint* et *Human Mobility*. Le premier a été choisi car il est incontournable et presque inconcevable aujourd'hui de faire une simulation de réseaux mobiles sans l'employer. Le modèle *Human Mobility* a lui été choisi car il nous a semblé être plus proche de situations réelles.

Stratégies de rendez-vous

Le rendez-vous repose en grande partie sur la façon de choisir un voisin avec lequel communiquer. Rappelons que l'objectif est que chaque station ait la version la plus à jour possible du document en fonction des autres stations qu'elle aura rencontrées. Ainsi, nous souhaitons, lorsque plusieurs voisins sont disponibles, pouvoir choisir d'une manière très simple le plus approprié avec lequel se synchroniser pour atteindre ce but.

Pour cela nous affectons un numéro de version unique au document qui doit être assez représentatif de la version portée par une station. L'algorithme pour déterminer ce numéro de version global consiste simplement à associer à un nœud de l'arbre un numéro de version global qui est son propre numéro de version additionné de la somme des numéros de version de ses fils. Par exemple,

le résultat de cet algorithme appliqué à l'arbre de la figure 7.5 donne 39. Cette valeur n'est pas unique dans le réseau. En revanche, cet algorithme donne le plus grand numéro de version global au document hypothétique correspondant à la synchronisation de tous les documents du réseau et qui correspond au document le plus récent que chaque station souhaite reconstituer.

Lors des simulations nous imposons aux stations d'arrêter de modifier leur document au delà d'un certain numéro de version global (phase 1 de la figure 7.4 page 99).

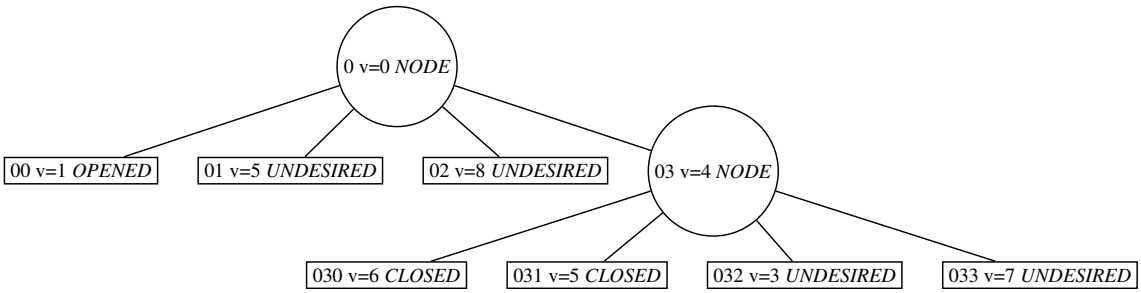


FIG. 7.5 – Exemple d'arbre représentant un document

La première stratégie de rendez-vous est aléatoire. Le voisin à qui est envoyé le message de demande de rendez-vous est choisi au hasard. Cette méthode est celle généralement utilisée en algorithmique distribuée [87] car elle est la plus générique.

Dans notre cas, le but du rendez-vous est de synchroniser les versions des documents de deux stations. Nous avons donc choisi de mesurer l'apport d'une deuxième méthode basée sur des informations contextuelles en procédant à des synchronisations entre des voisins qui ont le plus grand écart entre leurs numéros de version. Cette méthode devrait *a priori* permettre d'améliorer la diffusion des versions les plus récentes lors des synchronisations en réduisant globalement le plus possible l'écart entre les versions présentes. Concrètement, chaque nœud diffuse périodiquement un message d'annonce qui contient le numéro de version de son document. Chaque voisin qui reçoit ce message, enregistre la valeur qu'il contient et l'utilise lors de son propre choix d'un voisin.

Enfin, dans un troisième temps, plutôt que d'envoyer systématiquement une demande de rendez-vous nous verrons ce qu'apporte le fait de consulter les messages reçus pour trouver une demande de rendez-vous émise à l'étape précédente par une autre station. Ceci permet de considérer les requêtes qui peuvent avoir été envoyées au pas précédent par un voisin décalé d'une étape dans l'algorithme. Cette stratégie sera nommée dans la suite utilisation d'une file d'attente.

7.6.3 Mesures

Dans cette section nous présentons différentes mesures qui sont le résultat de la combinaison des différentes stratégies de rendez-vous avec les deux modèles de mobilité différents (*Random Waypoint* et *Human mobility*). Chaque graphique est le résultat de la moyenne de dix simulations successives dans la même configuration. Cette étude nous permet de voir l'évolution dans le temps

du numéro de version du document, défini dans la section précédente, en fonction des différentes stratégies de rendez-vous ou des différents modèles de mobilité utilisés. Cette évolution mesure l'apport des différentes stratégies employées. Concrètement c'est la convergence vers une version unique qui nous intéresse, plus elle sera rapide plus nous considérerons le résultat bon.

Le réseau simulé est composé de cents stations ayant des portées radio variant de quarante à quatre-vingt mètres réparties sur une aire de simulation d'un kilomètre carré. Les utilisateurs (simulés eux aussi) ont pour consigne de ne plus modifier le document dont ils disposent dès lors que son numéro de version dépasse 35. En effet, dans l'objectif de comparer différents scénarios nous fixons ce paramètre afin d'observer une phase de convergence vers la version la plus à jour possible pour toutes les stations. Lorsque ce numéro de version est atteint par le document d'une station, elle continue tout de même à se synchroniser avec les autres. Le numéro de version final sera donc supérieur ou égal à 35.

Les différents graphiques présentés dans cette section résultent de la combinaison de trois paramètres :

1. le modèle de mobilité : *Random Waypoint* ou *Human Mobility*. La configuration utilisée pour nos tests fait varier les vitesses des stations : de 4 à 10 m/s dans le cadre de l'utilisation du *Random Waypoint* ; de 1 à 5 m/s à l'intérieur d'un point d'intérêt et de 2 à 20 m/s à l'extérieur avec le modèle *Human Mobility*. Ces vitesses élevées permettent de simuler à la fois le mouvement de piétons etcelui de véhicules.
2. la sélection du numéro de version des voisins : avec ou sans ;
3. la prise en compte d'une demande de synchronisation dans la file de réception : avec ou sans.

Les graphiques présentés dans cette section montrent dans différentes configurations l'évolution des paramètres suivants :

- sur l'axe des ordonnées de gauche :
  - la version moyenne du document ;
  - l'écart type du numéro de version ;
  - le numéro de version maximal présent sur une station ;
  - le nombre de stations disposant d'un numéro de version égal au maximum ;
- sur l'axe des ordonnées de droite :
  - le nombre total de synchronisations (axes des ordonnées de droite).

On s'attachera principalement à observer la progression du nombre de nœuds disposant du document avec le plus grand numéro de version et, après l'arrêt de l'incrémentation de ce numéro, à la phase de convergence vers une version unique.

Le tableau 7.1 page suivante récapitule l'association entre ces paramètres et les graphes présentés dans la suite de ce document.

Paramètres / Graphique	modèle de mobilité	gestion du numéro de version	file d'attente
fig. 7.6	*RWP*		
fig. 7.7	*RWP*		X
fig. 7.8	*RWP*	X	
fig. 7.9	*RWP*	X	X
fig. 7.10	*HM*		
fig. 7.11	*HM*		X
fig. 7.12	*HM*	X	
fig. 7.13	*HM*	X	X

TAB. 7.1 – Correspondance entre la variation des paramètres et les graphiques associées

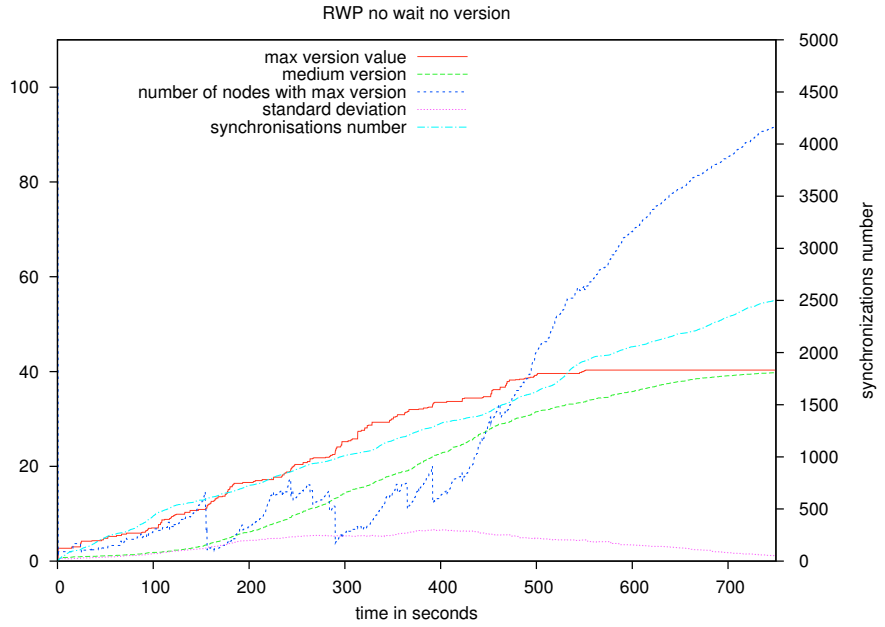


FIG. 7.6 – Résultat de simulation utilisant le *Random Waypoint* sans gestion des versions et sans file d'attente

Commentaires des graphiques

Figure 7.6. Le modèle de mobilité utilisé est le *Random Waypoint* sans file d'attente et sans gestion du numéro de version pour la prise de rendez-vous. On atteint un nombre de synchronisations d'environ 2500. La progression des mises à jour sera notre référence pour les tests suivants et la version maximum est atteinte en environ 550 secondes. Ce résultat est un résultat de référence car

il est celui qui utilise l'algorithme de rendez-vous le plus simple et le modèle de mobilité le plus répandu.

