THÈSE

Présentée pour l'obtention du titre de

Docteur de l'Université de Versailles Saint-Quentin-en-Yvelines

spécialité : Informatique

par **Jean-Marc PERCHER**

UN MODÈLE DE DÉTECTION D'INTRUSIONS DISTRIBUÉE POUR LES RÉSEAUX SANS FIL AD HOC

Soutenue le 21 Septembre 2004 devant la Commission d'examen :

Président	M.	LUDOVIC MÉ	Professeur à Supélec
Rapporteurs	M.	PIERRE ROLIN	Direction des Recherches de la Division R&D de France Télécom
	M.	GUY PUJOLLE	Professeur, Université de Paris VI
Examinateurs	M.	JEAN-PIERRE CLAUDÉ	Professeur, Université de Versailles Saint-Quentin-en-Yvelines
	M.	SAMIR THOMÉ	Professeur, Université de Versailles Saint-Quentin-en-Yvelines

Résumé

Les MANET (*Mobile Ad hoc NETwork*), ou réseaux sans fil ad hoc sont des réseaux ne disposant d'aucune infrastructure préexistante et formés de nœuds mobiles interconnectés par des liaisons sans fil. Leurs architectures évoluent au gré de l'apparition et du mouvement des nœuds. Ces réseaux présentent un contexte nouveau pour le routage et la mise en œuvre des services de sécurité. Alors que de nombreuses propositions de protocoles de routage ad hoc ont été développées et évaluées, il existe, aujourd'hui, peu de propositions pour protéger les MANET.

Dans cette thèse, nous proposons un modèle de sécurité pour les MANET. Celui-ci associe les mécanismes de sécurité préventifs et un système de détection d'intrusions ou IDS (*Intrusion Detection System*). Notre recherche est centrée sur l'IDS dont l'architecture doit être adaptée aux caractéristiques des MANET. Les spécificités de la détection des intrusions dans les MANET sont dues à :

- l'absence d'infrastructure préexistante et permanente,
- l'instabilité des liaisons entre les nœuds,
- l'hétérogénéité des équipements,
- l'instabilité de la topologie du réseau,
- la difficulté à identifier les nœuds présents dans le réseau.

Nous proposons une architecture d'IDS distribuée dotée d'un système de coopération basé sur des agents mobiles et montrons par des simulations comment l'utilisation des agents mobiles améliore la fiabilité du système de coopération entre les IDS installés sur les nœuds.

Nous présentons un prototype utilisé pour valider, dans un environnement de tests, les caractéristiques de l'IDS distribué et évaluer ses performances.

Mots-clés : détection d'intrusions, IDS distribué, MANET, réseaux sans fil ad hoc, agents mobiles.

Remerciements

Je remercie mon directeur de thèse, Jean-Pierre Claudé*, qui m'a permis de réaliser ce travail et a su me conseiller très efficacement.*

Je remercie, Ludovic Mé*, d'avoir accepté de co-diriger ma thèse. Sa rigueur scientifique et son soutien m'ont aidé tout au long de ce travail.*

Je remercie vivement les rapporteurs, Guy Pujolle *et Pierre* Rolin *d'avoir accepté cette lourde charge ainsi que celle de participer au jury de thèse.*

Je remercie Samir Thomé *d'avoir bien voulu juger ce travail et participer au jury de thèse.*

Je remercie Jacky Charruault*, Directeur de l'ESEO, pour sa confiance et son soutien.*

Je remercie Olivier Camp*, qui m'a très souvent fait progresser en partageant mes réflexions et en m'apportant ses conseils.*

Je remercie mes collègues enseignants-chercheurs du Département Informatique de l'ESEO pour leur soutien et leur aide : Patrick Albers*, Olivier* Beaudoux*, Slimane* Hammoudi*, Daniel* Schang*.*

Je remercie mes collègues enseignants-chercheurs de l'équipe SSIR de Supélec qui m'ont apporté leur aide dans mon travail de recherche : Christophe Bidan*, Bernard* Jouga*, Riccardo* Puttini*.*

Je remercie David Tylski *et Julien* Gagnet *pour l'aide qu'ils m'ont apportée, respectivement lors des simulations et de la mise au point du prototype.*

Je remercie Guillaume Desgeorge *et Joel* Madeline *pour leurs relectures.*

Je remercie toute l'équipe du projet RNRT RAHMS qui m'a permis de découvrir et de m'intéresser aux réseaux ad hoc.

Je remercie tous ceux qui à l'ESEO m'ont apporté leur aide dans mon travail de recherche et de rédaction.

Je remercie enfin ma famille qui m'a soutenu durant ces quatre années.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Depuis leur apparition, les réseaux locaux sans fil [1] ont suscité l'intérêt des professionnels confrontés à des besoins de mobilité et de connexion au réseau de leur organisme. Les réseaux 802.11, normalisés par l'IEEE [55] en 1997, se sont rapidement imposés jusqu'à, dans certains cas, remplacer les traditionnels réseaux filaires de type ETHERNET. Depuis leur arrivée sur le marché, l'évolution régulière de leurs performances et la baisse de leur coût d'acquisition ont contribué à accélérer leur diffusion.

L'IEEE 802.11 offre deux modes d'utilisation : le mode infrastructure et le mode ad hoc. Le mode infrastructure, aussi appelé mode cellulaire, s'appuie sur une topologie construite autour de points d'accès fixes. Ces derniers ont pour fonction de gérer les échanges entre les nœuds mobiles situés dans leur zone d'émission-réception. Plusieurs points d'accès peuvent être interconnectés par un réseau fédérateur, appelé *système de distribution*, pour offrir des connexions à un plus grand nombre de nœuds ou augmenter l'espace de mobilité des nœuds. Le mode infrastructure est utilisé pour étendre le réseau de l'organisme et offrir aux utilisateurs mobiles un moyen d'accès à des ressources permanentes et centralisées. Le mode ad hoc, lui, établit un échange point à point entre deux nœuds mobiles. Si deux nœuds ne partagent pas les mêmes zones d'émission-réception, la connexion directe est impossible. Dans ce cas, les nœuds intermédiaires sont utilisés pour établir un chemin entre les nœuds source et destination. Ces réseaux, dont l'architecture évolue au gré du mouvement et de l'apparition des nœuds, sont appelés MANET (*Mobile Ad hoc NETwork*) [20] ou réseaux spontanés.

Dans certains contextes, les utilisateurs peuvent bénéficier des caractéristiques des MANET pour échanger leurs informations. Un exemple fréquemment cité, dans les domaines civil et militaire, est un réseau ad hoc formé

par les interconnexions entre des véhicules en mouvement. Dans le domaine industriel les réseaux de capteurs (*Sensor Networks*) peuvent former un MANET pour s'adapter à différents environnements. Mais de nombreuses autres situations de la vie courante sont adaptées à l'usage des MANET. Nous considérons, par exemple, un réseau créé pour les besoins et la durée d'une réunion regroupant des participants de différents organismes, ou un réseau créé entre les élèves et leur professeur dans une salle de classe pour la durée d'un cours.

Les MANET constituent un nouveau type de réseaux qui, aujourd'hui encore et malgré les besoins déjà identifiés des utilisateurs, appartiennent au domaine de la recherche. Cette recherche est particulièrement active et principalement centrée sur les protocoles de routage spécifiques aux MANET [74] puisque chaque nœud peut être considéré comme un routeur mobile. Le caractère dynamique de l'architecture des MANET présente aussi un contexte nouveau et spécifique pour la mise en œuvre des services de sécurité [83]. La vulnérabilité des nœuds est accentuée par l'utilisation de liaisons sans fil. De plus, les nœuds mobiles d'un MANET ne peuvent bénéficier des protections offertes par les mécanismes de sécurité basés sur les services d'un serveur central ou sur une conception adaptée de l'architecture réseau. De ce fait, de nombreux mécanismes utilisés dans les réseaux filaires ne sont pas utilisables dans les MANET et chaque nœud mobile ne peut compter que sur ses propres ressources pour se protéger. La sécurité du protocole de routage fait l'objet de nombreux travaux de recherche dans le domaine spécifique de la sécurité des réseaux ad hoc [83][36][67][69].

Malgré les mécanismes de sécurité mis en place sur un système, la découverte quotidienne de nouvelles vulnérabilités[1,2] et la difficulté à réaliser toutes les corrections des nombreuses failles identifiées sont responsables d'une très grande partie des attaques réussies[3]. Dès lors, on ne peut considérer les mécanismes de sécurité comme un moyen de protection suffisant, absolu et définitif. De ce fait, pour renforcer l'action des mécanismes de sécurité préventifs, on est amené, dans certains contextes, à mettre en place des systèmes de sécurité complémentaires, comme les systèmes de détection d'intrusions ou IDS (*Intrusion Detection System*).

Depuis les travaux de Dorothy E. Denning [24] sur la détection d'intrusions, les performances des IDS ont progressé. L'IDWG (*Intrusion Detection Working Group de l'IETF*) propose un modèle d'architecture pour les

1. Common Vulnerabilites and Exposures, http:// www.cve.mitre.org/
2. http://www.securityfocus.com/
3. Computer Emergency Response Team http://www.cert.org/stats/cert_stats.html

IDS [49]. Le processus de détection d'intrusion commence par une collecte des données. Celle-ci peut être réalisée au niveau du réseau ou des fichiers du système : on qualifie alors respectivement les IDS de *Network Based* et de *Host Based*. Dans une seconde phase, les données collectées sont analysées. Deux approches principales sont couramment utilisées pour la détection. L'approche comportementale, ou *anomaly based detection*, consiste à identifier la déviation du comportement courant d'un *système* par rapport à un modèle de référence. L'approche par scénarios, ou *misuse based detection*, s'appuie sur la recherche, dans les données collectées, de signatures d'attaques préalablement spécifiées. La dernière phase est la gestion des alertes générées par la détection d'une intrusion.

Dans cette thèse, nous nous intéressons à la sécurité des nœuds et nous proposons de renforcer l'action des mécanismes de sécurité préventifs par un système de détection d'intrusions. Compte tenu de leurs caractéristiques intrinsèques, les MANET constituent un environnement spécifique et encore peu exploré pour la détection d'intrusions. Cette recherche nous a amenés à étudier les architectures des IDS distribués, pour finalement proposer une nouvelle architecture distribuée formée d'un ensemble d'IDS coopératifs installés sur chaque nœud.

Pour réaliser la coopération entre tous les IDS, nous proposons d'utiliser des agents mobiles et nous montrons comment les agents mobiles permettent de pallier l'absence de nœud central, l'instabilité des liaisons sans fil, et de prendre en compte la mobilité des nœuds.

Nous avons réalisé un prototype pour valider l'architecture proposée et effectuer des mesures de performances. Après ces premiers résultats significatifs, nous avons poursuivi nos expérimentations et nos mesures en associant sur chaque nœud l'IDS SNORT, reconnu pour ses performances, et notre IDS à base d'agents mobiles. Notre objectif est d'une part d'augmenter la base d'attaques détectées, et d'autre part d'utiliser les agents mobiles pour adapter SNORT au contexte des réseaux ad hoc.

Ce mémoire est organisé de la manière suivante. Le chapitre 2 décrit un état de l'art des trois domaines que nous abordons dans cette thèse. Le premier domaine concerne les MANET, le second celui des systèmes de détection d'intrusions et le troisième celui des agents mobiles.

Le chapitre 3 est la présentation de notre modèle du système de détection d'intrusions. A partir de l'étude des caractéristiques des MANET nous établissons, dans une première étape, les spécifications générales d'un IDS pour MANET. Dans une seconde phase nous proposons, pour les MANET, un IDS doté d'une architecture distribuée et coopérative, dans laquelle chaque

nœud est équipé d'un IDS local que nous appelons LIDS pour *Local Intrusion Detection System*. Nous présentons ensuite un modèle de coopération basé sur des agents mobiles. Nous terminons ce chapitre par une description fonctionnelle de l'architecture interne des LIDS.

Le chapitre 4 décrit l'implémentation et la validation expérimentale du prototype réalisé. Nous présentons les résultats des tests effectués pour valider les caractéristiques de l'IDS. Nous proposons et validons aussi une extension des LIDS pour intégrer des IDS locaux, autres que les LIDS, au sein de l'architecture distribuée. La validation expérimentale de cette extension est réalisée avec le NIDS (*Network Intrusion Detection System*) SNORT.

En conclusion, le dernier chapitre propose un récapitulatif des principaux travaux réalisés et des résultats obtenus. Nous terminons en donnant différentes perspectives pour la poursuite de nos travaux.

Chapitre 2

État de l'art

2.1 Les MANET (Mobile Ad hoc NETworks) et la sécurité

2.1.1 Définition des MANET

La couverture géographique est un critère fréquemment retenu pour classer les réseaux. Par cette caractéristique, nous distinguons les PAN (*Personal Area Network*) , les LAN (*Local Area Network*) , les MAN (*Metropolitan Area Network*) et les WAN (*Wide Area Network*) . Le type du média (*réseau filaire ou sans fil*) et le types d'informations transportées (*voix, données, images*) sont aussi utilisés pour différencier les réseaux.

Aujourd'hui, les différents réseaux que nous utilisons pour échanger nos informations possèdent une caractéristique commune : la stabilité de leur architecture. Qu'ils soient filaires ou sans fil, utilisés pour le transport des données ou de la voix, les architectures des réseaux sont statiques. Les réseaux de télécommunications connaissent peu d'évolution et ont des durées de vie qui dépassent plusieurs années. De même, les réseaux d'entreprises, conçus pour s'adapter rapidement aux changements fréquents d'organisation, possèdent des architectures stables dont l'évolution est basée sur la configuration et l'optimisation des équipements d'interconnexion.