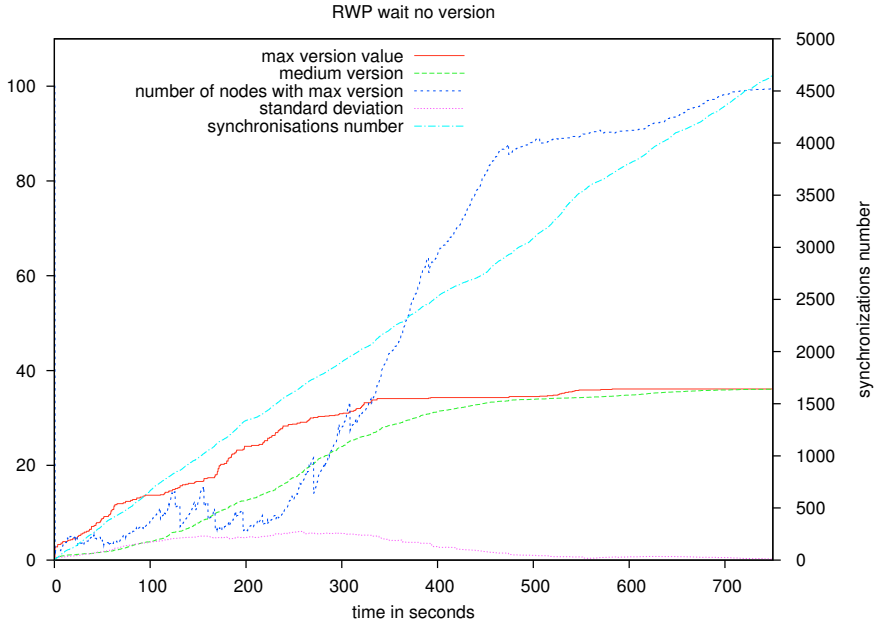


FIG. 7.7 – Résultat de simulation utilisant le *Random Waypoint* sans gestion des versions et avec file d'attente

Figure 7.7. Le modèle de mobilité utilisé est le *Random Waypoint* avec file d'attente et sans gestion du numéro de version pour la prise de rendez-vous. On atteint un nombre de synchronisations d'environ 4600. La progression des mises à jour est très rapide et la version maximum est atteinte en environ 550 secondes. Ce graphique montre une convergence plus rapide que pour la figure 7.6, ce qui s'explique par le nombre beaucoup plus important de synchronisations entre les nœuds.

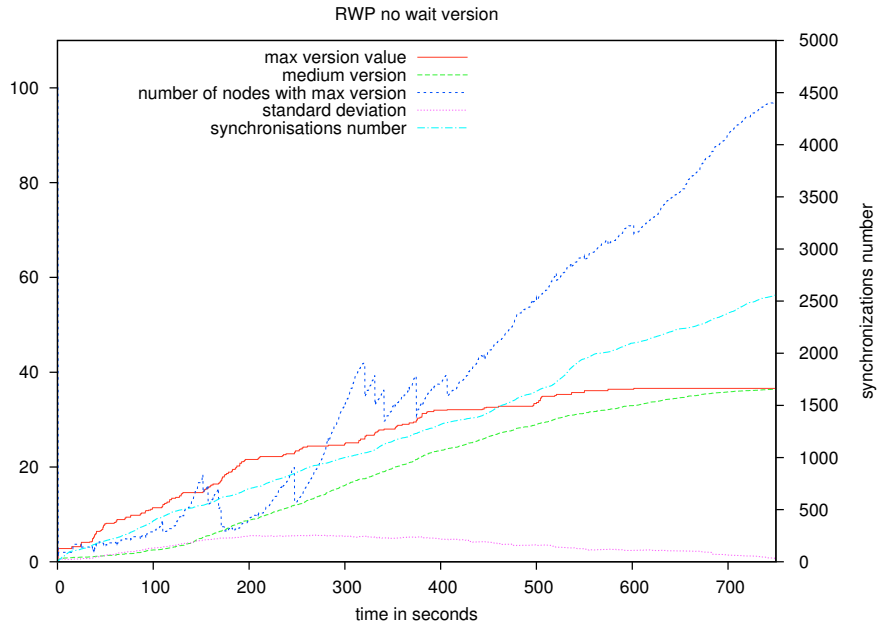


FIG. 7.8 – Résultat de simulation utilisant le *Random Waypoint* avec gestion des versions et sans file d'attente

Figure 7.8. Le modèle de mobilité utilisé est le *Random Waypoint* sans file d'attente et avec gestion du numéro de version pour la prise de rendez-vous. On atteint un nombre de synchronisations d'environ 2500. La progression des mises à jour est plutôt rapide et la version maximum est atteinte en environ 600 secondes. Ces résultats sont très comparables à ceux de la version de référence, on ne remarque pas ici de différence majeure en terme de performance.

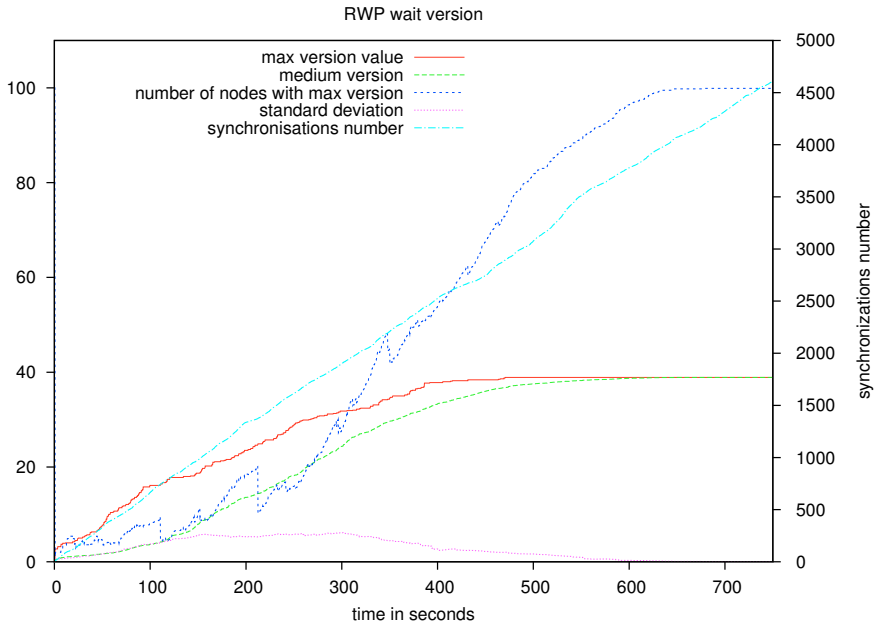


FIG. 7.9 – Résultat de simulation utilisant le *Random Waypoint* avec gestion des versions et avec file d'attente

Figure 7.9. Le modèle de mobilité utilisé est le *Random Waypoint* avec file d'attente et avec gestion du numéro de version pour la prise de rendez-vous. On atteint un nombre de synchronisations d'environ 4600. La progression des mises à jour est très rapide et la version maximum est atteinte en environ 450 secondes. Ce résultat est le plus efficace en ce qui concerne l'utilisation du *Random Waypoint*. Ceci confirme les hypothèses de départ sur l'apport de la prise en compte du numéro de version et d'une demande de synchronisation antérieure.

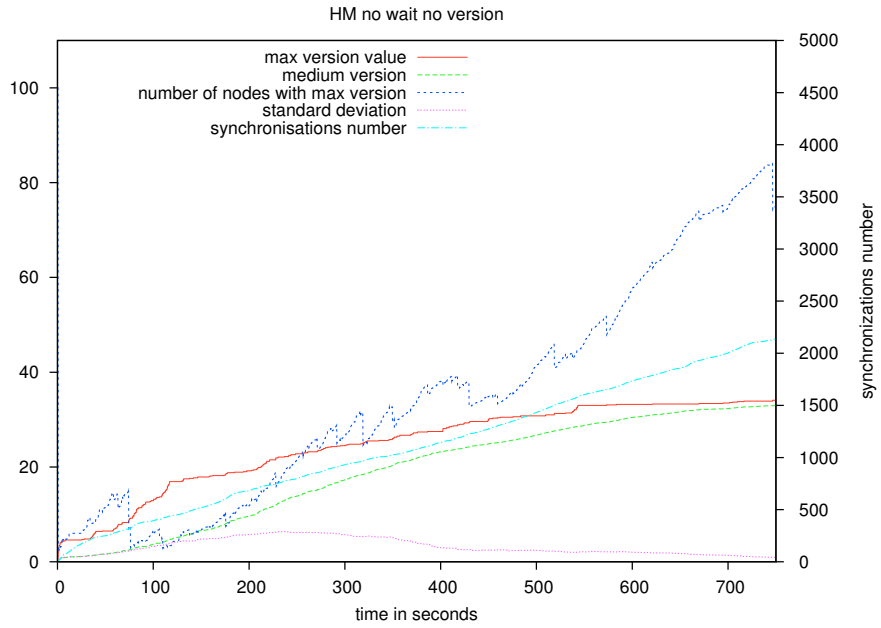


FIG. 7.10 – Résultat de simulation utilisant le *Human Mobility* sans gestion des versions et sans file d'attente

Figure 7.10. Le modèle de mobilité utilisé est le *Human Mobility* sans file d'attente et sans gestion du numéro de version pour la prise de rendez-vous. On atteint un nombre de synchronisations d'environ 2100. La progression des mises à jour est lente et la version maximum est quasiment atteinte en 700 secondes, qui est la durée d'une simulation. Ces performances sont moins bonne que dans la même configuration pour le *Random Waypoint*, ils serviront de référence pour le modèle *Human Mobility*.

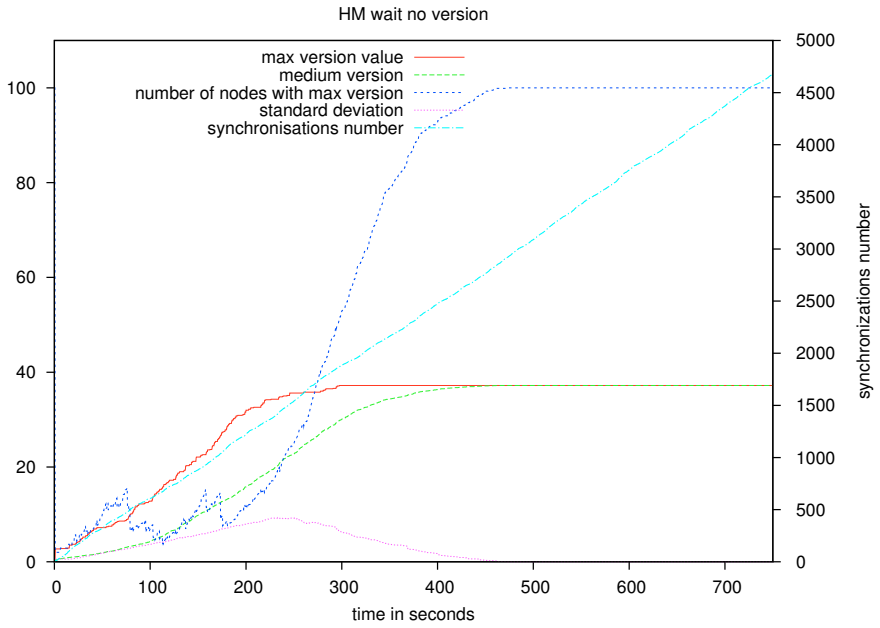


FIG. 7.11 – Résultat de simulation utilisant le *Human Mobility* sans gestion des versions et avec file d'attente

Figure 7.11. Le modèle de mobilité utilisé est le *Human Mobility* avec file d'attente et sans gestion du numéro de version pour la prise de rendez-vous. On atteint un nombre de synchronisations d'environ 4700. La progression des mises à jour est extrêmement rapide et la version maximum est atteinte en environ 300 secondes. Dans ce test aussi l'utilisation de la file d'attente permet de doubler le nombre de synchronisations et donc la vitesse de progression.

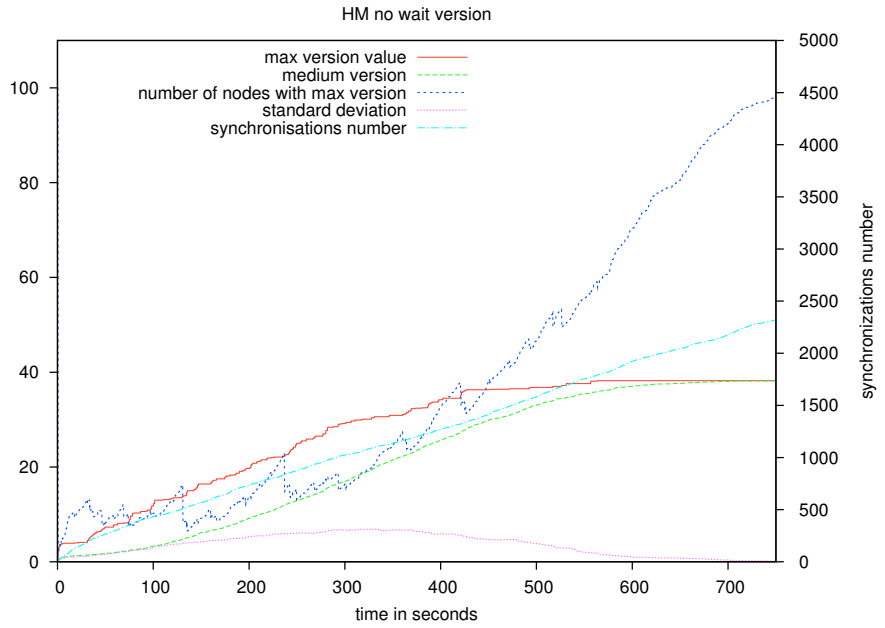


FIG. 7.12 – Résultat de simulation utilisant le *Human Mobility* avec gestion des versions et sans file d'attente

figure 7.12. Le modèle de mobilité utilisé est le *Human Mobility* sans file d'attente et avec gestion du numéro de version pour la prise de rendez-vous. On atteint un nombre de synchronisations d'environ 2400. La progression des mises à jour est plutôt lente et la version maximum est atteinte en environ 550 secondes. Le résultat est ici un peu meilleur que pour le test de référence (7.10) grâce à la considération des numéros de version des documents.

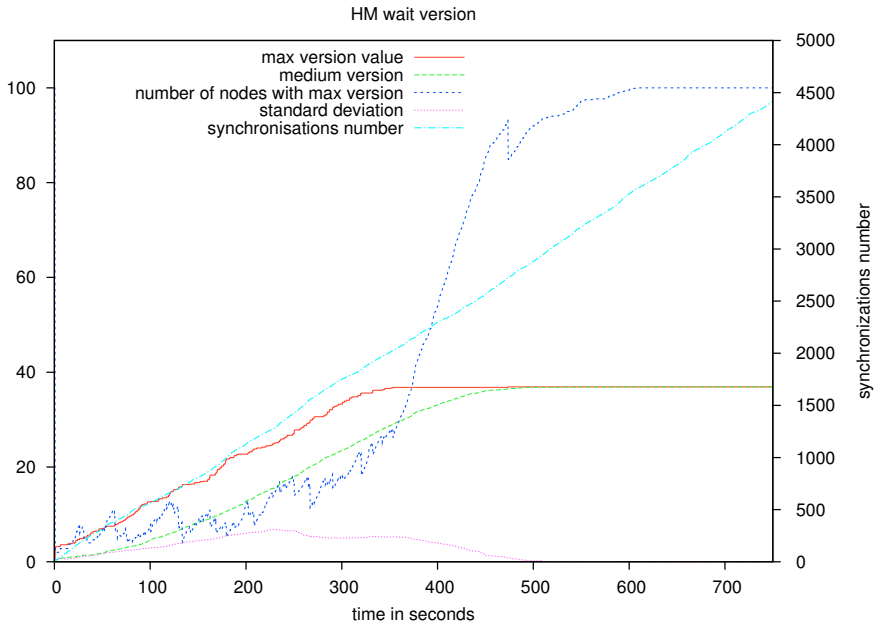


FIG. 7.13 – Résultat de simulation utilisant le *Human Mobility* avec gestion des versions et avec file d'attente

Figure 7.13. Le modèle de mobilité utilisé est le *Human Mobility* avec file d'attente et sans gestion du numéro de version pour la prise de rendez-vous. On atteint un nombre de synchronisations d'environ 4400. La progression des mises à jour est très rapide et la version maximum est atteinte en environ 350 secondes. Ce résultat est le meilleur pour le modèle *Human Mobility*.

On peut regrouper ces résultats en deux groupes distincts qui sont ceux qui utilisent une file d'attente de demande (à un pas de calcul) de rendez-vous et les autres. Avec cette file on compte un nombre total de synchronisations supérieurs à 4000 voire 4500 alors que sans cette file d'attente ce nombre est de l'ordre de 2500.

On remarque par contre que l'apport de la prise en compte du numéro de version des voisins a peu d'intérêt dans nos simulations. L'effet produit est assez difficile à percevoir car il améliore à peine les performances dans le cadre du modèle *Random Waypoint* et produit des résultats moins bons avec *Human Mobility*. Ceci peut s'expliquer par la densité assez faible de nœuds dans notre réseau qui est de 100 nœuds pour 1 kilomètre carré. De plus, la prise en compte des versions des voisins n'est un atout que dans des composantes quasi complètes (au sens des graphes) car son apport dépend de la réciprocité du choix du voisin. Dans une composante complète d'un réseau si une station A choisit une station B, et si il n'y pas d'autre entité avec le même numéro de version que A, alors B choisira A. Mais si les voisinages de A et de B sont différents, cette réciprocité est beaucoup moins probable. Enfin, les mesures produites par les simulations montrent clairement que le modèle de mobilité (ici *Random Waypoint* et *Human Mobility*) a peu d'influence sur le comportement de cette application.

7.7 Conclusion

La simulation de l'application a permis de valider son comportement à l'échelle d'une centaine de nœuds. La cohérence de notre système est conservée à chaque instant et n'est pas influencée par le nombre de stations. Ceci était prévisible car notre application n'utilise que des communications deux à deux. On peut aussi noter le peu d'influence des modèles de mobilité sur les résultats présentés. Cependant, il serait intéressant de définir un modèle clairement destiné au déplacement des militaires sur le terrain pour voir si l'on peut isoler un comportement particulier. Ainsi peut-être pourrait-on optimiser les applications en fonction d'un tel modèle.

Au niveau des stratégies de rendez-vous, on pouvait penser que la sélection du voisin ayant le plus grand écart de version aurait été plus efficace. Nous pensons que le faible degré des graphes sous-jacents aux réseaux simulés explique en partie ce phénomène. On remarque par contre que la prise en compte des demandes antérieures de synchronisation a un effet très significatif et positif.

Chapitre 8

Mise en œuvre de Shaadhoc

Dans ce chapitre nous exposons certains points clés de l'implémentation de l'application Shaadhoc. Dans un premier temps nous ferons une brève description de son architecture. Nous traiterons ensuite la question de la découverte du voisinage dans les réseaux MANets pour laquelle nous verrons des méthodes adaptées à des technologies de communication comme WiFi (IP) [1] ou Bluetooth [2].

Un des points clés de Shaadhoc est la synchronisation des documents entre deux utilisateurs. Nous présenterons comment ce protocole fonctionne concrètement et comment il a été implémenté.

Enfin, pour aider à comprendre ce qu'est la version fonctionnelle de Shaadhoc, nous illustrerons certaines manipulations possibles au travers d'images issues de l'interface homme machine que l'on retrouve sur la version PDA. Celle-ci provient d'un portage de l'application en *C#* [88] effectué grâce à un contrat d'ingénieur obtenu dans le cadre de la labellisation Carnot du LaBRI. La plate-forme sur laquelle a été déployée l'application compte moins d'une dizaine de nœuds. Des tests à plus grande échelle pourront être effectuer grâce à l'aquisition d'environ soixante terminaux mobiles.

8.1 Architecture logicielle

L'application Shaadhoc a tout d'abord été implémentée en Java sur une plate-forme de type PC classiques utilisant une connexion réseau WiFi. C'est cette version que nous présentons ici.

La figure 8.1 page suivante représente l'architecture du cœur de l'application (sans son interface graphique). Cette architecture se compose de quatre parties qui sont la gestion du document partagé, la gestion des protocoles propres à l'application, la gestion de la découverte du voisinage, et le formatage du document pour pouvoir l'envoyer sur le réseau (sérialisation Java [89] ou XML [90]).