Il existe une autre *famille* de réseaux, apportant davantage de flexibilité, de facilité de construction et de déplacement, et qui se différencie de tous les réseaux évoqués précédemment par l'absence de toute infrastructure fixe préexistante.

Ces réseaux sont formés d'un ensemble de nœuds mobiles interconnectés par des liaisons sans fil. Leurs architectures évoluent au gré de l'apparition et du mouvement des nœuds. Ils sont appelés réseaux spontanés ou réseaux

ad hoc, signifiant, selon la locution, *à cet effet*. Les environnements adaptés à leur utilisation sont donc caractérisés par l'absence (ou la détérioration) d'infrastructure réseau préexistante, telles les opérations de secours après un sinistre, les missions d'exploration ou les applications militaires. D'autres contextes, dont les spécificités de mobilité, de sécurité, de durée d'utilisation, et de rapidité de déploiement rendent impossible l'accès à une infrastructure réseaux existante, sont autant de cas d'utilisation idéale des réseaux ad hoc.

Pour préciser les domaines d'utilisation, nous donnons ci-après les principales caractéristiques des réseaux ad hoc.

2.1.2 Les caractéristiques des MANET

Un réseau ad hoc est un système autonome constitué de nœuds mobiles. Ces derniers communiquent avec leurs voisins par des liaisons sans fil point à point. Quand les zones d'émission/réception de deux nœuds en communication sont disjointes, les nœuds intermédiaires sont alors sollicités pour assurer le routage. A partir de cette définition générale nous présentons les caractéristiques principales qui différencient un réseau ad hoc d'un réseau doté d'une architecture fixe.

Les propriétés des nœuds dans un réseau ad hoc :

- *La mobilité de tous les nœuds* est une caractéristique intrinsèque des MANET . Le déplacement des nœuds provoque des modifications aléatoires et non prédictibles de l'architecture du réseau. De ce fait, les techniques de routage des réseaux classiques, basées sur des routes préétablies par des équipements spécialisés et dédiés, ne peuvent plus fonctionner correctement.
- *L'équivalence des nœuds* est une spécificité des MANET. Dans un réseau classique, il existe une distinction nette entre les nœuds terminaux (stations, hôtes) qui supportent les applications et les nœuds internes du réseau (routeurs), chargés de l'acheminement des données. Cette différence n'existe pas dans les réseaux ad hoc car tous les nœuds peuvent être amenés à assurer des fonctions de routage.
- *Le nombre de nœuds mobiles* présents dans un MANET varie selon les besoins ou la position de chaque nœud. D'une façon plus générale aucune limitation n'est faite sur la taille ou le nombre de nœuds d'un réseau ad hoc.
- *Les ressources énergétiques des nœuds mobiles*, alimentés par des sources d'énergies autonomes (*batteries*) sont limitées. Ces équipements intègrent des modes de gestion d'énergie et il est important

que les protocoles mis en place dans les réseaux ad hoc prennent en compte cette caractéristique.

L'absence de serveur centralisé rend complexe le contrôle et la gestion d'une architecture qui se forme et évolue au gré de l'apparition et des déplacements des nœuds. En conséquence, il n'existe aucune hiérarchie entre les nœuds et aucun service réseau ne peut prétendre être centralisé.

Les liaisons physiques s'appuient sur les technologies de communications sans fil, indispensables à la mise en place d'un réseau ad hoc. Malgré des progrès très importants, leurs performances sont encore aujourd'hui en deçà de celles des technologies des réseaux LAN filaires.

Les vulnérabilités des réseaux sans fil sont par nature plus sensibles aux problèmes de sécurité. Pour les réseaux ad hoc, le principal problème ne se situe pas tant au niveau du support physique mais principalement dans le fait que tous les nœuds sont équivalents et potentiellement nécessaires au fonctionnement du réseau. Les possibilités de s'insérer dans le réseau sont plus grandes, la détection d'une intrusion ou d'un déni de service est plus délicate et l'absence de centralisation rend plus complexe la collecte d'informations pour la détection d'intrusions.

2.1.3 La norme IEEE 802.11b en mode ad hoc

Le norme IEEE 802.11 [55] définit deux modes opérationnels : le mode infrastructure, qui nécessite la présence d'un équipement spécialisé, appelé point d'accès, pour contrôler les communications entre les hôtes, et le mode ad hoc, basé sur des communications point à point entre les hôtes d'un réseau. Le mode infrastructure représente la configuration la plus répandue. Celui-ci est utilisé par exemple pour permettre aux postes nomades d'accéder aux réseaux d'entreprises ou aux réseaux publics (*hotspot*). Les spécificités du mode ad hoc, présentées dans [52] [30] [1], sont telles que ses applications appartiennent encore aujourd'hui au domaine de la recherche.

La normalisation des réseaux locaux par L'IEEE [1] couvre les couches physique et liaison de données du modèle de référence O.S.I [2]. La couche liaison de données retenue par l'IEEE est composée de deux sous-couches : la sous-couche LLC (*Logical Link Control*) et la sous-couche MAC (*Medium Access Control*) . La couche LLC ou IEEE 802.2 est commune à toutes les normes 802 de l'IEEE. La couche IEEE 802.11 est une couche commune à plusieurs

1. Institute of Electrical and Electronics Engineers
2. Open Systems Interconnection

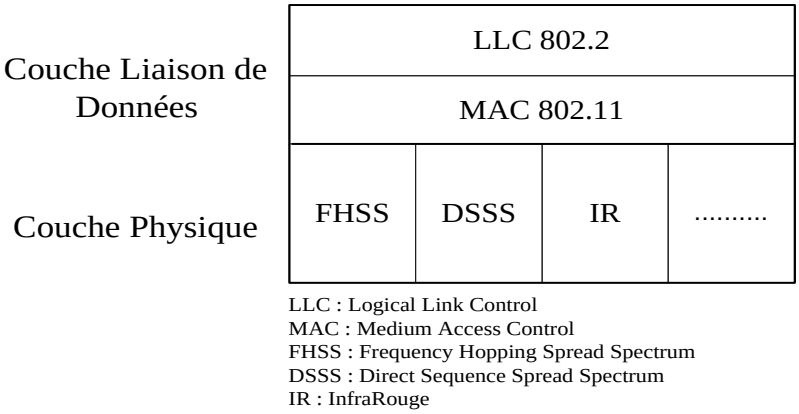


FIG. 2.1 – *Modèle IEEE 802.11*

couches physiques, représentées sur la figure 2.1. Les principales caractéristiques des normes IEEE 802.11 sont présentées le tableau A.1 de l'annexe A. L'apport de la norme IEEE 802.11b par rapport à la version initiale 802.11 de 1997 est la normalisation pour la couche physique de deux nouveaux débits : 5,5 et 11 Mbps et l'abandon du saut de fréquence. Dans cette section nous présentons les technologies de l'IEEE 802.11b spécifiques au mode ad hoc.

2.1.3.1 La couche Physique 802.11b

Les normes IEEE 802.11 et 802.11b utilisent la bande des 2,4 GHz de l'ISM (Industrial , Scientific and Medical). Ces fréquences, reconnues par les organismes réglementaires internationaux[3], sont utilisables sans licence mais avec des puissances limitées selon les pays.

En mode ad hoc la couche physique de l'IEEE 802.11b utilise la modulation à spectre étalé en séquence directe ou DSSS (*Direct Sequence Spread Spectrum*) . Les techniques d'étalement de spectre permettent d'améliorer la fiabilité, d'augmenter le débit, et de limiter l'effet des interférences. La technique de signalisation en séquence directe divise la bande des 2,4 GHz en 14 canaux de 22 MHz pour transmettre les données sur l'un de ces canaux. La technique DSSS de la norme IEEE 802.11 spécifie un codage de chaque bit de donnée (1 ou 0) par une séquence spécifique de 11 bits appelée séquence de Barker. Chaque élément de la séquence de codage ou *chip* est ensuite transformé en une forme d'onde émissible après une modulation adaptée. Les symboles, formés de 11 bits, sont émis à la vitesse de 1 MSps (*Millions*

3. FCC-Federal Communications Commission, l'ETSI-European Telecommunications Standards Institute, ou le MKK au Japon

Débit	Longueur du code	Modulation	Débit (symboles)	Nbre de bits /symbole
1 Mbps	11 bits (Barker Sequ.)	PSK	1 MSps	1
2 Mbps	11 bits (Barker Sequ.)	QPSK	1 MSps	2
5,5 Mbps	8 bits (CCK)	QPSK	1,375 MSps	4
11 Mbps	8 bits (CCK)	QPSK	1,375 MSps	8

TAB. 2.1 – *Caractéristiques de la couche physique - IEEE 802.11 et 802.11b*

de Symboles par seconde) selon une modulation BPSK (*Binary Phase Shift Keying*). Une modulation de type QPSK (Quadrature Phase Shift Keying) permet, en transmettant deux bits de données par élément de signal, d'atteindre un débit de 2 Mbit/s. Les débits de 5,5 Mbps et 11 Mbps de l'IEEE 802.11b sont obtenus en conservant le débit de 11 Mchip/s avec la technique de codage CCK (*Complementary Code Keying*) qui associe un mot de 8 bits sélectionné parmi un ensemble de 64 mots pour coder respectivement 4 bits et 8 bits par porteuse. Les deux modes font appel à la technique de modulation QPSK pour transmettre 1,375 MSps.

Le tableau 2.1 présente les différentes caractéristiques de DSSS pour des normes IEEE 802.11 et 802.11b.

2.1.3.2 La couche IEEE 802.11

Suivant les recommandations de l'IEEE, la fonction principale de la couche MAC est de gérer les accès au support. Deux méthodes, le PCF (*Point Coordination Function*) et DCF (*Distributed Coordination Function*) sont proposées dans la norme IEEE 802.11. Le mode ad hoc est uniquement basé sur le DCF. Le principe de base pour contrôler l'accès au média repose sur le CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*). Ce mode d'accès est une adaptation du protocole CSMA/CD (*Carrier Sense Multiple Access with Collision Detection*) utilisé dans les réseaux de type ETHERNET. Le protocole CSMA/CD autorise une station à émettre quand le média est libre et ensuite écoute le support pour détecter une éventuelle collision résultante d'une émission simultanée d'une autre station.

Les liaisons radios n'étant pas full-duplex, la détection d'une collision pendant l'émission est impossible. Le protocole 802.11 utilise des trames d'acquittement, appelées ACK (*ACKnowledgement*), pour confirmer qu'une collision ne s'est pas produite et donc que la trame émise a été correctement reçue.

Le contrôle de l'accès au support, suivant le DCF, utilise des espaces

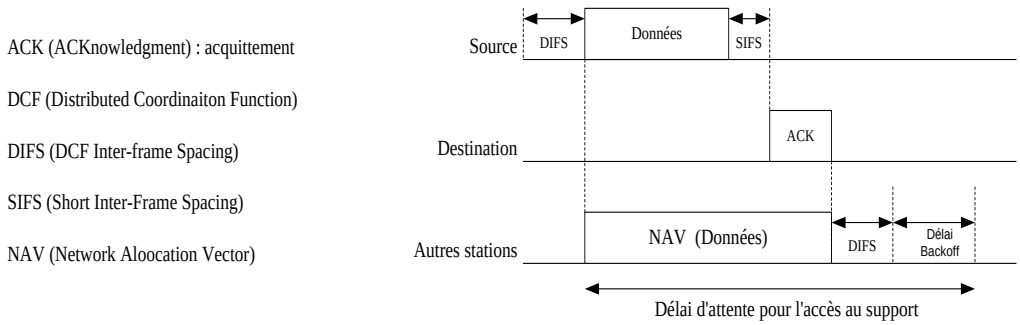


FIG. 2.2 – *Contrôle de l'accès au support*

inter-trames ou IFS pour (*Inter-Frame Space*). Avant d'émettre une trame de données, une station écoute le support pour déterminer si une transmission est en cours. Si aucune transmission n'est détectée pendant un intervalle égal à DIFS (*Distributed Inter-Frame Space*), la station transmet sa trame de données immédiatement. Toutefois plusieurs stations pourraient émettre simultanément. Pour prendre en compte cette situation la station destination confirme à l'émetteur, après un délai SIFS (*Short Inter-Frame Space*) plus court que l'intervalle DIFS, la bonne réception de sa trame par l'émission d'une trame d'acquittement ACK (Voir la figure 2.2).

Quand une transmission est en cours et qu'une station a des données à transmettre, celle-ci attend que le canal reste libre pendant un intervalle de temps égal à DIFS augmenté de la valeur courante de son temporisateur de backoff (*Backoff Timer*) après la fin de la transmission pour pouvoir émettre ses données.

L'algorithme de backoff exponentiel permet de résoudre le problème de l'accès au support quand plusieurs stations ont des données à transmettre en même temps.

Après la fin d'une transmission, pour pouvoir émettre, une station teste que le support reste libre pendant une période égale à DIFS augmentée de la valeur courante du temporisateur de backoff.

Lorsque le support est libre, les stations décrémentent leur temporisateur jusqu'à ce que le support soit occupé ou que la valeur du temporisateur soit nulle. Si le média redevient occupé avant que le temporisateur atteigne la valeur 0, le temporisateur de backoff se bloque jusqu'à la prochaine libération du média plus une période égale à DIFS. Dès que la valeur du temporisateur de backoff est nulle, la station émet sa trame.

Malgré ce mécanisme, deux stations peuvent émettre en même temps.