La gestion locale du document est organisée autour de la classe *Tree* qui gère la structure de

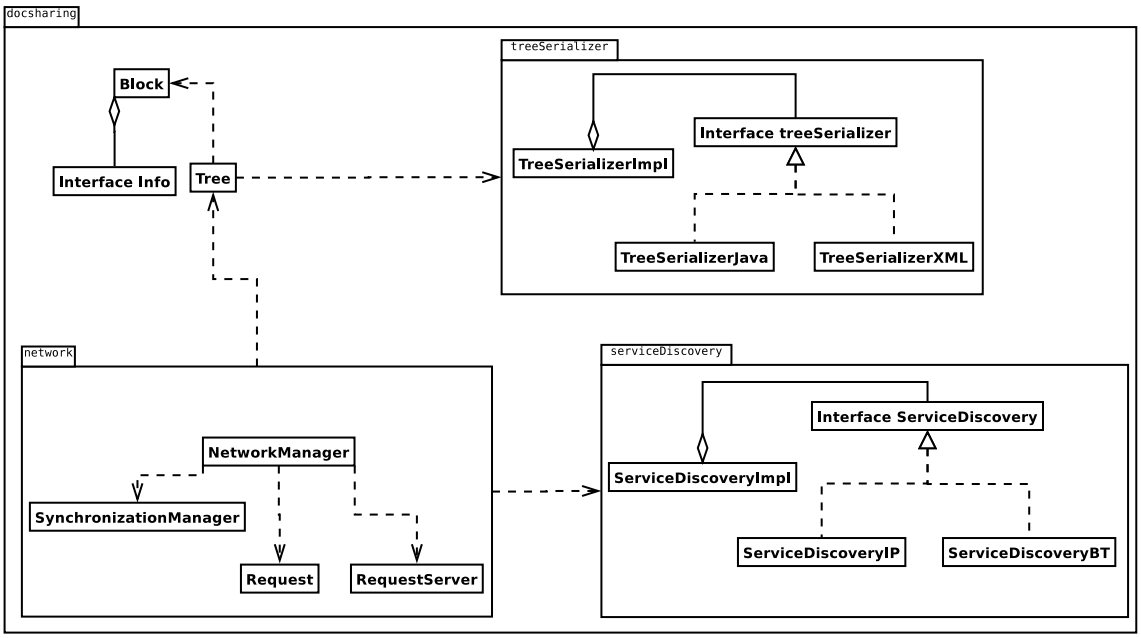


FIG. 8.1 – Architecture simplifiée de la partie gestion de document de Shaadhoc

l'arbre et les états des nœuds. Elle implémente en particulier les opérations de découpage et de fusion des feuilles de l'arbre qui permettent de modifier sa structure pour changer le grain de la découpe du document.

Chaque partie du document est représentée par un objet de type *Block* qui contient les annotations associées. Cette classe représente l'information portée par chaque feuille de l'arbre et définit, comme la classe *Tree*, ses propres méthodes de découpage et de fusion. Ces méthodes permettent de spécialiser le découpage en fonction du type de document (texte, carte stratégique,...). Dans le cadre spécifique de Shaadhoc, le document est une carte stratégique et l'interface *Block* est implémentée par la classe *MapBlock*. Lors du découpage d'une sous partie du document, cette classe répartie les informations contenues dans la feuille de l'arbre qui est découpée entre les nouvelles sous-parties en fonction des zones de la carte couvertes par ces sous-parties et de la localisation des informations en question. Chaque objet *MapBlock* contient plusieurs objets de type *Info* qui représentent les annotations.

La gestion des protocoles réseaux est réalisée par le paquetage *network*. Sa classe principale, *NetworkManager*, gère principalement les opérations de synchronisation avec les voisins en utilisant pour cela des requêtes qu'elle génère grâce à la classe *Request*. Afin de pouvoir répondre à des requêtes de façon réactive chaque nœud embarque un serveur (*RequestServer*) qui interagit directement avec la classe *NetworkManager*.

Enfin la gestion de la découverte de voisinage (paquetage *serviceDiscovery*) et du formatage des données (paquetage *treeSerializer*) ont été « externalisées » afin d'augmenter la modularité de l'application. Chacun de ces composants a été conçu suivant le motif de conception (design

pattern) *bridge* [91] et ces parties sont assez modulaires pour que l'on puisse changer d'implémentation au cours de l'exécution du code.

8.2 Découverte du voisinage et des services

La découverte du voisinage, c'est-à-dire des autres nœuds avec lesquels un nœud peut communiquer, est un problème récurrent dans les réseaux et plus particulièrement dans les réseaux MANets. Si une solution d'annuaire centralisé est envisageable dans les réseaux statiques, il est évident que cette solution n'est pas réaliste dans un contexte hautement dynamique à cause de la non-prédictibilité des connexions et déconnexions ou de la constitution éventuelle d'îlots.

La spécification Bluetooth a résolu le problème en définissant le protocole SDP [2] (*Service Discovery Protocol*, section 8.2.2). En revanche, même si certaines solutions émergent, il n'y a pas au moment de la rédaction de ce document de standard de fait qui ait émergé pour les réseaux ad hoc mobiles IP. Ces solutions sont issues de travaux dans les réseaux statiques ; on peut notamment citer SLP[92] (*Service Location Protocol*) qui a fait l'objet de plusieurs RFC (*Request For Comments*) ou UPnP™[93] destiné à des réseaux domestiques. Une volonté de standardisation apparaît tout de même dans les réseaux MANets. La tentative de l'IETF avec l'apparition du protocole NHDP[94] (*Neighborhood Discovery Protocol*), dont la RFC est encore à l'état de brouillon, ne comble qu'en partie ce manque. En effet, NHDP est basé sur la découverte de périphériques de OLSR[3] et ne permet pas la découverte de services. De même le standard IPv6 remplace la couche ARP par un protocole de découverte du voisinage destiné à obtenir l'adresse des routeurs dans un réseau statique.

8.2.1 Découverte de voisinage dans les réseaux mobiles IP (WiFi)

Lorsqu'une entité recherche un voisin pour synchroniser son document, la solution la plus simple dans un réseau dynamique offrant une connectivité IP (comme un réseau WiFi), est de diffuser un message dans le réseau. Lorsqu'un voisin reçoit ce message il y répond et l'émetteur du premier message peut alors se synchroniser avec lui. Cette solution ne permet pas d'utiliser de critère de sélection des voisins, et les réponses tardives et qui peuvent donc être postérieures au choix d'un voisin ne seront pas considérées ni utilisées pour construire une vision du voisinage. Cette solution n'est donc pas économe en nombre de messages au vu des messages inutiles qui circulent.

Une autre solution consiste à maintenir une liste de voisins accessibles la plus à jour possible en permanence, puis de choisir parmi ceux-ci lorsque l'on veut effectuer une synchronisation. Dans ce contexte la méthode généralement employée repose sur le fait que chaque entité du réseau diffuse de façon périodique des messages d'annonce, appelés *beacons*. Chaque membre du réseau peut se donner une période pendant laquelle un voisin détecté sera considéré comme potentiellement accessible. Cet algorithme très simple a l'avantage de laisser à chaque nœud le choix

de la fréquence à laquelle il souhaite s'annoncer, mais a l'inconvénient de ne pouvoir initier une découverte de façon pro-active. Un nouveau nœud doit simplement attendre une période de temps suffisamment longue pour obtenir une liste complète de ses voisins.

Esbjörnsson *et al.* [95] se sont penchés sur cette problématique et ont proposé une solution simple mais efficace. Le principe repose sur l'envoi de messages en *multicast* , au sens IP, entre des utilisateurs étant abonnés à un même groupe (i.e. partageant une adresse de *multicast* propre au protocole de découverte de services). Un initiateur envoie un message *HELLO* en *multicast*, et lorsqu'un voisin reçoit ce message, il répond lui aussi en *multicast* en ajoutant au message sa liste de voisins connus. Les voisins qui reçoivent ce nouveau message et qui ne sont pas dans la liste répondent avec un message contenant leur propre liste de voisins, alors que ceux qui sont présent dans cette liste ne répondent pas. L'algorithme se termine lorsque la composition du réseau se stabilise (voir figure 8.2). La constitution de cette liste n'étant pas atomique, sa composition peut éventuellement contenir des nœuds qui ne font plus partie du voisinage. Ainsi, il convient de définir pour chaque nœud une période de temps pendant laquelle il la considérera valide avant d'initier une autre découverte.

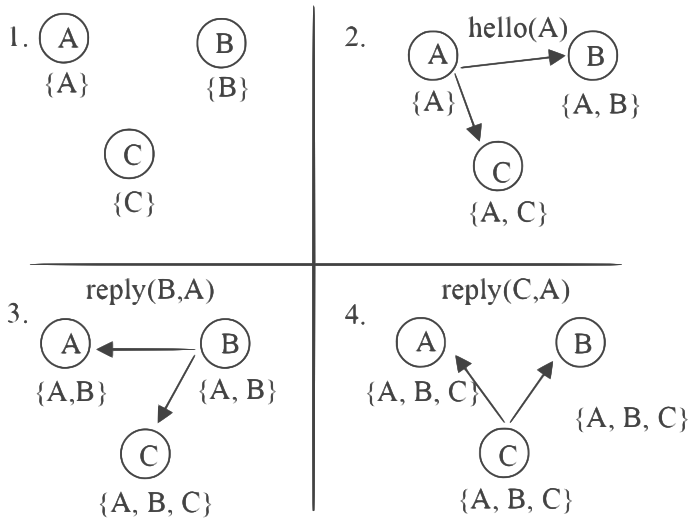


FIG. 8.2 – Algorithme de découverte de services de Esbjörnsson *et al.*

Cet algorithme converge et termine lorsque le voisinage n'évolue plus, et il permet aussi d'obtenir des informations sur le voisinage à deux sauts (voisins des voisins). En effet, le fait que les nœuds envoient aussi la liste de leurs voisins permet pour chaque nœud de reconstituer le voisinage direct de chacun de ses voisins et donc par transitivité son voisinage à deux sauts.

Cet algorithme peut aussi s'adapter à la dynamicité du réseau. La validité de ses résultats dépend de la période pendant laquelle on considère la liste des voisins comme correcte. Il est évident que plus le réseau est dynamique plus cette période doit être courte. Chaque nœud peut adapter cette durée en fonction des évolutions passées du réseau. Par exemple si la composition de

la liste des voisins d'un nœud n'a pas évolué depuis plusieurs périodes (phases lors desquelles la liste est réinitialisée) alors elles pourraient être allongées. Au contraire si entre deux périodes cette composition a beaucoup évolué, c'est-à-dire si le nombre de nœuds communs dans deux listes successives est faible, alors cette période peut être raccourcie.

C'est cet algorithme que nous avons choisi d'utiliser pour la mise en œuvre de Shaadhoc car il supporte le niveau de dynamicité relatif au contexte dans lequel nous nous plaçons. De plus sa simplicité nous permet de l'intégrer dans les différentes versions de notre logiciel. L'intégration d'un protocole plus standard reste toutefois envisageable dès lors qu'il sera clairement identifié et que des implémentations seront disponibles dans les différents langages que nous utilisons. SLP semble être un bon candidat car une version Java est désormais disponible, ce qui n'était pas le cas au moment de l'implémentation de Shaadhoc.

8.2.2 Bluetooth et SDP

La technologie Bluetooth [2] a été conçue dans une démarche orientée services avec l'objectif de remplacer une partie des connexions USB utilisées pour connecter certains périphériques qui nécessitent relativement peu de débit, comme des souris, des imprimantes ou des récepteurs GPS. Cette technologie peut aussi être utilisée pour des applications réseau, comme par exemple du transfert de fichier, et a comme avantage majeur le fait qu'il n'est pas nécessaire de configurer le réseau préalablement à son utilisation.

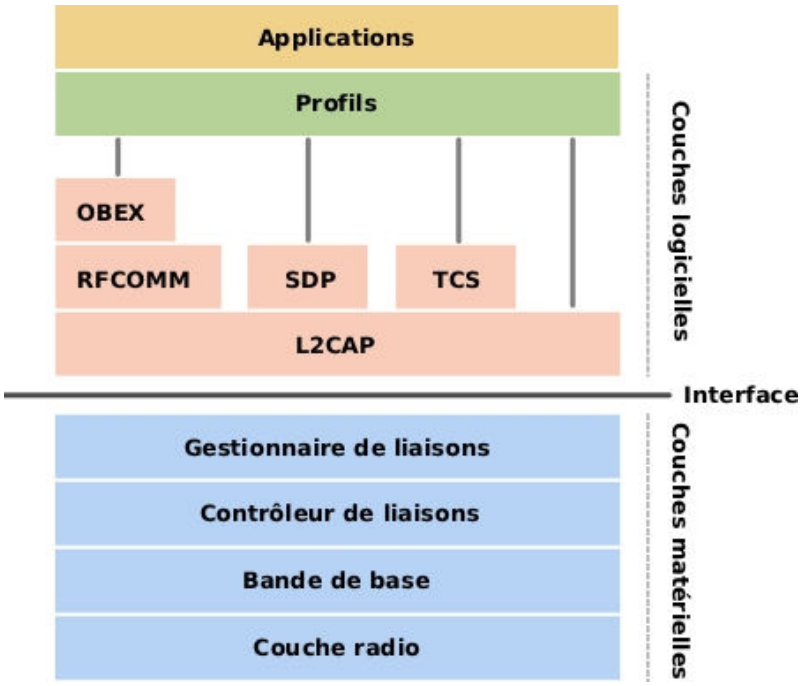


FIG. 8.3 – Schéma des couches de la pile Bluetooth (extrait de wikipedia et validé dans [96])

La figure 8.3 page précédente présente la pile des protocoles constituant la technologie Bluetooth. On accède aux couches gérées par le matériel au travers d'une interface appelée HCI (*Host Controller Interface*). Cette couche permet notamment de procéder à une découverte physique du voisinage qui se concrétise par la récupération des adresses MAC des voisins accessibles (cette découverte physique est différente de SDP, voir plus bas). On trouve ensuite la couche L2CAP (*Logical Link Control and Adaptation Protocol*) qui gère la segmentation et le ré-assemblage des paquets puis la couche RFCOMM qui émule un port série (RS-232). Cette couche est généralement utilisée lors du développement d'applications car elle offre des fonctionnalités proches d'une connexion TCP/IP. Enfin la couche OBEX [97] permet le transfert d'objets comme des fichiers. Elle a été développée à l'origine pour les transmission infra rouge (irDa).

Alors qu'il n'existe pas de standard pour la recherche de voisins dans un réseau ad hoc WiFi, la norme Bluetooth spécifie le protocole SDP [97] (*Service Discovery Protocol*). Ce standard permet de découvrir des services logiques associés à des voisin physiques. Pour cela, le standard Bluetooth définit des profils de services parmi lesquels :

- *Fax Profile*,
- *SPP : Serial Port Profile*,
- *HFP : HandsFree Profile*,
- *A2DP : Advanced Audio Distribution Profile*,

Ces services sont enregistrés sur chaque station au sein d'un serveur SDP, qui doit écouter sur le port 1 de la couche L2CAP et répondre aux requêtes SDP extérieures. Il suffit donc pour découvrir les voisins qui offrent un service donné de faire une découverte de périphériques au niveau matériel (couche HCI) puis d'interroger le serveur SDP associé à chaque matériel trouvé. Ainsi on peut savoir s'il fournit le service recherché et connaître quel protocole utiliser et quel numéro de port (ou *channel*) est associé à ce service sur le périphérique. La pile Bluetooth ne permet pas de communication en mode non connecté à l'aide de ses couches logicielles. Il n'est donc pas concevable d'adapter l'algorithme vu en section 8.2 page 115 à Bluetooth si ce n'est en simulant un mode non connecté. Nous utilisons donc le protocole SDP. Par contre si SDP, semble être la solution idéale d'un point de vue interface pour la découverte de services, il n'en est pas de même pour la découverte des voisins au niveau matériel. En effet, celle-ci s'effectue en temps constant de 10.24 secondes [2] ce qui limite fortement la dynamicité du système. On conviendra donc que même si Bluetooth a l'avantage d'offrir une implémentation complète de la découverte de services de par sa spécification, son utilisation est plus adaptée à un contexte moins mobile (d'autant que sa portée est plus faible que celle de WiFi).

8.3 Protocole de synchronisation

Une fois qu'une station a choisi un voisin (de façon aléatoire ou avec des critères contextuels) afin de mettre à jour sa version, elle peut appliquer le processus de synchronisation. Ce processus

est réalisé en utilisant des connexions deux à deux en mode connecté (i.e. TCP/IP pour WiFi et RFCOMM pour Bluetooth). Pour cela nous avons défini un protocole basé sur l'utilisation de plusieurs types de paquet qui sont tous composés d'un en-tête d'un octet pour spécifier le type de paquet, suivi des octets de donnée (cf. Tab. 8.1). Il n'est pas fait état dans cette section de considérations liées au langage de programmation, au mode de communication réseau, ou au format des données. Par exemple, la partie *DATA* peut aussi bien contenir la sérialisation d'un objet Java comme un arbre, sa description XML ou tout autre format propre à l'implémentation.

TYPE	DATA

TAB. 8.1 – Format des paquets de synchronisation utilisés dans Shaadhoc

Le tableau 8.2 page suivante présente les différents paquets utilisés pour réaliser les synchronisations. La figure 8.4 page suivante montre que la synchronisation s'effectue en quatre phases principales. La première phase consiste en la demande de synchronisation avec un voisin (étape 1). Cette étape est équivalente à une demande de rendez-vous, et, dans le cas où le voisin est disponible, chacun envoie sa description du document à l'autre. Chaque entité peut ensuite calculer la liste des nœuds de l'arbre dont elle a besoin afin de mettre à jour sa version du document et envoie une requête à l'autre station afin de récupérer les informations nécessaires (étape 2). Après que chaque station a envoyé les parties nécessaires à son voisin, chacune confirme cette réception avant de procéder aux modifications locales de sa version du document (étape 3). Chaque station peut ensuite reprendre son comportement normal de façon autonome (étape 4). L'opération critique de ce protocole est la réception des confirmations (étape 3). Si une des deux entités ne reçoit pas le message alors elle ne va pas appliquer les modifications sur son document et la cohérence du système peut être corrompue. Une entité pourrait ainsi récupérer un verrou sur une partie que l'autre n'aurait pas libéré. Pour remédier à ce problème les parties qui devraient passer de l'état *OPENED* à l'état *UNDESIRED* après réception de la confirmation peuvent changer d'état dès la réception des parties nécessaires provenant de l'autre entité (requête BLOCKS). Ainsi, la cohérence du système est garantie mais il se peut que les verrous sur ces parties soient perdus. Il faut toutefois relativiser ce risque, les messages de confirmation étant de petite taille leur temps de transmission est très court et les échecs sont donc rares (inexistants lors des simulations). Ce mécanisme n'est pas présent dans l'implémentation actuelle de Shaadhoc et fait partie des extensions futures.

8.4 Interface et scénario d'utilisation

Afin de donner une vision plus concrète de l'application Shaadhoc nous présentons dans cette section son interface dans sa version PDA (il existe une version PC). Nous nous appuyons sur différents cas d'utilisation comme l'ajout d'une information, le découpage et la fusion de parties.