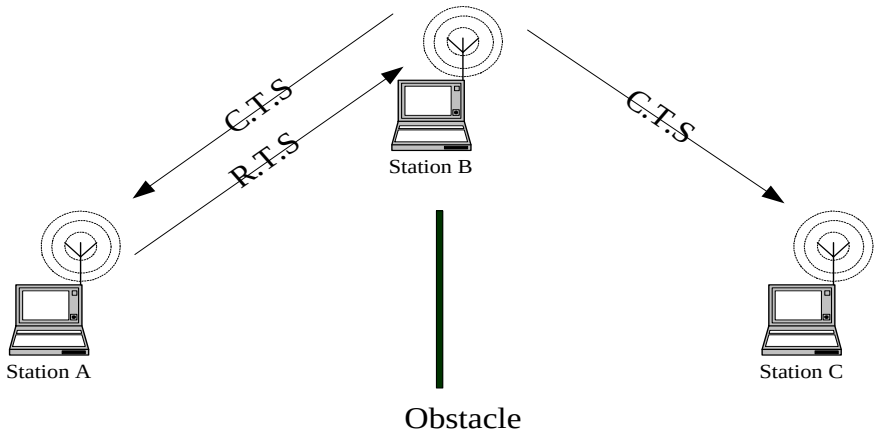


FIG. 2.3 – *Stations cachées*

Dans ce cas, chaque station régénère une nouvelle valeur de temporisateur calculée en fonction du nombre de tentatives de retransmission de la trame et de la valeur d'une variable aléatoire CW (*Contention Window*) jusqu'à une valeur maximale. Lorsque la transmission se produit, le temporisateur de backoff est alors initialisé avec sa valeur minimale.

Cet algorithme permet aux stations d'accéder au support avec la même probabilité, mais ne permet pas de garantir un délai minimal nécessaire aux applications multimédia.

L'écoute du support est réalisée à la fois au niveau de la couche physique avec le PCS (*Physical layer Carrrier Sense*) et au niveau de la couche MAC avec le VCS (*Virtual Carrier Sense*). Le PCS analyse l'activité du support pour détecter la présence d'autres stations 802.11. Le VCS est un mécanisme de réservation basé sur l'échange de trames RTS/CTS (*Request To Send/Clear To Send)* entre une station source et une station destination avant tout envoi de données.

Ces trames possèdent un champ, appelé *durée de vie* ou *Duration*, qui indique aux autres stations du réseau le temps pendant lequel le média est réservé pour la transmission des données.

Une trame RTS est émise par la station source vers la station destination avant tout envoi de données. A la réception d'une trame RTS, la station destination répond par l'émission d'une trame CTS dont le champ durée de vie sera lu par les autres stations pour mettre à jour leur NAV (*Network Allocation Vector*). Après la réception du CTS, la station source est assurée de la disponibilité et de la stabilité du support pour la transmission de ses données et de leur acquittement.

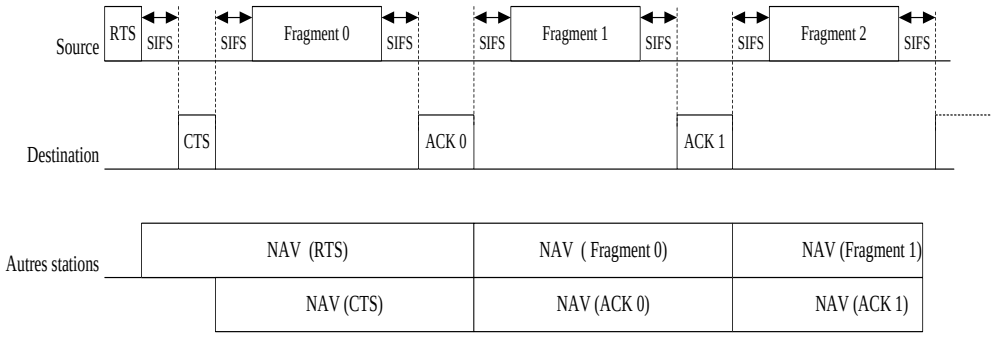


FIG. 2.4 – *Fragmentation des trames 802.11*

Ce mécanisme additionnel, consommateur de bande passante, n'est pas obligatoire pour transmettre toutes les trames. Il peut être activé uniquement pour l'émission des trames dont la longueur est supérieure à la valeur de la variable, appelée *RTS_ Threshold*, dont la retransmission n'est pas souhaitée. Ce mécanisme est aussi utilisé pour résoudre le problème de la station cachée (voir la figure 2.3) qui peut se rencontrer dès que trois stations forment un réseau et que deux d'entre elles ne sont pas à la portée l'une de l'autre. Dans ce cas, si le RTS émis par une station source n'est pas reçu par les deux autres, le CTS émis par la station destination sera pris en compte par la troisième station.

La fragmentation consiste à diviser les trames de taille importante en fragments plus petits. Ceci à pour conséquence d'améliorer les performances du réseau car plus une trame est petite plus le risque d'erreur de transmission est faible. Tout d'abord, les trames RTS/CTS donnent la durée du prochain fragment et de son acquittement. Elles ne sont utilisées qu'une seule fois avant la transmission du premier fragment (Voir la figure 2.4).

Ensuite, les champs *durée de vie* du premier fragment et de la trame ACK permettent aux autres stations du réseau de mettre à jour leur temporisateur NAV. La station source garde le contrôle du support pendant toute la durée de transmission de la trame fragmentée. La station destination acquitte chaque fragment correctement reçu par une trame ACK. Si un ACK n'est pas correctement reçu, la station source suspend l'émission des fragments et tente d'accéder de nouveau au support pour reprendre la transmission à partir du dernier fragment non acquitté.

Nous donnons dans l'annexe A, les caractéristiques des trois types de

trames 802.11 mises en œuvre dans le mode ad hoc :

- Les trames de données,
- Les trames de contrôle, utilisées pour contrôler l'accès au support (RTS, CTS, ACK,...),
- Les trames de gestion, utilisées pour les opérations d'association, de synchronisation et d'authentification.

Dans la section suivante, nous présentons le principe d'établissement d'un réseau ad hoc dès que deux stations 802.11, ou plus, peuvent communiquer par des liaisons point à point.

2.1.3.3 L'établissement d'un réseau IEEE 802.11b en mode ad hoc

Dans un réseau ad hoc ou IBSS (*Independent Basic Service Set*) , les stations communiquent entre-elles sans utiliser les services d'un point d'accès.

Pour contrôler les fonctionnalités d'accès au canal radio par le protocole CSMA/CA, et celles du mode d'économie d'énergie, les stations d'un IBSS doivent maintenir une horloge synchronisée. La synchronisation des horloges locales des stations est contrôlée par l'émission périodique de trames balises (*Beacon frames*).

Ces trames balises doivent être transmises à intervalles réguliers appelés TBTT (*Target Beacon Transmission Time*). Aussi, à l'approche de la fin d'un intervalle TBTT, chaque station suspend tout autre trafic pour permettre l'émission ou la réception de trames balises.
A la fin de l'intervalle TBTT, chaque station attend la fin d'un délai aléatoire supplémentaire, appelé *Beacon Backoff*, pour transmettre sa trame balise.
A la fin de ce délai, si aucune trame balise n'a été reçue, la station envoie sa trame balise avec la valeur de son horloge locale (temporisateur TSF : *Timing Synchronisation Function*).
Les autres stations comparent la valeur de leur temporisateur avec celle transmise dans la trame balise et ajustent leur temporisateur sur la plus grande des deux valeurs. De cette façon, l'horloge locale d'une station n'est jamais reculée et toute les stations sont synchronisées sur l'horloge la plus rapide des stations de l'IBSS. La station qui à émis la dernière trame balise n'est pas autorisée à passer en mode économie d'énergie.

Quand une station veut s'intégrer à un IBSS, identifié par son SSID (*Service Set identity*) indexSSID : Service Set identity déjà existant, elle écoute le support pour détecter des trames spécifiques. Cette écoute (*scanning*) peut être réalisée de deux manières différentes : en écoute passive, la station attend de recevoir une trame balise (*Beacon*); en écoute active, la station émet

une trame de requête (*Probe Request*) et attend de recevoir la réponse (*Probe Response*). Celle-ci est générée par la station qui a émis la dernière trame balise (*Beacon Frame*).

2.1.3.4 La gestion de l'énergie

En environnement mobile, le contrôle de la consommation d'énergie est une fonction importante. La norme IEEE 802.11 dispose d'un mode de gestion d'énergie distribué, pour s'adapter au caractère spontané de l'IBSS. Une station d'un IBSS peut être en mode éveillé (*awake*), c'est à dire opérationnelle, ou en mode veille (*doze*).

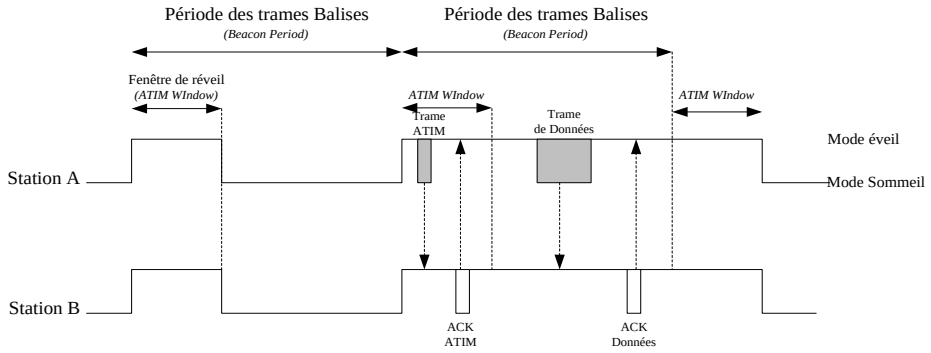


FIG. 2.5 – *Contrôle du mode économie d'énergie*

Dans un IBSS, une station annonce aux autres stations son passage en mode économie d'énergie en positionnant le bit *Pwr Mgt* dans le champ contrôle de la dernière trame émise. Les stations en mode économie d'énergie quittent leur état de veille à intervalles réguliers (*Beacon Period*) pour pouvoir recevoir les trames ATIM (*Ad hoc Traffic Information Map*) leur annonçant que d'autres stations de l'IBSS ont des trames à leur transmettre (Voir la figure 2.5). Les stations attendent les trames ATIM pendant une durée égale à la valeur de l'*ATIM window*. Cette valeur est une caractéristique de l'IBSS, diffusée par les trames de balises (*Beacon*). Les trames ATIM doivent être acquittées par la station destination, qui restera dans l'état éveillé pendant la durée de l'intervalle *Beacon Period*, pour recevoir les trames de données.

Dans un IBSS, le mode ad hoc de la norme IEEE 802.11 permet à deux stations mobiles situées à portée l'une de l'autre d'échanger directement leurs données. Quand celles-ci sont dans l'incapacité de réaliser un échange direct, la mise en œuvre d'un protocole de routage ad hoc, distribué sur les nœuds du

réseau, est nécessaire. Les protocoles de routage développés pour les réseaux à architectures fixes s'appuient sur des équipements spécialisés et permanents. Ils sont, dans l'état, inadaptés aux réseaux ad hoc. Une part importante des travaux de recherche dans le domaine des réseaux ad hoc concerne l'optimisation de la fonction de routage.

Dans la section suivante nous présentons les principes et caractéristiques des principaux protocoles de routage ad hoc [74] [27].

2.1.4 Les protocoles de routage ad hoc

Un protocole de routage a pour fonction de déterminer le chemin entre deux nœuds en fonction d'une stratégie prédéfinie. Dans un réseau ad hoc, le protocole de routage, distribué sur l'ensemble des nœuds, vise de plus à minimiser le temps d'établissement de la route, aussi appelé temps de *latence*, ainsi que l'utilisation des ressources nécessaires à cette opération. Nous limitons notre présentation aux protocoles de routage point à point, encore appelés protocoles de routage *unicast*.

Lorsque deux nœuds échangent directement leurs paquets de données sans passer par des nœuds mobiles intermédiaires, la connexion est dite *directe*. Si le chemin entre les nœuds source et destination nécessite la présence de plusieurs nœuds intermédiaires, la connexion est alors qualifiée de *multi hop*.

Dans un réseau à architecture fixe, les routes vers les différents réseaux sont pré-définies et maintenues par les équipements d'interconnexion fixes, appelés routeurs. L'architecture dynamique d'un réseau ad hoc, qui résulte du mouvement, de l'apparition des nœuds ou de l'état de la connexion physique, nécessite une mise à jour régulière des tables de routage situées dans chaque nœud. Pour acheminer un paquet entre deux nœuds mobiles d'un réseaux ad hoc, le mécanisme de base est l'inondation. Celle-ci consiste à transmettre le paquet à l'ensemble des nœuds du réseau. L'inondation est réalisée par diffusions successives à l'ensemble des voisins de chaque nœud. Des mécanismes complémentaires de contrôle peuvent être utilisés pour éviter les bouclages ou la duplication des paquets. Ce mécanisme d'inondation, très coûteux en ressources réseau, ne peut s'appliquer qu'à de très petits réseaux. Les protocoles de routage viseront, eux, à limiter la propagation des paquets par inondation. Selon le rôle joué par les nœuds dans la diffusion des messages, nous pouvons réaliser une première différenciation des protocoles de routage :

- lorsque tous les nœuds ont des fonctionnalités identiques, le protocole est qualifié d'*uniforme*,
- si certains noeuds ont des fonctionnalités particulières dans la diffusion

des messages, le protocole est *non uniforme*.

Nous pouvons aussi différencier les protocoles *uniformes* par leur mode de fonctionnement. Comme dans les réseaux à architectures fixes, les deux techniques à *état de liens* et à *vecteur de distance* sont utilisées. Les protocoles de routage à *vecteur de distance* possèdent une table de routage qui, à chaque nœud du réseau, associe l'adresse du prochain nœud, *le vecteur,* et le nombre de nœuds intermédiaires, *la distance*.