Type	Valeur	Définition
REMOTE_SYNCHRO	0x1B	Demande de synchronisation
SYNCHRONIZE	0x03	Envoi de la description de l'arbre
GET	0x04	Demande des parties du document nécessaires à la mise à jour
BLOCKS	0x06	Envoi des parties nécessaires au voisin
OK	0x1A	Confirmation de réception des parties

TAB. 8.2 – Types de paquets utilisés lors des synchronisations de versions

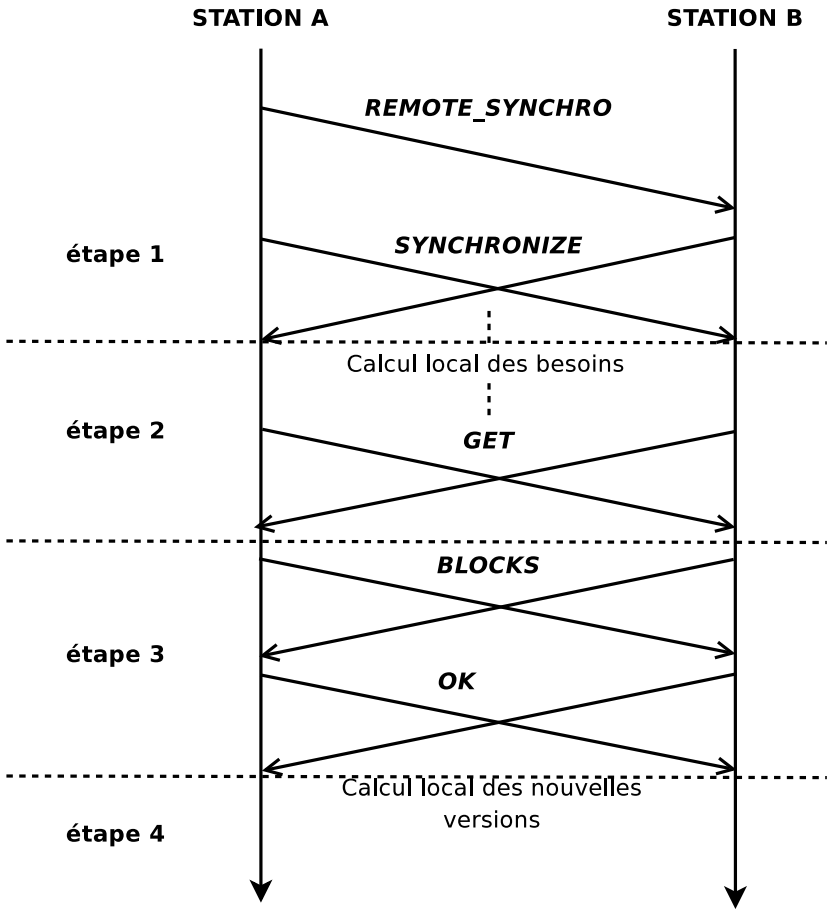


FIG. 8.4 – Processus de synchronisation entre deux stations

La figure 8.5 présente l'interface (PDA donc) de Shaadhoc. L'écran principal est réservé à l'affichage de la carte sur laquelle les modifications sont apportées et à une miniature permettant de repérer la projection actuelle de la fenêtre de visualisation sur la carte globale. La partie basse de l'écran permet d'accéder aux fonctions d'interaction :

- Ajout d'information (*Soldier* et *Tank*),
- Verrouillage/Déverrouillage d'une partie (*Lock* et *Unlock*),
- Découpage/Fusion de parties (*Split* et *Merge*),
- Sélection d'une information (*Select*) afin de lui appliquer une opération : déplacement, suppression, ...,
- Déplacement de la zone de la carte affichée (*Slide*),
- Suppression d'une information (*Remove*).

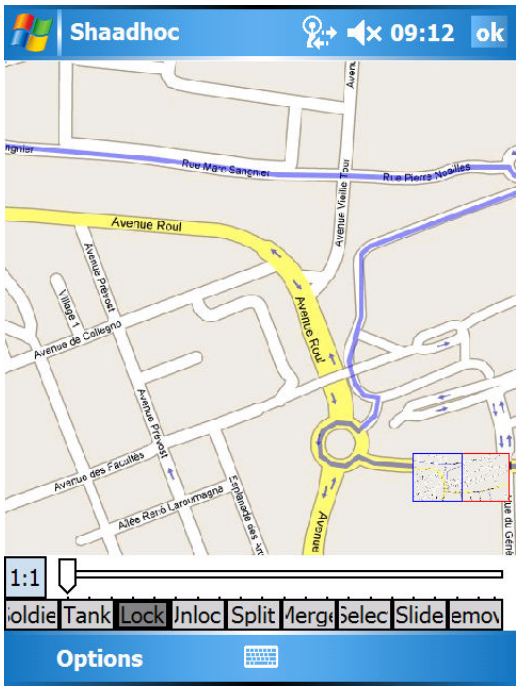


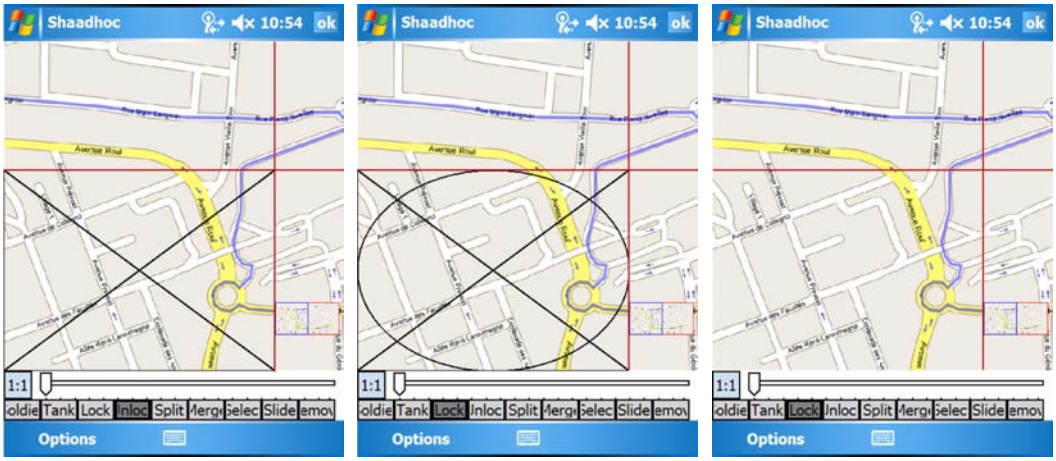
FIG. 8.5 – Vue générale de Shaadhoc

8.4.1 Verrouillage/Déverrouillage

La figure 8.6 page suivante illustre le mécanisme de verrouillage des parties dans Shaadhoc. La figure 8.6(a) représente la carte globale découpée en quatre sous-parties. La sous-partie en bas à gauche de l'écran est marquée d'une croix noire, ce qui signifie qu'elle est non modifiable et que l'utilisateur ne cherche pas à obtenir le droit de la modifier (état *UNDESIRED*). Lorsque l'utilisateur de la station souhaite verrouiller cette partie il utilise le bouton *Lock*. Cette partie passe ainsi dans l'état *REQUESTED* et est marquée par un cercle noir qui s'ajoute à la croix

(fig 8.6(b)) et qui signifie que l'utilisateur n'a pas le verrou sur cette zone mais cherche à l'obtenir. Enfin, quand la station peut se synchroniser avec l'entité disposant du verrou (i.e. en état *OPENED* pour cette partie) alors la partie passe dans l'état *CLOSED* et toutes les marques disparaissent. Il peut maintenant être modifié (fig 8.6(c)).

Lors du déverrouillage d'une partie qui se traduit par le passage de l'état *CLOSED* à l'état *OPENED*, la partie est marquée par la superposition d'une croix blanche et d'un cercle blanc, par analogie avec l'état *REQUESTED*.



(a) (b) (c)

FIG. 8.6 – Scénario de verrouillage d'une partie dans Shaadhoc

8.4.2 Découpage/Fusion

La figure 8.7 page suivante décrit les actions de découpage et de fusion de parties. Nous nous intéressons ici à la partie en bas à gauche de l'écran du PDA. On voit sur la figure 8.7(a) que l'utilisateur dispose du verrou (état *CLOSED*) sur toute cette partie du document car elle est exempte de tout symbole de surcharge. Il peut donc choisir de la découper en utilisant la fonction *Split*. Il choisit un point de découpe et crée quatre sous-parties (figure 8.7(b)). L'utilisateur peut ensuite modifier les nouvelles sous-parties (fig. 8.7(c)) ou en laisser le droit de modification à d'autres entités en en déverrouillant certaines (*Unlock*). Enfin si cet utilisateur (ou un autre) récupère les verrous sur les quatre sous-parties alors il peut les fusionner en utilisant la commande *Merge* (figure 8.7(d)).

8.5 Conclusion

Nous disposons à ce jour de deux implémentations fonctionnelles de Shaadhoc : une développée en Java qui peut fonctionner sur un PC classique et une développée en C# qui est destinée

(a) (b)

(c) (d)

FIG. 8.7 – Découpage/Fusion d'une partie

à une utilisation sur PDA (sous Windows Mobile). Cette dernière a été réalisée en partie grâce au support du LaBRI dans le cadre de la labellisation Carnot qui a permis de financer un contrat d'ingénieur pour une durée de trois mois au sein de l'équipe SOD. L'application Shaadhoc n'est pas encore distribuée car le type de licence n'a pas encore été déterminé. Celle-ci pourra être soit *open source* (comme la GPL [98] ou CECILL [99]) soit d'un autre type si le logiciel peut être valorisé en collaboration avec un industriel.

Chapitre 9

Conclusion générale

L'objectif de cette thèse était de fournir une base méthodologique sur laquelle s'appuyer pour concevoir des applications dans des réseaux militaires de type MANet. Ces travaux devaient conduire à la réalisation d'applications de référence. C'est l'ensemble de ces contributions que nous décrivons dans cette section.

Problématiques de référence et applications associées

Dans un premier temps, nous nous sommes intéressés à la collecte d'information pour l'aide à la prise de décision. Pour cela nous nous avons cherché à obtenir une vision aussi complète que possible de données statiques collectées sur un champ d'opération en utilisant des méthodes d'interpolation pour reconstruire les échantillons éventuellement perdus. La solution originale, à savoir l'architecture permettant l'interpolation, que nous avons conçue a été publiée lors de la conférence IEEE MILCOM 2007 [73] et une bourse IEEE a été obtenue pour participer à cette conférence. Ce travail s'est articulé autour du développement d'une application de collecte de température dans un réseau de capteurs qui est aujourd'hui opérationnelle et qui a pu être testée sur la plate-forme de capteurs de l'équipe. L'expérience acquise sur les équipements utilisés nous a aussi permis de donner un tutoriel lors de la conférence WISTP 2008 [74].

Dans un deuxième temps nous nous sommes intéressés au travail collaboratif dans un réseau MANet. Nous avons défini une méthode permettant d'assurer la cohérence de données modifiées de manière partagée dans un environnement instable. Ces recherches ont abouti à une solution générique reposant sur un système totalement distribué d'écriture exclusive sur un document et ont donné lieu à deux publications dans des conférences internationales. Une version préliminaire basée sur la constitution de groupes logiques d'entités mobiles a été publiée lors du workshop ACM MobiWac 2006 [75]. Le cœur de la solution présentée dans cette thèse et l'application de partage de carte stratégique ont été présentés lors de la conférence IEEE MILCOM 2006 [76].