Les protocoles de routage à *état de liens* utilisent, eux, une base de données qui leur permet de construire la topologie du réseau et de connaître ainsi le chemin vers tous les nœuds du réseau. Une métrique basée généralement sur plusieurs paramètres relatifs aux liaisons est utilisée pour sélectionner la meilleure route.

Si la mise à jour de la table de routage est effectuée de façon périodique, le protocole est alors qualifié de *proactif*. De cette façon, l'ensemble des routes, mêmes celles inutilisées, sont mises à jour et demeurent immédiatement disponibles. Cette technique génère de nombreux paquets sur le réseau mais permet de minimiser le temps de découverte d'une route lors de son utilisation.
Pour diminuer la charge réseau due aux paquets de mise à jour, certains protocoles de routage déclenchent la recherche d'une route uniquement quand celle-ci est demandée. Le délai d'obtention d'une route est alors plus long. Les protocoles qui utilisent ce mode de fonctionnement sont dits *réactifs*.

Pour diminuer le nombre de messages de contrôle nécessaires à la découverte des routes, les protocoles de routage *non uniformes* sélectionnent certains nœuds pour créer des architectures hiérarchiques et dynamiques. Ainsi, pour les protocoles à sélection de voisins, chaque nœud décharge la fonction de routage à un sous ensemble de voisins directs. Tandis que pour les protocoles à partitionnement, le réseau est découpé en zones dans lesquelles le routage est assuré par un unique nœud maître. Certains de ces protocoles, qualifiés d'hybrides, utilisent conjointement le routage à état de liens et le routage à vecteur de distance.

Les principaux protocoles de routage sont présentés selon leur mode de fonctionnement sur la figure 2.6, page 31. Dans l'annexe B, nous donnons les caractéristiques, les principes de fonctionnement et les références des protocoles cités.

Depuis juillet 2003, seuls les protocoles AODV [62] et OLSR [17], font l'objet d'un RFC avec le statut *Expérimental*. Ces deux protocoles appartiennent, chacun, à une famille de protocoles et possèdent des stratégies de

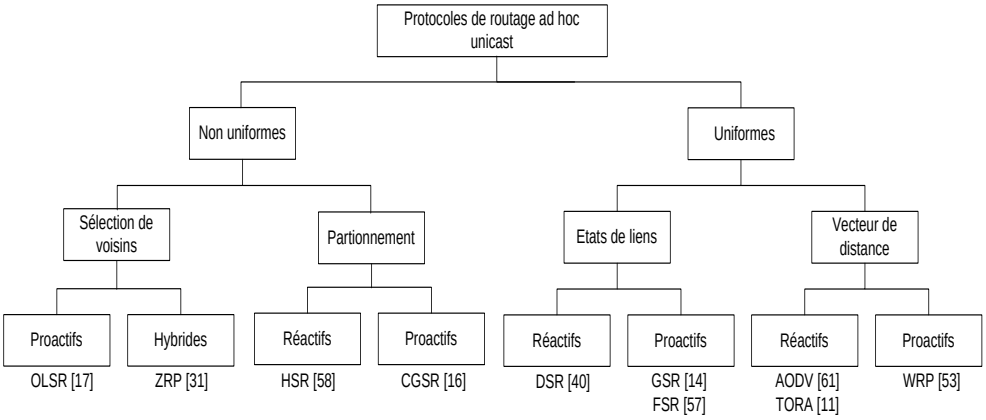


FIG. 2.6 – *Les protocoles de routage ad hoc*

routage différentes :

- AODV (*Ad hoc On demand Distance Vector*) est un protocole réactif, uniforme et orienté destination. La route retenue est bidirectionnelle et correspond au plus court chemin (en nombre de nœuds) entre la source et la destination. Chaque nœud maintient une table de routage dont les entrées mémorisent, pour une destination :
  - l'identifiant de cette destination,
  - l'identifiant du prochain nœud vers cette destination.,
  - le nombre de nœuds jusqu'à cette destination.

  La demande de route est diffusée par la source à travers tout le réseau. Celle-ci permet à tous les nœuds de mémoriser une route vers la source. Quand la destination reçoit cette demande, la transmission de sa réponse permet aux nœuds intermédiaires de mémoriser la route recherchée. Pour maintenir les routes, chaque nœud vérifie périodiquement la présence de ses voisins. De plus, un numéro de séquence date chaque requête ou réponse pour éviter les multiples traitements et actualiser les routes.
- OLSR (*Optimized Link State Routing Protocol* [17]) est l'autre proposition retenue par l'IETF pour le routage dans les réseaux ad hoc. OLSR est un protocole proactif qui repose sur l'échange régulier d'informations sur la topologie du réseau. L'algorithme est optimisé par la réduction de la taille et du nombre des messages échangés : seuls des nœuds particuliers, les MPR, *MultiPoint Relay* , diffusent des messages de contrôle sur la totalité du réseau.

Les définitions suivantes sont utilisées dans la description du protocole :

Nœud : hôte d'un réseau ad hoc implémentant le protocole OLSR.

Interface : point d'accès au réseau ad hoc. Un nœud peut avoir plusieurs interfaces, chacune ayant une adresse IP propre.

Voisin immédiat : le nœud X est un voisin immédiat du nœud Y si Y est à portée du nœud X (une des interfaces de X peut envoyer des messages sur l'une des interfaces de Y).

MPR (*MultiPoint Relay*) : nœud sélectionné par un de ses voisins immédiats (appelé MS, *MPR Selector*) pour retransmettre ses messages de mise à jour. L'ensemble des MPR d'un nœud est choisi parmi les voisins immédiats, de manière à permettre d'atteindre tous les nœuds situés exactement à 2 sauts.

Lien : couple d'interfaces capables de communiquer (i.e. recevoir des messages). L'état d'un lien peut être :

- symétrique *SYM*, si les deux interfaces peuvent s'entendre,
- asymétrique *ASYM* ou *HEARD*, si les nœuds ont des puissances d'émission différentes,
- MPR, si l'émetteur a sélectionné le nœud comme MPR, dans ce cas le lien doit être symétrique.
- Lost, quand le lien est perdu.

Tous les nœuds envoient périodiquement des messages HELLO à leurs voisins immédiats (temporisateur HELLO_INTERVAL) sur chacune de leurs interfaces. Ces messages ne sont pas relayés vers d'autres nœuds.

OLSR utilise un seul format de message, transporté par le protocole UDP. L'entête précise si le message doit être seulement transmis au voisinage immédiat ou bien à l'ensemble du réseau.

Chaque nœud garde en mémoire la description de son voisinage : interfaces voisines, voisins à 2 sauts, MPR et MS. Cette description est mise à jour à chaque réception d'un message HELLO, et les informations obsolètes sont effacées.

Le routage OLSR est basé sur l'acheminement des messages par les nœuds qui ont un voisinage symétrique. Un lien ne peut participer à une route que s'il est symétrique. Le routage vers les stations éloignées de plus d'un saut (1+N sauts) se fait grâce aux MPR, qui diffusent périodiquement des messages TC (Topology Control) contenant la liste de leurs MS. Un numéro de séquence permet d'éliminer les doublons. Ces messages servent à maintenir dans chaque station une table de la topologie.

La table de routage est construite et mise à jour à partir des informations contenues dans la table des interfaces voisines et la table de la topologie, en utilisant un algorithme de plus court chemin. La métrique prise en compte est le nombre de sauts.

2.1.5 Les vulnérabilités et les outils de protection des MANET

Dans cette section nous présentons en premier lieu une typologie des attaques et leurs conséquences pour les réseaux ad hoc. Puis nous présentons les principaux mécanismes de sécurité développés pour pallier ces attaques.

Les MANET sont vulnérables aux mêmes attaques que les autres types de réseaux. Celles-ci sont par contre plus aisées à mettre en œuvre car dans un réseau ad hoc, on ne peut ni contrôler l'accès au support de transmission, ni définir les limites du réseau.
Le protocole de routage, spécifique aux MANET et indispensable à son fonctionnement, constitue une cible privilégiée. La taxonomie de attaques contre les WLAN proposée dans [44] et représentée sur la figure 2.7 identifie deux familles d'attaques : les attaques passives et les attaques actives.

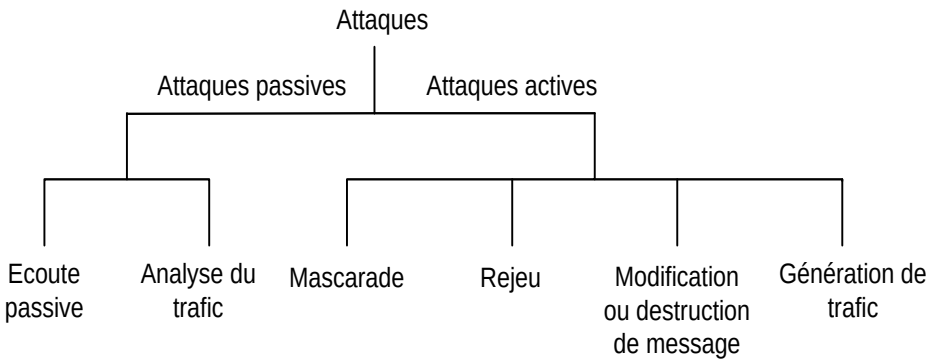


FIG. 2.7 – *Taxonomie des attaques contre les réseaux - WLAN*

Les attaques passives sont basées sur l'accès, par un nœud non autorisé, aux trames qui transitent sur le réseau pour collecter de l'information sans altérer les données échangées. L'écoute passive consiste à prendre connaissance du contenu des messages échangés entre deux nœuds. L'analyse du trafic consiste à déduire, en fonction des différents échanges réseaux, des informations sur l'organisation ou la configuration du réseau.

Les attaques actives ont pour objectif de permettre à un nœud autorisé de modifier un message, des données ou encore un flot de données. La mascarade est une attaque qui consiste pour un attaquant à se faire passer pour un autre utilisateur afin de bénéficier de ses droits d'accès aux différentes ressources.

Pour réaliser une attaque par rejeu, l'attaquant commence par enregistrer une séquence de trafic compatible avec la politique de sécurité et ensuite régénère cet échange à la place d'une des parties pour tromper l'autre. La modification partielle du contenu d'un message échangé entre deux entités, ou la génération de trafic non légitime sont aussi des techniques exploitées par des attaquants pour compromettre le fonctionnement d'un réseau.

Dans un réseau ad hoc les attaques peuvent être dirigées contre les services d'une station ou ceux du réseau. Dans ce dernier cas, elles ciblent principalement le protocole de routage afin de perturber les communications entre les nœuds [83],[36],[67]. Les principales conséquences de ces attaques, présentées dans [37], sont résumées ci-dessous :

- L'introduction d'une boucle de routage.
- La création d'un *trou noir* qui consiste à rediriger le trafic vers un nœud qui ne retransmet pas les informations.
- La division du réseau en plusieurs sous-réseaux afin de bloquer les échanges entre les nœuds appartenant à des sous réseaux différents.
- La non retransmission par un nœud de certains messages.
- L'arrêt d'un nœud en raison de son manque d'énergie.
- Le déni de service d'un nœud qui est empêché d'émettre ou de recevoir des paquets.

Pour pallier ces menaces, différents mécanismes de sécurité peuvent être mis en place. Présentés sur la figure 2.8, ils sont basés sur des algorithmes de cryptographie et appliqués à différents niveaux du modèle O.S.I pour assurer les services de sécurité.

Les normes IEEE 802.11 proposent d'utiliser le protocole WEP (*Wired Equivalent Privacy*) pour sécuriser les échanges. Malheureusement ce protocole souffre de défauts majeurs pour la mise en œuvre des services de sécurité :

- l'algorithme de hachage CRC32 pour contrôler l'intégrité;
- un mécanisme de chiffrement basé sur RC4;
- un mécanisme d'authentification unilatérale basé sur un challenge de 128 bits codé avec la clé partagée.

En attendant l'arrivée de la norme 802.11i[4], chargée d'apporter une sécurité optimale aux réseaux WI-FI[5], les constructeurs ont proposé une amélio-

4. http://grouper.ieee.org/groups/802/11/Reports/tgi_update.htm
5. http://www.wi-fi.org/OpenSection/index.asp?TID=1

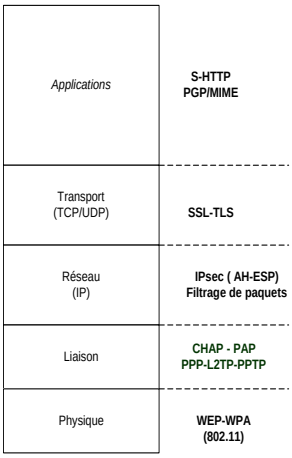


FIG. 2.8 – *Les principaux mécanismes de sécurité*

ration du WEP appelée WPA (*WI-FI Protected Access*). WPA, compatible avec les équipements existants, propose :

- différents modes d'authentification basés sur le protocole 802.1x/EAP [6] (*Extensible Authentication Protocol*) [9];
- l'utilisation d'une somme de contrôle d'intégrité MIC (*Message Integrity Control*);
- le renouvellement des clés de chiffrement par l'utilisation de TKIP (*Temporal Key Integrity Protocol*).

Les technologies de sécurité classiques appliquées à des niveaux supérieurs, comme les VPN (*Virtual Private Network*) pour le niveau 3 ou les protocoles de chiffrement SSL et SSH pour les applications, peuvent aussi être utilisées pour renforcer le niveau de sécurité. Ces dernières nécessitent des traitements locaux et des transferts réseaux supplémentaires et n'apportent pas de solution au problème posé par la distribution des clés dans un réseau ad hoc.