Cette publication a notamment fait l'objet d'une collaboration avec Jacques Turbert (DGA), responsable métier télécommunications, qui en est co-auteur. Deux bourses ont été obtenues pour participer à cette conférence, l'une provenant de l'IEEE et l'autre de COMSOC, *IEEE Communications Society*. Ce travail a aussi donné lieu au développement de l'application Shaadhoc qui supporte l'édition collaborative distribuée d'une carte stratégique dans un environnement instable. Ce logiciel est arrivé à un stade pré-industriel grâce au soutien du projet par le LaBRI (Carnot). Une réflexion en terme de choix de licence est engagée afin de rendre le logiciel disponible, soit en *open source*, soit en le valorisant avec un industriel.

Les apports méthodologiques de cette thèse

D'un point de vue méthodologique, les apports de cette thèse se situent sur deux axes répondant à des problèmes majeurs des MANets plus particulièrement sensibles dans un contexte militaire.

En premier lieu, nous proposons une méthode permettant de construire une vision complète d'un phénomène qui n'a été observé que partiellement. Du point de vue méthodologique on peut retenir les enseignements suivants :

- il est nécessaire de prendre en compte les pertes d'informations pouvant intervenir au cours de la collecte. En effet les risque de perte son réels et pour prendre des décisions sensibles, les militaires doivent disposer de l'information la plus complète possible.
- On peut dans certains cas reconstituer des informations manquantes en s'appuyant sur une connaissance experte du contexte.
- l'utilisation de caches permet souvent de pallier l'aspect instable du réseau.

Ces considérations permettent d'obtenir les informations contextuelles les plus complètes possibles et ainsi d'aider au mieux la prise de décision. En plus de ces enseignements, on pourra tirer partie de l'expérience acquise dans le développement de notre application de référence pour la résolution d'autres problèmes.

En second lieu, nous avons étudié le partage et la modification collaborative et distribuée de documents dans un réseau MANet. La méthode proposée permet d'assurer la cohérence globale du document lors de sa modification. D'un point de vu méthodologique on peut retenir que :

- il est utile de découper (dynamiquement) les tâches globales en plusieurs sous-tâches indépendantes pour faire en sorte que deux utilisateurs aient peu de raison d'interagir. Les interactions minimales qui demeurent doivent être pilotées globalement par des contrôles locaux.
- il peut être intéressant dans certaines situations de considérer une information floue (principe de précaution).
- il est parfois indispensable de considérer le contexte applicatif pour résoudre des problèmes

durs insolubles dans un contexte trop général. Par exemple, la hiérarchie militaire permet d'apporter une structure logique entre les utilisateurs et de donner des droits et donc des capacités supplémentaires à certains.

Ces principes sont mis en œuvre dans l'application Shaadhoc, qui au delà de son utilité propre, pourra servir de base d'expérience pour la réalisation d'autres services.

Perspectives de recherche

Cette thèse montre l'intérêt de la prise en compte des spécificités du contexte applicatif dans la conception de nouveaux services pour les réseaux mobiles ad hoc. Les travaux dans ce sens peuvent être approfondis ou étendus pour élargir le spectre des contextes dans lesquels nos approches pourront être mises en œuvre.

Collecte d'informations

Nos travaux sur l'interpolation de mesures physiques permettant la construction d'une vision continue d'un phénomène (comme la variation de température) pourront être adaptés à d'autres contextes (comme l'observation du passage d'un nuage chimique ou radioactif). La technique de positionnement des nœuds est un point qui devra être précisé et pourra éventuellement faire l'objet d'une collaboration avec l'équipe d'Abderrahim Benslimane qui dispose d'une plus grande expertise dans ce domaine et qui travaille lui aussi avec la DGA. Enfin, les résultats présentés dans ce documents ont été obtenus grâce au déploiement sur une plateforme de test à petite échelle en laboratoire. Pour s'assurer que ces travaux sont opérationnels dans un contexte réel, il serait nécessaire d'effectuer des tests en milieu naturel sur une plateforme comptant plus de nœuds.

Travail collaboratif

L'édition collaborative d'un document est une question très intéressante dans les réseaux MANets. L'application Shaadhoc que nous avons présentée est fonctionnelle et déployée sur une plate-forme expérimentale. Des essais à grande échelle pourront être mis en place grâce au déploiement de Shaadhoc sur un nombre de nœuds plus importants suite à la mise en place par notre équipe d'une plate-forme d'environ soixante assistants personnels. Ces tests devraient confirmer les bonnes capacités de passage à l'échelle que nous avons constatées lors des simulations. Pour autant ces simulations n'ont pas montré de problème particulier au moment des étapes d'envoi et de réception des accusés de réception et il se peut que cette situation soit due à un artefact amené par le simulateur. Les deploiements futurs pourront ainsi lever cette incertitude. L'extension logicielle majeure consiste dans la mise en œuvre effective de la gestion de la perte de verrous. Enfin ces travaux seront pour partie repris dans le cadre d'une nouvelle thèse (DGA) dont l'objet est de proposer et concevoir des services sécurisés dans le contexte d'une flotte de drones.

Bibliographie

[1] B.P. Crow, I. Widjaja I., L.G. Kim, and P.T. Sakai. IEEE 802.11 Wireless Local Area Networks. *IEEE Communications Magazine*, vol. 35 issue 9 :116–126, Sep 1997.

[2] Bluetooth SIG. Specification of the Bluetooth System, 2007.

[3] RFC3626 : Optimized Link State Routing Protocol (OLSR), 2003.

[4] C. Perkins, E. Belding-Royer, and S. Das. RFC3561 : Ad hoc On-Demand Distance Vector (AODV) Routing, 2003.

[5] IETF MANET group.
http ://www.ietf.org/html.charters/manet-charter.html.

[6] ACM International Workshop on VehiculAr Inter-NETworking : VANET 2008.
http ://www.sigmobile.org/workshops/vanet2008/.

[7] DGA. Projet BOA : Bulle Opérationnelle Aéroterrestre.
http ://www.defense.gouv.fr/dga/content/download/43878/438181/file/presentation_de_boa_22_boa1.pdf, juin 2002.

[8] Jean-Baptiste Waldner. *Nano-informatique et Intelligence Ambiante*. Hermes Science, 2007. ISBN : 2746215160.

[9] Kewei Sha, Weisong Shi, and Orlando Watkins. Using wireless sensor networks for fire rescue applications : Requirements and challenges. In *Proceedings of the 6th IEEE International Conference on Electro/Information Technology*, May 2006.

[10] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, , and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *European Workshop on Sensor Networks*, 2005.

[11] Gyula Simon, Miklós Maróti, Ákos Lédeczi, György Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and Ken Frampton. Sensor network-based countersniper system. In *SenSys '04 : Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 1–12. ACM Press, 2004.

[12] Cederqvist et al. *Version Management with CVS*. Network Theory LTD.

[13] Michael Pilato. *Version Control With Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[14] Konrad Lorincz and Matt Welsh. MoteTrack : A Robust, Decentralized Approach to RF-Based Location Tracking. *Personal and Ubiquitous Computing, Special Issue on Location and Context-Awareness*, october 2006. ISSN : 1617-4909.

[15] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1987. isbn : 0934613273.

[16] Andrew Tanenbaum. *Computer Networks*. Prentice Hall PTR, 2003. ISBN : 0130661023.

[17] Charles E. Perkins. *Ad Hoc Networking*. Addison-Wesley, 2001. ISBN : 0201309769.

[18] M. Ripeanu. Peer-to-Peer Architecture Case Study : Gnutella Network. Technical report, University of Chicago, 2001.

[19] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24(4) :234–244, 1994.

[20] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing*, 353 :153–181, 1996.

[21] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. *SIGCOMM Comput. Commun. Rev.*, 34(4) :145–158, 2004.

[22] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03 : Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34, 2003.

[23] Amin Vahdat and David Becker. Epidemic Routing for Partially-Connected Ad Hoc Networks. Technical Report CS-200006, Dept. of Computer Science, Duke University, April 2000.

[24] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic Routing in Intermittently Connected Networks. *Service Assurance with Partial and Intermittent Resources*, pages 239–254, 2004.

[25] Khaled A. Harras, Kevin C. Almeroth, and Elizabeth M. Belding-Royer1. Delay Tolerant Mobile Networks (DTMNs) : Controlled Flooding in Sparse Mobile Networks. In *Proc. of Networking 2005*, pages 1180–1192, 2005.

[26] Tara Small and Zygmunt J. Haas. The shared wireless infostation model : a new ad hoc networking paradigm (or where there is a whale, there is a way). In *MobiHoc '03 : Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 233–244, 2003.

[27] Jörg Ott, Dirk Kutscher, and Christoph Dwertmann. Integrating dtn and manet routing. In *CHANTS '06 : Proceedings of the 2006 SIGCOMM workshop on Challenged networks*, pages 221–228, 2006.

[28] Boulkenafed M. and Issarny V. AdHocFS : sharing files in WLANs. In *Proceedings of Network Computing and Applications*, pages 156–163, 2003.

[29] Benjamin Atkin and Kenneth P. Birman. MFS : an adaptive distributed file system for mobile hosts. Technical report, Departement of Computer Science Cornell University, Ithaca, 2003.

[30] Frédéric Guidec and Yves Mahéo. Opportunistic Content-Based Dissemination in Disconnected Mobile Ad Hoc Networks. In *International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2007)*, pages 49–54, Papeete, French Polynesia (Tahiti), November 2007. IEEE Computer Society Press.

[31] Alina Bejan and Ramon Lawrence. Peer-to-peer cooperative driving. In *Proceedings of ISCIS2002 International Symposium on Computer and Information Sciences*, pages 259–264. CRC Press, October 2002.

[32] Majid Ali Khan and Ladislau Boloni. Convoy driving through ad-hoc coalition formation. In *RTAS '05 : Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 98–105. IEEE Computer Society, 2005.