2.1.6 Les mécanismes de sécurité et la détection d'intrusions

Pour sécuriser un système, représenté par un ou plusieurs équipements informatiques connectés en réseau, la première étape consiste à établir la *politique de sécurité*. Celle-ci donne les règles d'application des *services de sécurité* pour protéger les *ressources* du système. Les ressources systèmes com-

6. http://www.ieee802.org/1/pages/802.1x.html

prennent les données hébergées localement, les applications accessibles aux utilisateurs, ainsi que l'ensemble des ressources matérielles locales, comme l'espace disque ou le processeur. Dans [56], l'ISO a défini les services de sécurité suivants :

- **L'authentification,** qui garantit l'identité de l'entité, de l'utilisateur ou du processus.
- **Le contrôle d'accès,** qui garantit le respect des droits d'accès aux services et données hébergés par le système.
- **La confidentialité des données,** qui garantit que l'information ne doit être ni rendue accessible, ni divulguée à un utilisateur, une entité ou un processus non autorisé.
- **L'intégrité des données,** qui garantit que l'information ne doit pas être altérée ou détruite de manière non autorisée.
- **La disponibilité des services,** qui garantit que l'accès aux services offerts par le système est possible.
- **La non répudiation,** qui garantit que toute entité responsable d'une action sur le système ne peut nier l'avoir effectuée.

Pour appliquer les règles définies dans la politique de sécurité, l'étape suivante consiste à mettre en œuvre les mécanismes de sécurité idoines sur l'ensemble des postes de travail et sur des équipements dédiés, comme par exemple un pare-feu ou un serveur d'authentification. Ces mécanismes de sécurité préventifs forment la première ligne de défense du système contre les différentes menaces.
On ne peut toutefois considérer cette protection comme absolue, permanente et incontournable. Une surveillance permanente du système protégé peut permettre de renforcer l'action des mécanismes de sécurité.

La surveillance d'un système, aussi appelée *audit de sécurité*, consiste à collecter des informations représentatives des actions réalisées sur le système. Dès 1980, J.P Anderson [3] a été le premier à montrer l'importance de l'audit de sécurité pour détecter des violations de la politique de sécurité appliquée à un système. Il propose d'utiliser les traces d'audits de sécurité pour construire un modèle statistique représentatif du comportement usuel d'un utilisateur du système afin d'en détecter toute action inhabituelle. Cette première approche qui consiste à détecter une déviation par rapport à un comportement normal préalablement défini est appelée *l'approche comportementale* ou encore *anomaly detection.*

Une autre approche consiste non plus à se référer aux comportements normaux et détecter les écarts par rapport à cette référence, mais à modéliser les menaces par des signatures et à rechercher la présence de ces signatures

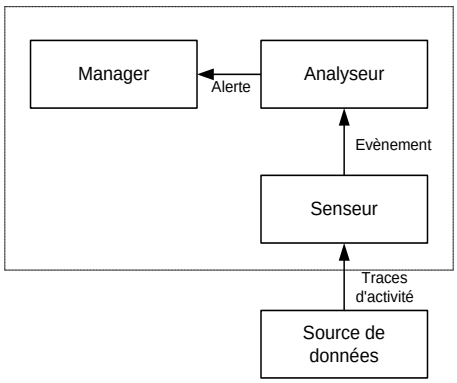


FIG. 2.9 – *Modèle d'architecture IDS-IDWG*

dans les traces d'audits. Cette approche est appelée *l'approche par scénarios* ou *misuse detection*. Dorothy E. Denning et al. [24] ont proposé le premier système de détection d'intrusion hybride, appelé IDES (*Intrusion Detection Expert System*)[23], regroupant les deux approches.

Dans la section suivante, nous présentons les différentes étapes du processus de détection d'intrusion ainsi que les caractéristiques des IDS.

2.2 Les systèmes de détection d'intrusions

Le groupe de travail IDWG (*Intrusion Detection Working Group*) de l'IETF a proposé dans [54] un modèle fonctionnel d'IDS, représenté sur la figure 2.9. Dans ce modèle, l'IDS est défini comme une combinaison, d'un ou plusieurs composants suivants : senseur, analyseur, manager. La fonction des différents composants et les messages échangés entre les composants sont aussi définis.

Le Senseur est le composant chargé de collecter les données brutes et de les mettre en forme pour transmettre des *événements* à l'*Analyseur*. Les données brutes sont collectées, par exemple, au niveau du réseau, dans les fichiers d'audit des applications ou du système d'exploitation.

L'Analyseur est le composant qui recherche, dans les événements transmis par le senseur, des signes d'activités indésirables ou non autorisées et génère les *alertes* destinées au *Manager*.

Le Manager est le composant qui permet de consolider les alertes et de valider les réponses à une intrusion.

Dans les implémentations, plusieurs composants peuvent être regroupés dans un seul module ou plusieurs instances des différents composants peuvent exis-

ter. Le groupe IDWG travaille aussi à la description du format des alertes, le format IDMEF (*Intruison Detection Message Exchange Format*)[7], ainsi qu'à celle du protocle d'échange d'alertes : le protocole IDXP (*Intrusion Detection eXchange Protocol*)[8].

2.2.1 Caractéristiques des IDS

Nous proposons une classification des IDS, inspirée de [22], basée sur les caractéristiques de cinq critères différenciateurs :

La source des données : Pour surveiller un système, il faut disposer de données significatives [3]. Ces dernières sont, par exemple, présentes dans les fichiers de logs, qui mémorisent les actions réalisées par le système d'exploitation ou les applications. Sur le réseau, on peut, à l'aide d'une sonde, collecter les données relatives aux échanges entre les systèmes. Si plusieurs IDS sont utilisés pour surveiller un ou plusieurs systèmes, les alertes générées par l'un des IDS peuvent être une source de données pour les autres IDS. Selon la source des données les informations générées sont différentes. Par exemple, les logs d'un système peuvent fournir des informations sur des tentatives d'abus de privilèges réalisées localement après une connexion alors que les données collectées sur le réseau peuvent fournir des informations sur des tentatives de connexions externes non autorisées.

La méthode de détection : Les méthodes de détection ont pour objet d' identifier, à partir des données collectées, les actions non conformes à la politique de sécurité. Ces méthodes sont présentées ci-après :

- L'approche par scénarios ou *misuse detection* identifie une intrusion par la présence dans les données collectées d'une suite d'événements, appelée *signature*, définie comme étant révélatrice d'une attaque connue. Cette méthode possède les inconvénients suivants :
  - les nouvelles attaques ne peuvent être détectées (faux négatifs) tant que leur signature n'a pas été établie.
  - la génération de fausses alarmes (faux positifs) quand les signatures ne sont pas suffisamment précises.
- L'approche comportementale ou *anomaly detection* identifie une intrusion par une déviation du *comportement* d'une entité par rapport au profil du comportement *normal* préétabli de cette entité. L'entité, dont le comportement est observé pour la détection, peut

7. www.ietf.org/ internet-drafts/draft-ietf-idwg-idmef-xml-10.txt
8. www.ietf.org/ internet-drafts/draft-ietf-idwg-beep-idxp-07.txt

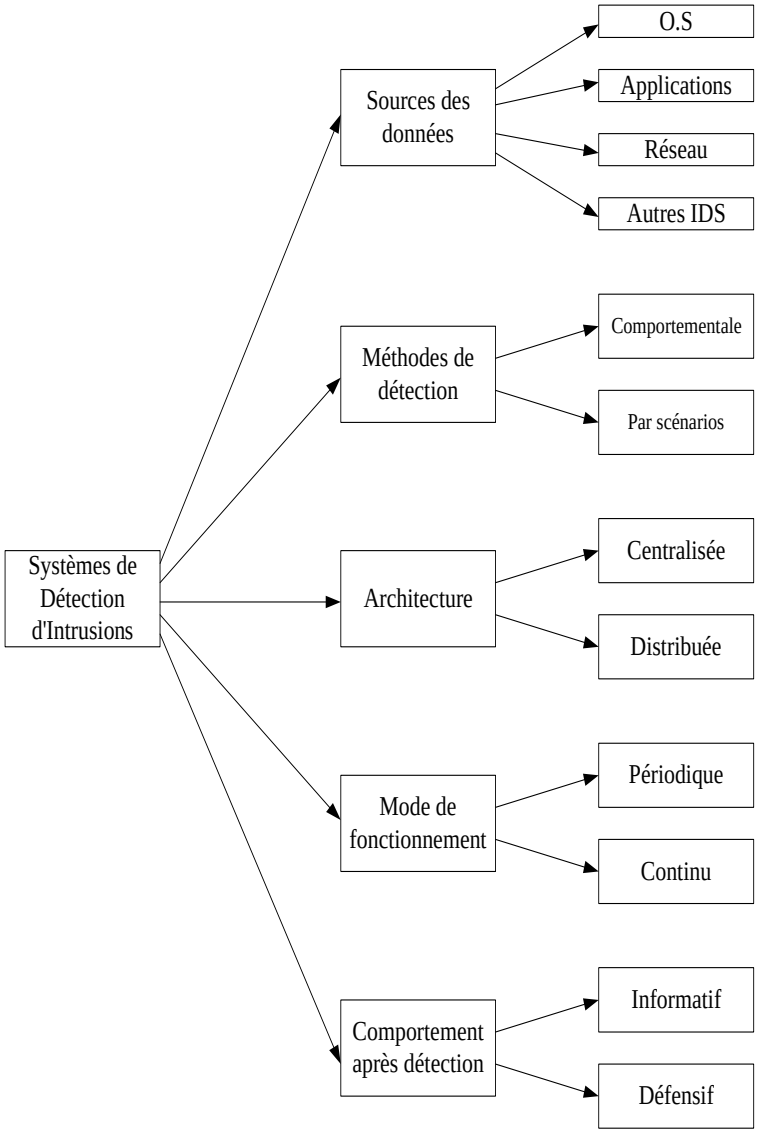


FIG. 2.10 – *Critères de classification des IDS*

être, par exemple, l'utilisateur, le système, ou le réseau. L'intérêt de cette approche est de permettre la détection d'attaques inconnues. Néanmoins, elle présente aussi quelques défauts :

- des faux positifs si le profil de l'entité observée est incomplet ou si des modifications brutales de l'environnement interfèrent sur ce profil,
- des faux négatifs si l'apprentissage du profil est faussé, par exemple, par une modification lente du comportement d'un utilisateur dans l'intention de faire apprendre au système un comportement intrusif.

L'architecture de l'IDS représente la façon dont sont réalisées ses principales fonctions. Par exemple, l'analyse des données collectées peut être réalisée sur un système central ou au contraire distribuée sur différents systèmes. Cette distinction est beaucoup moins évidente quand on se réfère à la collecte des données qui peut être effectuée sur différents systèmes avec, dans certains cas, une phase de pré-analyse locale. Néanmoins, nous pouvons considérer la topologie du système complet et rechercher si son fonctionnement requiert une organisation hiérarchique des différents équipements impliqués dans la détection pour qualifier l'architecture décentralisée. L'avantage d'une architecture centralisée est de regrouper des événements qui, localement, peuvent être considérés comme normaux, mais dont la corrélation peut révéler une attaque.

Le mode de fonctionnement dépend de la façon dont est effectuée l'analyse des données collectées. Cette analyse peut être effectuée de façon périodique ou encore en *temps réel*. Ce dernier mode de fonctionnement doit être privilégié pour minimiser le temps de réponse à une attaque et donc ses conséquences.

Le comportement après détection est caractérisé par la réponse de l'IDS face à l'attaque détectée. Soit la réponse se limite à une simple information, soit elle réagit à l'attaque en la neutralisant.

2.2.2 Les IDS distribués et coopératifs

Dans un réseau ad hoc, en l'absence de point de concentration permanent, la détection des intrusions, comme tous les autres services réseaux, doit aussi être distribuée sur l'ensemble des nœuds du réseau. Chaque nœud est autonome, et de ce fait, il ne peut s'appuyer que sur ses ressources propres pour détecter les intrusions dont il est la cible. La détection de certains types d'intrusions peut nécessiter la collecte d'informations complémentaires disponibles uniquement sur d'autres nœuds. Dans ce cas, les nœuds sont amenés

IDS	Origine des données	Type de détection	Prétraitement des données	Détection	Analyse temps réel	Type de réponse
AAFID [6]	Système	Scénarios	distribué	centralisée	Oui	Passive
DIDS [76]	Système/Réseau	Hybride	distribué	centralisée	Oui	Passive
DPEM [45]	Système	Comportementale	distribué	centralisée	Oui	Passive
GrIDS [77]	Système/Réseau	Hybride	distribué	centralisée	Non	Passive
CSM [80]	Système	Comportementale	distribué	distribuée	Oui	Active
JiNao [29]	MIB/Réseau	Hybride	distribué	centralisée	Oui	Passive
IDA [4]	Système	Scénarios	Agents mobiles	centralisée	Oui	Passive
SPARTA [47]	Système/Réseau	Scénarios	Agents mobiles	distribuée	Oui	Passive
MICAEL [21]	MIB	Scénarios	Agents mobiles	distribuée	Oui	Passive

TAB. 2.2 – *Principaux IDS distribués et leurs caractéristiques*

à coopérer pour s'échanger des données. Dans [81] et [2], Y. Zhang and W. Lee retiennent aussi la distribution et la coopération, comme principales caractéristiques des architectures d'IDS pour réseaux ad hoc.

Dans le Tableau 2.2, nous présentons les principaux IDS reconnus pour posséder une architecture à la fois distribuée et coopérative. Ces derniers ont atteint un niveau de développement permettant de valider les orientations retenues. Notre objectif est ici d'analyser leurs caractéristiques afin d'en évaluer la compatibilité avec les exigences des MANET.