[33] Fredrik Axelsson and Mattias Östergren. SoundPryer : Joint Music Listening on the Road. In *Extended Abstracts of UBICOMP'02*, 2002.

[34] Mickael Wiberg. FolkMusic : a mobile peer-to-peer entertainment system. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, 2004.

[35] Arianna Bassoli, Julian Moore, and Stefan Agamanolis. tunA : Local Music Sharing with Handheld Wi-Fi Devices. In *Proceedings of the 5th Wireless World Conference*, 2004.

[36] Arianna Bassoli, Julian Moore, and Stefan Agamanolis. *Consuming Music Together Social and Collaborative Aspects of Music Consumption Technologies*, chapter Tuna : Socialising Music Sharing on the Move, pages 151–172. Springer Netherlands, 2006.

[37] Proxidating. http ://www.proxidating.com/.

[38] Aka'aki. http ://www.aka-aki.com/.

[39] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks : a survey. *Comuter Networks*, 38(4) :393–422, march 2002.

[40] Klaus Finkenzeller. *RFID Handbook Fundamenetals and Applications in Contactless Smart Cards and Identification*. Wiley, 2003. ISBN :0470844027.

[41] Lionel M. Ni, Yunhao Liu, Yiu Cho Lau, and Abhishek P. Pati. LANDMARC : Indoor Location Sensing Using Active RFID. *Wireless Networks*, 10(6), november 2004.

[42] International Telecommunication Union. The Internet of Things, 2005.

[43] Championchip.
http ://www.championchip.com.

[44] Xingxin (Grace) Gao, Zhe (Alex) Xiang, Hao Wang, Jun Shen, Jian Huang, and Song Song. An approach to security and privacy of rfid system for supply chain. In *CEC-EAST '04 : Proceedings of the E-Commerce Technology for Dynamic E-Business, IEEE International Conference*, pages 164–168, Washington, DC, USA, 2004. IEEE Computer Society.

[45] Wolfgang Rankl and Wolfgang Effing. *Smart Card Handbook*. Wiley, 2004. ISBN :0470856688.

[46] ATmel. 8-bit microcontroller with 128k bytes in-system programmable flash.
http ://www.atmel.com/atmel/acrobat/doc2467.pdf.

[47] XBow. Mica2.
http ://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.

[48] XBow. Micaz.
http ://www.xbow.com/products/product_pdf_files/wireless_pdf/6020-0060-01_a_micaz.pdf.

[49] Intel Corp. Intel xscale micorarchitecture.
http ://www.intel.com/design/intelxscale/xscaleproductbriefweb.pdf.

[50] IEEE Computer Society. Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs).
http ://standards.ieee.org/getieee802/download/802.15.4-2006.pdf, 2006.

[51] Zigbee Alliance.
http ://www.zigbee.org/.

[52] S. Madden P. Levis, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. *Ambient Intelligence*, chapter TinyOS : An Operating System for Sensor Networks, pages 115—-148. Springer, 2005.

[53] TinyOS. http ://fr.wikipedia.org/wiki/TinyOS.

[54] David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC Language : A Holistic Approach to Networked Embedded Systems. In *Proceedings of Programming Language Design and Implementation (PLDI) 2003*, june 2003.

[55] GlobalSecurity.org. Sound Surveillance System (SOSUS).
http ://www.globalsecurity.org/intell/systems/sosus.htm.

[56] C. E. Nishimura and D. M. Conlon. IUSS dual use : Monitoring whales and earthquakes using SOSUS. *Marine Technology Society Journal*, volume 27 no. 4 :13–21, 1994.

[57] Christopher W. Clark and David K. Mellinger. Application of Navy IUSS for whale research. *The Journal of the Acoustical Society of America*, volume 96 issue 5, november 1994.

[58] Carnegie Mellon Univ., editor. *Proceedings od the Distributed Sensor Nets workshop*. Dept. Comput. Sci., 1978. Pittsburg.

[59] Gabriel M. Rebeiz. *RF MEMS : Theory, Design, and Technology*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[60] S. Kumar and D. Shepherd. SensIT : Sensor Information Technology for the warfighter. In *Proccedings of the 4th Conference on Information Fusion*, 2001.

[61] R. Wright, , L. P. Flynn, H. Garbeil, A. J. Harris, and E. Pilger. Automated Volcanic Eruption Detection Using MODIS. *Remote Sensing of Environment*, pages 135–155, 2002.

[62] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking : design tradeoffs and early experiences with zebranet. *SIGPLAN Not.*, 37(10) :96–107, 2002.

[63] Jenna Burrell, Tim Brooke, and Richard Beckwith. Vineyard computing : Sensor networks in agricultural production. *IEEE Pervasive Computing*, 3(1) :38–45, 2004.

[64] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02 : Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. ACM Press, 2002.

[65] Jyh-How Huang, Saqib Amjad, and Shivakant Mishra. Cenwits : a sensor-based loosely coupled search and rescue system using witnesses. In *SenSys '05 : Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 180–191. ACM Press, 2005.

[66] Miklos Maroti, Branislav Kusy, Gyula Simon, and Akos Ledeczi. The flooding time synchronization protocol. Technical Report ISIS-04-501, Vanderbilt University, 2004.

[67] David Deeths and Glenn Brunette. Using NTP to Control and Synchronize System Clocks - Part I :Introduction to NTP. Technical report, Sun MicroSystems, 2001.

[68] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *SenSys '04 : Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49. ACM Press, 2004.

[69] Kannan Srinivasan and Philip Levis. RSSI is Under Appreciated. In *Proceedings of EmNets 2006 : Third Workshop on Embedded Networked Sensors*, may 2006.

[70] Fredrik Gustafsson and Fredrik Gunnarsson. Positioning using time-difference of arrival measurements. In *In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2003.

[71] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *MobiCom'00 : Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 32–43. ACM, 2000.

[72] Rong Peng and Mihiail L.Sichitiu. Angle of arrival localization for wireless sensor networks. In *Proceedings of IEEE SECON 2006*, 2006.

[73] Lionel Barrère, Serge Chaumette, and Cyril De Peretti. Delay Tolerant Dynamic Data Collection Over a Sensor Network. In *Proceedings of MILCOM 2007*, Orlando FL, USA, october 2007. IEEE.

[74] Jérémie Albert, Lionel Barrère, and Serge Chaumette. A Tutorial on using Sensor Networks, ZigBee and RFIDs. In *WISTP 2008 : Workshop in Information Security Theory and Practices*.

[75] Lionel Barrère, Arnaud Casteigts, and Serge Chaumette. A totally decentralized document sharing system for mobile ad hoc networks. In *MobiWac '06 : Proceedings of the 4th ACM international workshop on Mobility management and wireless access*, pages 116–120, Torremolinos, Malaga, Spain, 2006. ACM.

[76] Lionel Barrère, Serge Chaumette, and Jacques Turbert. A tactical active information sharing system for military manets. In *Military Communications Conference, 2006. MILCOM 2006*, Washington, DC, October 2006. IEEE.

[77] Mark de Berg, Marc Van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry : Algorithms and Applications*, chapter chap 14 : Quadtrees, pages 291–304. Springer, 2000. isbn : 3540656200.

[78] Zhiqun Chen. *Java Card Technology for Smart Cards : Architecture and Programmer's Guide*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[79] Harold F. Tipton and Micki Krause. *Information Security Management Handbook*, chapter 88.2 : What are hash algorithms, page 1142. CRC Press, 2007.

[80] Luc Hogie, Pascal Bouvry, Frédéric Guinand, Grégoire Danoy, and Enrique Alba. Simulating Realistic Mobility Models for Large Heterogeneous MANETs. In *Demo proceedings of the 9th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM'06). October 2-6. Malagà, Spain*. IEEE, 2006.

[81] Luc Hogie. *Mobile Ad Hoc Networks : Modelling, Simulation and Broadcast-based Applications*. PhD thesis, University of Le Havre (France) and University of Luxembourg, april 2007.

[82] Christian Bettstetter, Giovanni Resta, and Paolo Santi. The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(3) :257–269, 2003.

[83] The Network Simulator - ns-2.
http ://nsnam.isi.edu/nsnam/index.php/User_Information, 2008.

[84] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim : a library for parallel simulation of large-scale wireless networks. *SIGSIM Simul. Dig.*, 28(1) :154–161, 1998.

[85] Scalable Networks Simulator.
http ://www.scalable-networks.com/.

[86] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications & Mobile Computing (WCMC) : Special issue on Mobile Ad Hoc Networking : Research, Trends and Applications*, 2(5) :483–502, 2002.

[87] Yves Métivier, Nasser Saheb, and Akka Zemmari. Analysis of a randomized rendezvous algorithm. *Inf. Comput.*, 184(1) :109–128, 2003.

[88] Jesse Liberty. *Programming C#*. O'Reilly, 2003.

[89] David Flanagan. *Java in a Nutshell*. O'Reilly, 2005.

[90] Elliotte Rusty Harold and W. Scott Means. *XML en concentré*. O'Reilly, 2005.

[91] Laurent Debrauwer. *Design Patterns - Les 23 modèles de conception : descriptions et solutions illustrées en UML 2 et Java*. Editions ENI, 2007.

[92] Erik Guttman. Service Location Protocol : Automatic Discovery of IP Network Services. *IEEE Internet Computing*, 03(4) :71–80, 1999.

[93] UPnP™Standards. http ://www.upnp.org/standardizeddcps/default.asp.

[94] T. Clausen, C. Dearlove, and J. Dean. MANET Neighborhood Discovery Protocol (NHDP). http ://www.ietf.org/internet-drafts/draft-ietf-manet-nhdp-09.txt, march 2009.

[95] Mattias Esbjörnsson and Mattias Östergren. Hocman : supporting mobile group collaboration. In *CHI '02 : CHI '02 extended abstracts on Human factors in computing systems*, pages 838–839. ACM, 2002.

[96] Robert Morrow. *Bluetooth : Operation and Use*. McGraw-Hill Professional, 2002. isbn : 9780071387798.

[97] *Bluetooth Profiles : The Definitive Guide*. Prentice Hall PTR, 2003.

[98] The GNU General Public License.
http ://www.gnu.org/licenses/gpl.html.

[99] CECILL : Licence française de logiciel libre.
http ://www.cecill.info/.