Nous constatons pour la localisation de la détection, qu'à l'exception de CSM, MICAEL et SPARTA, le processus de détection nécessite une organisation hiérarchique autour d'un nœud central permanent. Selon nous, on ne peut dans ce cas parler de véritable IDS distribué. Seule la collecte des informations à analyser l'est.

Ces modèles d'architectures ne répondent donc pas aux exigences de distribution des MANET.

Nous considérons que l'architecture CSM (*Cooperating Security Managers*) est véritablement distribuée. Elle est composée d'un ensemble d'IDS locaux présents sur chaque système. Ces derniers sont dotés d'une fonction spécifique de suivi des connexions dont l'objectif est de détecter les tentatives d'intrusions initiées depuis une machine vers plusieurs autres machines du réseau. Cet IDS distribué, conçu et optimisé spécifiquement pour tracer

les connexions à travers un réseau filaire, n'est pas suffisamment généraliste pour être retenu tel quel ici.

Micael possède une architecture distribuée dans laquelle des IDS locaux installés sur chaque nœud sont chargés de détecter les intrusions localement. L'activité de ces IDS locaux, appelés *Sentinel* est contrôlée par un agent spécial, appelé *Head Quater*, hébergé sur l'un des nœuds. Cet agent a pour fonction de rassembler les informations collectées par tous les *Sentinels Agents* et de les assister dans le processus de détection.

Parmi les architectures citées ci-dessus SPARTA (*Security Policy Adaptation Reinforced Through Agents*) est la seule conçue spécifiquement pour les environnements sans fil. L'architecture de ce système, développé parallèlement au nôtre, nécessite la présence d'un nœud central dont le rôle est de maintenir une base de connaissance des nœuds présents dans le réseau.

Pour trois des IDS retenus, (IDA, SPARTA, MICAEL), le prétraitement à distance des données est effectué par des agents mobiles. Dans [19], un agent mobile est défini comme une entité logicielle qui fonctionne de manière autonome et continue dans un environnement particulier, capable de se déplacer et de s'adapter aux changements de l'environnement, de communiquer et de coopérer avec d'autres agents.

La mise en œuvre d'agents mobiles permet de déplacer le code vers les données à analyser. C'est une alternative aux architectures client/serveur. Du point de vue de la charge du réseau, cette solution de communication entre les nœuds est efficace si le code de l'agent est moins volumineux que celui des données à analyser.

Dans la section suivante, nous présentons le principe de fonctionnement des plates-formes à agents mobiles ainsi que leurs caractéristiques. Nous analysons aussi les apports potentiels de cette technologie pour réaliser la coopération et la distribution dans les réseaux sans fil ad hoc.

2.3 Les systèmes distribués et les agents mobiles

2.3.1 Les modèles de conception des systèmes distribués

Les *agents mobiles* [15] [64] constituent un modèle d'architecture des systèmes distribués au même titre que le *client/serveur* ou le *code à la demande*.

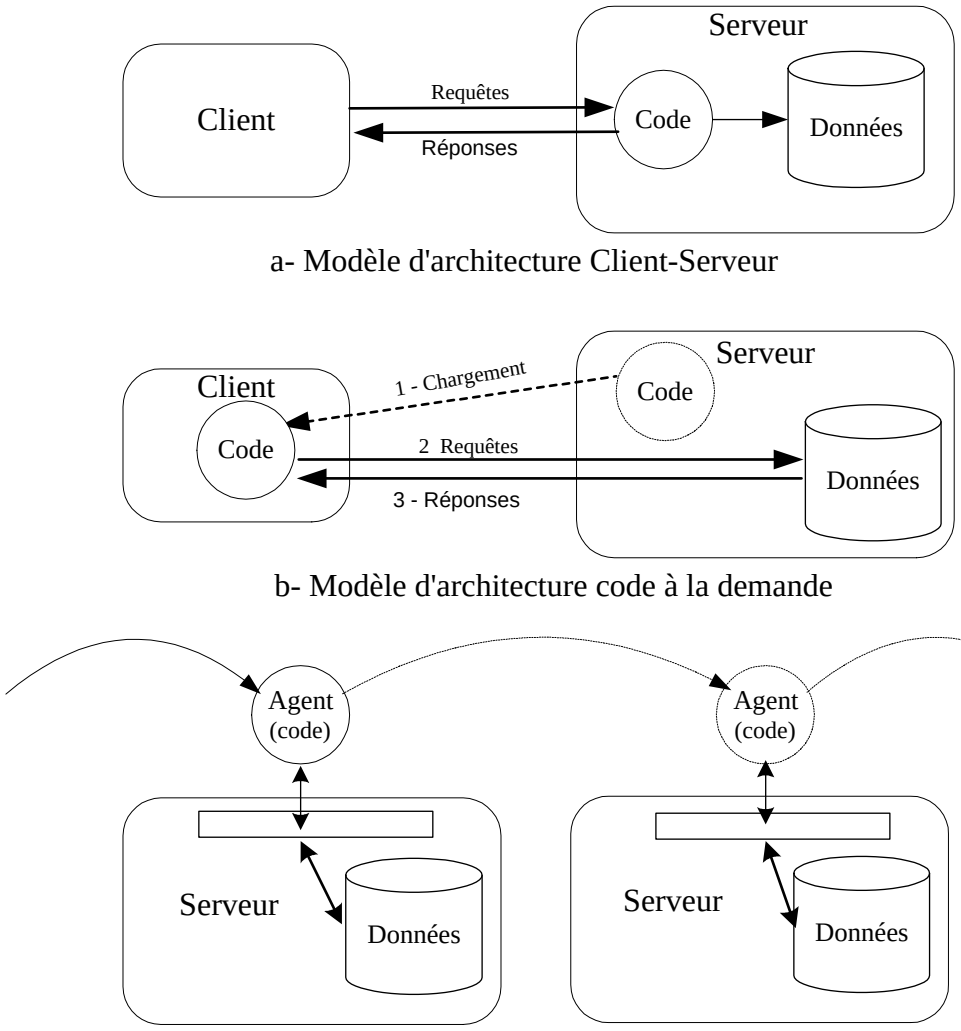


FIG. 2.11 – *Trois modèles de communications pour applications distribuées*

Dans une architecture client/serveur (Figure 2.11-a), le client effectue une requête d'exécution d'une partie de code au serveur qui lui retourne le résultat de cette exécution. Les RPC (*Remote Procedure Calling*), CORBA (*Object Requet Brokers*) et java RMI (*Remote Method Invocation*) sont les principales technologies utilisées pour la conception de ces architectures.

Dans une architecture distribuée basée sur *du code à la demande* (Figure 2.11-b), le client demande au serveur le module de code qu'il souhaite exécuter. Les *Applets* java constituent le modèle de code à la demande des

applications construites pour les serveurs et navigateurs WEB.

Dans une architecture à agents mobiles (Figure 2.11-c), le module de code qui constitue alors une partie de l'agent, n'est pas disponible via un serveur. Il est autonome et possède la capacité de se déplacer pour s'exécuter, selon ses besoins, sur les différents systèmes d'un réseau.

Avec le déplacement du code de système en système pour disposer des ressources nécessaires à leur exécution, les agents mobiles présentent la plus grande flexibilité des trois modèles présentés. Les deux autres modèles nécessitent notamment la présence permanente d'un serveur avec l'ensemble du code disponible pour pouvoir répondre aux demandes des clients.

Nous constatons que par leur principe même de fonctionnement, les architectures basées sur des agents mobiles sont adaptées au caractère spontané des réseau ad hoc. De plus, ils peuvent contribuer à réduire le volume des données échangées par le réseau. Dans [33] les auteurs comparent les caractéristiques des architectures client/serveur et les architectures basées sur des agents mobiles et concluent qu'aucun service ne peut être rendu par une seule de ces architectures. Ils relèvent, tout comme Kuhna Shah dans [75], que les agents mobiles sont plus flexibles que les autres techniques de codes mobiles et que dans certains contextes spécifiques les agents peuvent aussi présenter certains avantages :

- un agent mobile peut décider quand et où il doit se déplacer,
- un agent mobile peut se déplacer d'un nœud client vers un nœud serveur ou inversement,
- un agent mobile supporte les déconnexions entre le client et le serveur et de ce fait est plus performant lorsque les connexions réseaux sont instables,
- différentes applications distribuées peuvent être implémentées facilement et de façon fiable à partir d'une plate-forme à agents mobiles.

2.3.2 Définition et principes de fonctionnement des agents mobiles

Le terme *Agent Mobile* repose sur le concept d'agent et sur la fonction de mobilité. Un agent est un objet autonome doté de capacités d'apprentissage et d'interaction avec d'autres agents. La mobilité est la caractéristique des objets dotés d'une capacité de déplacement selon un itinéraire pré-défini ou en fonction des besoins. Dans [15], [64] et [19], un agent mobile est vu comme une entité logicielle, comprenant son code, son état courant et des données.

Au cours de son exécution, il est capable de se déplacer de machine en machine dans un réseau.
Un agent est autonome et possède un cycle de vie représenté par ses différents états tels que sa création, sa duplication ou *clonage*, son déplacement, sa suspension ou cessation d'activité, etc. La communication entre agents se fait généralement par échange de messages synchrones ou asynchrones (voir l'annexe D.2).

Une plate-forme à agents mobiles est, d'une façon générale, formée d'*agents mobiles*, d'*agences*, aussi appelées *places*, chargées d'accueillir les agents mobiles pour leur exécution sur les différents systèmes, et de *canaux de communications*.

La migration de l'agent à travers le réseau est dite forte quand celui-ci peut interrompre son exécution et la reprendre sur une autre place. Dans ce cas, l'agent devra se déplacer avec son état d'exécution sur la machine distante. Une migration faible est la caractéristique d'un agent qui reprend son exécution sur une méthode/fonction prédéfinie en arrivant sur une nouvelle place après un déplacement à travers le réseau. Dans ce cas, le déplacement de l'agent nécessite au minimum le déplacement du code. Pour pouvoir se déplacer de plate-forme en plate-forme et s'exécuter sur différents systèmes, les agents mobiles sont programmés en langages interprétés, comme Java ou Tcl[9] (*Tool Command Language*). Bien que possible, le déplacement de code binaire n'est pas utilisé car il entraînerait d'importantes contraintes pour les systèmes.

De par leur fonction, on distingue trois types différents d'agents mobiles :

- *Les agents notificateurs* préviennent le client lorsqu'un événement particulier se réalise.
- *Les agents itinérants* ont besoin de la collaboration de plusieurs systèmes pour accomplir leur mission. Ces agents privilégient les accès locaux aux ressources. Leur usage est particulièrement adapté dans les réseaux peu fiables ou possédant des bandes passantes faibles en regard du volume de données à traiter.
- *Les agents d'adaptation* fournissent des services à la demande. Ils offrent ainsi un moyen simple de mettre à jour les fonctionnalités d'un système selon le besoin des utilisateurs.

9. http://www.tcl.tk/software/tcltk/

2.3.3 L'apport des agents mobiles pour la distribution et la coopération

Nous limitons cette analyse aux caractéristiques qui différencient les agents mobiles des autres techniques utilisées dans les systèmes distribués pour réaliser des fonctions équivalentes. Les travaux présentés dans [47], [78], [50], [10] et [21] donnent les principaux avantages, résumés ci-dessous, des agents mobiles pour les MANET et la détection d'intrusions :

La réduction de la charge réseau résulte du fait que le réseau n'est sollicité que pour déplacer l'agent. De plus, le maintien de la connexion réseau n'est nécessaire que pendant le déplacement et non pendant toute la durée de l'interaction. Plus le débit du réseau est faible ou plus le nombre de données à consulter est important, plus l'avantage des agents mobiles est significatif.

L'asynchronisme est une propriété intrinsèque des agents mobiles qui permet à un utilisateur de déléguer une tâche à un agent sans rester bloqué en attendant le résultat. Ce mode de traitement est particulièrement adapté pour des liaisons réseaux peu stables comme celles des WLAN.

L'autonomie et l'intelligence permettent aux agents d'adapter leur comportement selon les informations collectées localement et éventuellement de poursuivre leurs déplacements au sein du réseau pour terminer leur mission sans revenir au point de départ après chaque déplacement. Une autorité de contrôle central n'est pas nécessaire pour contrôler le fonctionnement des agents. Cette capacité d'adaptation permet ainsi de réduire le trafic généré sur le réseau.

Le temps de réponse des applications et la charge réseau peuvent être réduits par le déplacement du code vers les données plutôt que le déplacement des données vers les applications.

La répartition de charge (*load balancing*) peut être obtenue en concevant une application distribuée comme un ensemble de composants indépendants représentés chacun par un agent mobile. La charge du système local peut dans ce cas être répartie sur plusieurs systèmes.

La personnalisation résulte des fonctions allouées à un agent, qui en se déplaçant sur un système peut augmenter ou mettre à jour ses fonctionnalités. De ce fait, un système peut rejoindre un réseau sans disposer au préalable de l'ensemble des services qu'il devra rendre.

La robustesse et la tolérance aux fautes résultent du fait qu'une application distribuée basée sur des agents mobiles peut se poursuivre alors qu'un ou plusieurs systèmes sont devenus inopérants ou que les liens du réseau sont coupés.

L'adaptabilité et la modularité (*scalability*) permettent de réagir face aux évolutions de l'environnement, notamment grâce la possibilité qu'offrent les agents mobiles de se multiplier et de se diffuser à travers un réseau pour maintenir un niveau de service constant quel que soit le nombre de systèmes connectés au réseau.

La portabilité résulte des langages de programmation (Java, Tcl) utilisés pour la conception des plates-formes à agents. Ceux-ci permettent le développement d'applications distribuées indépendant de leur contexte d'exécution.

L'interopérabilité des systèmes en réseau est renforcée par la mise en œuvre de protocoles d'échanges d'agents contrôlés par la plate-forme. Des travaux de normalisation des systèmes sont en cours aussi bien par l'OMG[10] (*Object Management Group*) que par la FIPA[11] (*Foundation for Intelligent Physical Agent*). La FIPA centre ses travaux sur les caractéristiques intrinsèques des agents comme la communication et l'interaction. L'OMG propose, à travers CORBA (*Common Object Request Broker Architecture*) et MASIF (*Mobile Agent System Interoperability Facilities*) [51], une base pour l'interopérabilité des différentes plates-formes.

La fiabilité des échanges entre les systèmes peut être renforcée si l'agent mobile adapte son déplacement en fonction de l'état de la connexion physique.

2.3.4 Les principales plates-formes à agents mobiles

La plupart des systèmes à agents sont des applications spécialisées, appelées plates-formes et développées en java. Elles offrent les ressources nécessaires au contrôle et à l'exécution des agents. Les principaux services fournis par une plate-forme à agents, présentés dans [64], sont :

- la portabilité,
- la mobilité,
- la sécurité,
- la communication,
- et le contrôle des agents.

L'utilisation du langage java pour le développement d'une plate-forme à agents permet de bénéficier de nombreuses librairies et des caractéristiques

10. http://www.omg.org/
11. http://wwww.fipa.org

intrinsèques au langage pour fournir les services offerts par la plate-forme. L'exécution des applications sur une machine virtuelle les rend *portables* et indépendantes des systèmes d'exploitation.
La mobilité du code peut tirer profit de la fonction de sérialisation qui consiste à utiliser un format intermédiaire pour le transport du code java. La mobilité d'un agent écrit en java est qualifiée de faible, car celui-ci ne peut se déplacer avec son contexte d'exécution. Les nombreuses fonctionnalités réseau du package *java.net* et le chargement dynamique des classes sont autant de ressources et fonctions impliquées dans la mobilité d'un agent.
La protection des systèmes impliqués dans les échanges d'agents mobiles représente aujourd'hui un frein pour le déploiement des plates-formes à agents. Les études en cours[12] devraient permettre d'aboutir rapidement à des solutions satisfaisantes. La *sécurité* des agents mobiles concerne quatre sous-systèmes interactifs :

- l'agent mobile, lui même, qui se déplace d'un nœud à l'autre,
- la plate-forme chargée d'accueillir et de contrôler l'exécution des agents mobiles sur un nœud,
- le nœud qui héberge la plate-forme,
- le réseau qui est impliqué dans le transport de l'agent.

Pour protéger les agents et le nœud, les plates-formes à agents mobiles [43], proposent différents services de sécurité, comme l'authentification, le contrôle d'accès et s'appuient sur les services de sécurité (*java Security Manager*) fournis par le langage java [7]. Par exemple, l'exécution d'un programme java dans un "bac à sable" (*sandbox*), permet de protéger les ressources du système.
Nous expliquons dans l'annexe D les principes de la *communication* par messages utilisée par les agents mobiles. Nous présentons aussi le cycle de vie d'un agent mobile et les méthodes impliquées pour *contrôler* ses changements d'états.

Dans le tableau 2.3 nous présentons les caractéristiques générales des principales plates-formes disponibles actuellement.

Dans ce chapitre intitulé *état de l'art* nous avons introduit et présenté les composantes nécessaires à la détection d'intrusions dans les MANET.

Après avoir situé le contexte spécifique de la détection d'intrusions dans les MANET nous présentons les caractéristiques des principaux IDS distribués et coopératifs. Nous présentons ensuite les modèles de communications des applications distribuées et les caractéristiques des architectures à agents mobiles. Le prochain chapitre a pour objet de présenter un modèle d'IDS

12. http://www.fipa.org

Plates-formes	**Aglet** [a]	**Concordia** [b]	**Voyager** [c]	**D'agents** [d]
Origine	IBM	Mitsubishi	ObjectSpace	Darmouth College
Support de mobilité	ATP(IBM) Sérialisation	Sockets TCP Sérialisation	Sérialisation	Sockets TCP - Capture d'état
Type de mobilité	Faible	Faible	Faible	Forte
Communication	Messages, RMI	Événements distribués	Messages, Événements distribués, RMI	Appel de méthodes, RPC
Conformité MASIF	Non	Non	Non	Non
Langage	Java	Java	Java	Java - Tcl
Sécurité	Sécurité java + Authentification serveur	Sécurité java + Authentification utilisateur	Sécurité java	Sécurité java - Safe Tcl

TAB. 2.3 – *Principales plates-formes à agents mobiles*

[a] Aglet : //www.trl.ibm.co.jp/aglets/
[b] Concordia : //web.vsisinc.com/concordia/documents/MobileAgentsWhitePaper
[c] Voyager : //www.objectspace.com/voyager/index.html
[d] D'agents : //ftp.cs.darthmouth.edu/rgray/transportable.html

distribué pour les réseaux sans fil ad hoc. L'analyse des IDS distribués et des modèles de distribution réalisée dans l'état de l'art nous permettra d'orienter la suite de nos travaux.

Chapitre 3

Un modèle d'IDS distribué pour les réseaux sans fil ad hoc

L'analyse des vulnérabilités et des outils de protection des MANET présentée dans le chapitre 2 montre la nécessité de mettre en œuvre des mécanismes de sécurité adaptés aux caractéristiques de leur architecture.

Nous commençons ce chapitre en proposant un modèle de sécurité pour protéger les nœuds. Celui-ci associe des mécanismes de sécurité préventifs et un système de détection d'intrusions. La suite de ce chapitre est centrée sur l'étude d'un modèle d'IDS pour MANET.

Pour établir les spécifications de l'IDS, nous recherchons d'abord les contraintes imposées par les MANET pour la détection d'intrusions. Nous proposons ensuite un modèle d'IDS distribué doté d'un mécanisme de coopération fiable. Nous terminons ce chapitre par une description des différents composants de l'IDS.

3.1 Le contexte de la détection d'intrusions dans les MANET

Nous avons proposé dans [59] un modèle de sécurité pour protéger les nœuds des MANET. Ce modèle est représenté sur la figure 3.1 de la page 52. Il associe les mécanismes de sécurité préventifs et un système de détection d'intrusions.
Les mécanismes de sécurité présentés dans ce modèle sont soit issus des outils de sécurité classiques et éprouvés, comme SSL ou IPSec, soit développés spécifiquement pour les besoins des MANET, comme MAE (*Manet Authen-*

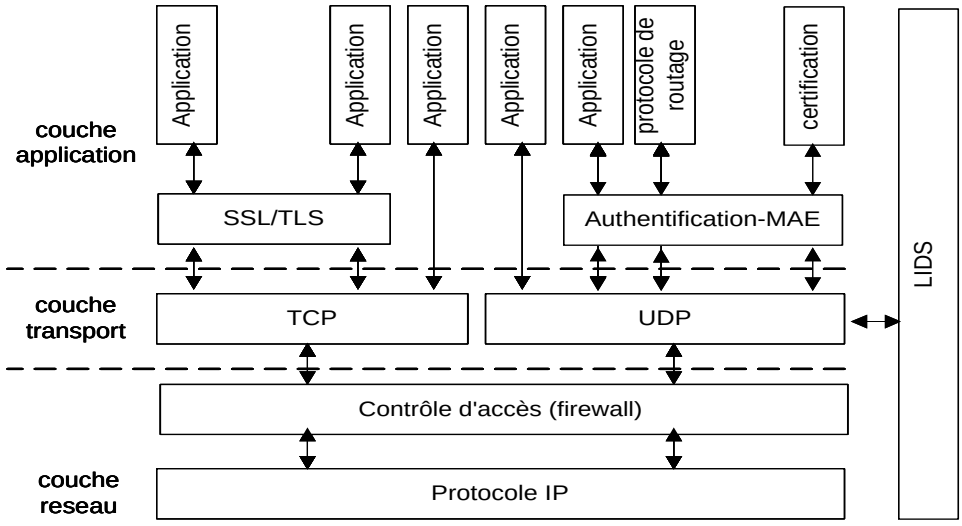


FIG. 3.1 – *Modèle de sécurité pour les MANET*

tication Extension) [70].
De par ses caractéristiques, le système de détection d'intrusions doit, lui aussi, être adapté aux contraintes des MANET.

Nous commençons par identifier les contraintes imposées par les MANET pour la détection des intrusions afin de pouvoir définir ensuite les spécifications de l'IDS.
Ces contraintes sont imposées par les nœuds, l'architecture dynamique du réseau, et les liaisons sans fil :

- *Les ressources limitées des nœuds :*
  Les nœuds d'un réseau ad hoc sont des systèmes portables et mobiles. De ce fait, leurs capacités sont limitées, notamment leur ressource en énergie, leur puissance de calcul et leur capacité de stockage.
  L'IDS doit donc être conçu pour limiter l'utilisation des ressources locales.
- *La flexibilité, la mobilité et les limites du réseau :*
  Le nombre et la position des nœuds, qui constituent l'infrastructure du réseau, évoluent selon les comportements des utilisateurs et le contexte d'utilisation. Il revient donc à chaque nœud d'assurer sa propre sécurité, et d'accorder une confiance mesurée aux autres nœuds. La technique de filtrage des paquets utilisée pour contrôler les accès aux points d'entrée d'un réseau filaire n'est pas applicable aux réseaux ad hoc. Ainsi, nous pouvons considérer que les attaques sont toujours internes.

L'IDS doit prendre en compte la dynamique permanente du réseau et l'absence de frontière identifiable entre l'extérieur et l'intérieur du réseau.

- *L'absence de système central permanent :*
  Dans un réseau ad hoc, il n'existe pas de nœud central permanent capable de collecter l'ensemble des informations nécessaires à la détection des intrusions. A un instant donné, celles-ci sont distribuées sur l'ensemble des nœuds actifs.
  L'IDS doit permettre la collecte d'informations situées sur les différents nœuds du réseau.
- *Les performances limitées du réseau :*
  Dans les réseaux sans fil, les performances, notamment en matière de débit, sont aujourd'hui en deçà des débits disponibles dans les réseaux filaires.
  Les échanges de données nécessaires à la détection des intrusions devront donc être réduits à leur strict minimum.
- *La politique de sécurité :*
  D'une façon générale un IDS se base sur une politique de sécurité, définie de façon explicite ou implicite, pour identifier les attaques. Pour les réseaux ad hoc, nous proposons de définir la politique de sécurité par rapport à des communautés d'utilisateurs.

3.2 Les caractéristiques d'un IDS pour MANET

Dans un MANET, la mobilité des nœuds, la distribution des fonctions et des données et les caractéristiques des connexions sans fil imposent des contraintes spécifiques pour la détection d'intrusions. A partir des caractéristiques générales des IDS, présentées dans la section 2.2.1 page 38, et des contraintes imposées par les réseaux ad hoc, nous établissons dans cette section les spécifications d'un IDS pour MANET.

Principes de détection : l'architecture de l'IDS doit être indépendante de la méthode de détection. La sélection d'un principe de détection ne pourra se faire qu'à l'issue de tests comparatifs.

Sources de données : l'architecture de l'IDS doit être indépendante de la source des données et répondre aux critères de portabilité et d'indépendance vis-à-vis des différents systèmes d'exploitation utilisés sur les nœuds.

Fréquence d'utilisation : la détection des tentatives d'intrusions doit se faire à l'exécution de l'attaque pour permettre aux utilisateurs de prendre les mesures nécessaires et renforcer ainsi l'action des mécanismes de sécurité.

Comportement après détection : sous le contrôle de l'utilisateur, la réponse doit être active en local pour accroître le niveau de sécurité et informative vers les autres nœuds présents dans le réseau.

Distribution des nœuds : l'architecture de l'IDS doit prendre en compte le caractère spontané des réseaux ad hoc ainsi que l'absence de nœud central permanent.

Débits limités des liens inter nœuds : les technologies WLAN offrent encore aujourd'hui des débits inférieurs à ceux des LAN. L'IDS doit s'appuyer sur les technologies les moins consommatrices de ressources réseaux.

Mobilité des nœuds : l'IDS doit posséder les mécanismes lui permettant de prendre en compte la mobilité des nœuds. Toutefois, différentes optimisations pourront être proposées selon les caractéristiques de mobilité et de densité des nœuds.

Normalisation : l'architecture de l'IDS doit adopter les normes actuelles notamment pour pouvoir coopérer avec d'autres IDS.

Nous présentons dans le tableau 3.2 les spécifications d'un IDS pour MANET et celles des IDS distribués présentés dans le Tableau 2.2, page 41. Ces derniers, exceptés SPARTA et CIDS-AHN [42] ne sont pas spécifiques aux réseaux sans fil. Leurs points communs sont un pré-traitement des informations à analyser dès leur collecte et à l'exception de CSM, conçu et optimisé pour un type d'attaque, une organisation fonctionnelle hiérarchique autour d'un nœud central permanent ou élu.

L'architecture de l'IDS SPARTA (*Security Policy Adaptation Reinforced Through Agents*) présentée par C. Krugel *et al.* dans [47] propose un langage de spécification d'événements EDL (*Event Definition Langage*) pour enregistrer sur chaque nœud des événements significatifs de l'activité du système et du réseau. La collecte de ces événements sur les différents nœuds est ensuite réalisée par des agents mobiles. Cette architecture est contrôlée par une application centrale (*management console*) installée sur un des nœuds. Sa fonction principale est de maintenir la base d'information des nœuds présents dans le réseau et impliqués dans le processus de détection des intrusions. Cet IDS proposé pour les réseaux sans fil disposant d'une infrastructure n'est pas dans l'état adapté aux MANET.

		IDS Ad Hoc	DIDS	GrIDs	CSM	DPEM	AAFID	JiNao	IDA	SPARTA	CIDS AHN
Données	Applications	✓				✓					✓
	Réseau	✓	✓	✓				✓		✓	✓
	Système	✓	✓	✓	✓	✓	✓		✓	✓	✓
Architecture	Sur chaque nœud	✓			✓						
Distribuée	Hiérarchique (fixe)		✓	✓		✓	✓	✓	✓	✓	
	Hiérarchique (élection)										✓
Réseau	Filaire		✓	✓	✓	✓	✓	✓	✓		
	WLAN	✓								✓	✓
	Ad hoc	✓									✓
Attaques ciblées	Types différents	✓	✓	✓		✓	✓		✓	✓	✓
	Spécifiques	✓			✓			✓			

FIG. 3.2 – *Caractéristiques des IDS Distribués*

Dans un MANET, un modèle d'organisation distribuée et hiérarchique autour d'un nœud ne peut être envisageable, en l'absence d'un nœud permanent, qu'avec des mécanismes supplémentaires de gestion de cette hiérarchie basée, par exemple, sur un processus d'élection de nœud central.
Cette approche est mise en œuvre dans l'IDS pour réseau ad hoc proposé par d'O. Kachirski et R. Guha dans [41] et [42]. L'architecture proposée est basée sur le regroupement des nœuds en *clusters* et l'élection dynamique d'un nœud à la tête de chaque *cluster*. La gestion des *clusters*, s'accompagne d'échanges réseaux supplémentaires et fait apparaître un nouveau type de vulnérabilités liées au processus d'élection du nœud central. En conséquence, nous ne souhaitons pas la retenir.

Les IDS que nous avons sélectionnés pour leur architecture distribuée ne sont pas, selon nous, adaptés au contexte de la détection d'intrusions dans les MANET.

3.3 Caractéristiques et positionnement de notre proposition

Dans un réseau ad hoc, l'architecture distribuée d'un IDS pourrait, selon nous, bénéficier de certaines caractéristiques des agents mobiles. L'analyse du comportement des agents mobiles réalisée dans le premier chapitre montre que dans certaines conditions ils peuvent contribuer à réduire le volume des données échangées sur le réseau. Nous pensons que l'autonomie des agents peut permettre de prendre en compte les changements de topologie et ainsi améliorer la fiabilité de la coopération entre les nœuds.

Nous proposons d'évaluer le comportement des agents mobiles pour réaliser la collecte et les pré-traitements des données situées sur les nœuds distants.

Dans les sections suivantes, nous établissons les spécifications des LIDS (*Local Intrusion Detection System*). Ensuite, nous montrons, par des simulations, comment les agents mobiles, permettent de renforcer la fiabilité de la coopération entre les LIDS.

3.3.1 Un modèle d'architecture distribuée, égalitaire et coopérative

Pour réaliser la détection d'intrusions au sein d'un MANET, nous proposons une architecture d'IDS distribuée, égalitaire, et dotée d'un système de coopération fiable.

Dans [2], nous avons présenté une nouvelle architecture d'IDS distribuée, représentée sur la figure 3.3, formée d'un ensemble d'IDS autonomes, que nous appelons LIDS, installés sur chaque nœud. En conséquence, l'architecture globale du système se forme et évolue en fonction de l'arrivée, du départ et du déplacement des nœuds dans le réseau.

L'instabilité des connexions physiques nécessite de doter le système global d'un mécanisme de coopération inter-LIDS fiable et robuste pour permettre aux LIDS de collecter des données disponibles sur les nœuds distants. Nous proposons de réaliser cette coopération au moyen d'agents mobiles. Les messages d'alertes générés par les LIDS dans le réseau représentent une source d'information supplémentaire à traiter de façon spécifique.

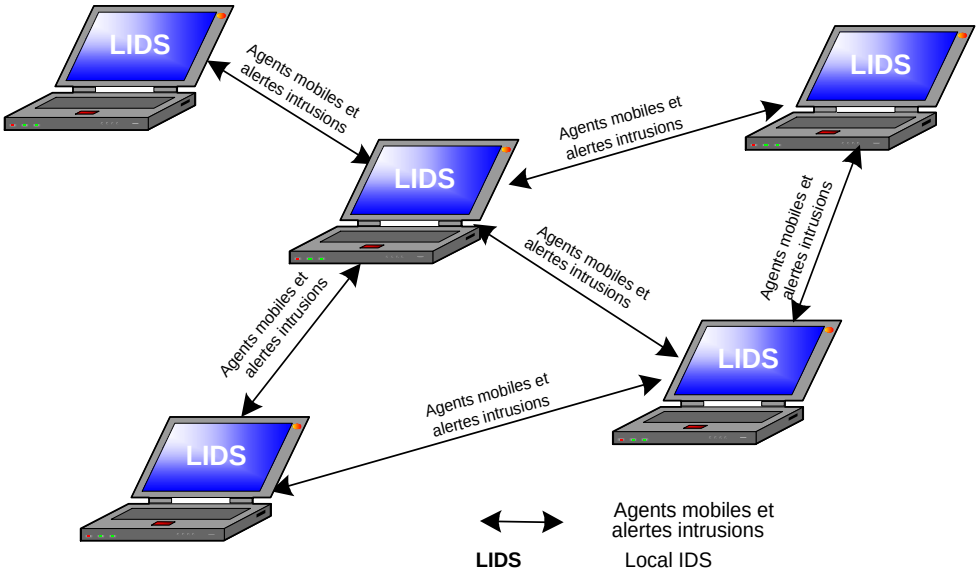


FIG. 3.3 – *Architecture globale de l'IDS distribué*

3.3.2 Caractéristiques des LIDS

Dans le réseau ad hoc, chaque LIDS constitue un IDS à part entière pour son hôte. A ce titre, pour établir ses caractéristiques, nous adoptons le modèle générique des IDS, proposé par l'IDWG, représenté sur la figure 2.9, page 37, et nous reprenons les caractéristiques des IDS présentées sur la figure 2.10, page 39.

Le senseur : ce module a pour fonction de collecter les informations nécessaires à la détection des intrusions. Le modèle distribué et coopératif que nous proposons exploite en priorité les informations issues du système local mais doit aussi avoir la capacité de collecter des informations complémentaires sur des hôtes distants.

Des modules logiciels additionnels, comme la bibliothèque *LibpCap* disponible pour différents types de systèmes d'exploitation, permettent de collecter les informations brutes échangées sur le réseau. Dans ce cas, les informations contenues dans les trames collectées nécessitent des traitements supplémentaires pour être extraites, classées et archivées dans un format directement exploitable par l'algorithme de détection. Cette approche, utilisée par le NIDS SNORT[1], ne permet pas de prendre

1. //www.snort.org

en compte les informations directement liées aux activités locales du système d'exploitation et des applications. Ces dernières sont généralement disponibles dans des fichiers de logs, dont le format dépend du système.

Pour disposer d'information dans une sémantique indépendante de sa source, nous proposons d'utiliser les données des MIB (*Management Information Base*) [32]. La représentation des informations dans un format *standard* n'est pas le seul avantage des MIB [68]. Leur utilisation comme source d'information pour la détection des intrusions a déjà été validée dans [12],[79] et [21]. L'extension de la branche *Experimental* de la MIB_II permet de collecter, si nécessaire, des informations complémentaires à celles déjà disponibles ou spécifiques à certaines applications comme le protocole de routage ad hoc.

Toutefois, si l'utilisation des données de la MIB comme source d'information permet d'homogénéiser la syntaxe des données collectées sur les différents systèmes, le protocole de transport UDP (*User Datagram Protocol*) et l'authentification basée sur les noms des communautés constituent, dans l'état, des défauts majeurs pour la fiabilité et la sécurité d'un système coopératif impliquant le protocole SNMP.

Nous proposons deux orientations complémentaires pour contourner ces inconvénients inhérents à l'usage du protocole SNMP :

- L'accès SNMP uniquement local aux données des MIB permet de s'affranchir des attaques dirigées contre le protocole si ces dernières sont mises en œuvre à partir des informations collectées, par écoute passive, dans des paquets réseaux. En particulier, le nom de la communauté, utilisé pour l'authentification, ne transite plus sur le réseau.
- L'accès aux variables situées sur les systèmes distants est pris en charge par le système de coopération des IDS, sans impliquer le protocole UDP qui n'apporte aucune garantie sur l'acheminement des données collectées.

L'analyseur : ce module a pour fonction d'exploiter les données collectées pour identifier les violations de la politique de sécurité. A notre connaissance, il n'existe pas d'étude comparative permettant de montrer qu'une méthode de détection est plus performante qu'une autre dans le contexte spécifique des réseaux ad hoc.

Dans les réseaux ad hoc, la comparaison des différentes méthodes de détection devrait, selon nous, pouvoir être réalisée de façon expérimentale. Pour tester, ultérieurement, différents algorithmes de détection sans re-

mettre en cause l'architecture complète, nous adoptons une conception modulaire des LIDS.

Le manager : il a pour pour fonction de traiter les alertes locales ainsi que celles reçues des autres IDS. La réponse du LIDS à une intrusion doit être contrôlée par l'utilisateur. Elle sera, si possible, active sur le système local pour accroître le niveau de sécurité et informative vers les autres nœuds présents dans le réseau.

En résumé, les choix d'orientations que nous venons d'analyser ainsi que les caractéristiques d'un IDS pour MANET présentées dans la partie *État de l'art* nous permettent de retenir les propriétés suivantes pour les LIDS.

Architecture interne : elle doit être modulaire et respecter le schéma général de l'IDWG. Chaque module doit pouvoir être optimisé par des expérimentations sans remettre en cause l'ensemble de l'architecture.

Principes de détection : l'architecture de l'IDS doit être indépendante de la méthode de détection.

Sources de données : l'architecture de l'IDS doit être indépendante de la source des données et répondre aux critères de portabilité et d'indépendance vis-à-vis des différents systèmes d'exploitation.

Fréquence d'utilisation : la détection des tentatives d'intrusions doit se faire à l'exécution de l'attaque.

Comportement après détection : sous le contrôle de l'utilisateur, la réponse doit, lorsque c'est possible, permettre d'améliorer le niveau de sécurité du système cible de l'attaque. La détection d'une tentative d'intrusions doit aussi permettre de générer une alerte vers les autres systèmes.

Coopération des LIDS : les LIDS doivent s'appuyer sur une architecture de communication fiable et sécurisée pour collecter des données présentes sur d'autres nœuds.

Charge réseau : la coopération entre les LIDS doit limiter la charge réseau induite.

Normalisation : l'architecture de l'IDS doit adopter les normes et standards afin de pouvoir coopérer avec d'autres IDS.

Portabilité : les LIDS doivent être multiplate-forme (Matérielle et logicielle).

Configuration : les paramètres de configuration doivent être indépendants du code.

Fiabilité-robustesse : la défaillance d'un LIDS ou du système qui l'héberge ne doit pas perturber le fonctionnement des autres LIDS présents dans le réseau.

Parmi toutes ces caractéristiques, la fiabilité de la coopération entre les LIDS installés sur chaque nœud est une composante principale et spécifique du modèle d'IDS distribué que nous proposons pour les MANET.

3.3.3 Une coopération fiable et robuste entre les LIDS

Les LIDS sont des systèmes de détection d'intrusions autonomes chargés de détecter les tentatives d'attaques dirigées contre le système qui les héberge et les services du réseau. Cette détection s'appuie sur la collecte d'information sur le système local ou sur les systèmes distants présents dans le réseau au moment de l'attaque et pendant sa détection. Quand ces données sont présentes sur des nœuds distants, le mécanisme de coopération entre les LIDS doit permettre de collecter ces données d'une manière fiable et adaptée aux contraintes inhérentes aux caractéristiques des réseaux sans fil ad hoc. Pour cela, ce mécanisme de coopération doit s'affranchir de l'instabilité des connexions physiques et contribuer, si possible, à réduire le volume de données échangées entre les LIDS.

Dans un réseau IP (*Internet Protocol*)[65], la fiabilité des échanges entre applications communicantes est prise en charge le protocole TCP (*Transmission Control Protocol*) [66]. TCP est un protocole de transport en mode connecté dont les objectifs sont d'assurer la fiabilité des données échangées et de contrôler les flux des données et les congestions du réseau.

Dans cette section nous analysons et évaluons le comportement de ces mécanismes dans les réseaux ad hoc. Nous terminons cette section par une évaluation du comportement et des performances des agents mobiles dans les réseaux ad hoc.

3.4 Comportement du protocole TCP dans les réseaux sans fil ad hoc

La première spécification du standard de TCP a été réalisée en 1981 par J. Postel dans le RFC 793. TCP y est décrit comme un protocole de transport de bout en bout, orienté connexion entre deux stations interconnectées par un réseau IP. Le service offert aux applications par le protocole TCP est vu comme le transfert d'un flux continu d'octets, acheminé de façon fiable et au mieux des capacités du réseau.

Nous présentons dans l'annexe H, les mécanismes mis en œuvre par le protocole TCP pour assurer la fiabilité des échanges de données et évaluons

dans la section suivante le comportement de TCP dans un réseau sans fil ad hoc.

3.4.1 Le comportement du protocole TCP dans les réseaux sans fil ad hoc

Les résultats présentés dans [35], [73] et [5] montrent que l'instabilité des liaisons sans fil perturbe fréquemment le fonctionnement du protocole TCP et de ce fait entraîne des dysfonctionnements pour les applications communicantes.

TCP considère que toute perte d'octets est due à une congestion et réagit par une réduction de la fenêtre de congestion (*cwnd*). Nous présentons, dans l'annexe H, les principes de contrôle de la fenêtre de congestion.

Pour évaluer la fiabilité du protocole TCP dans un réseau sans fil ad hoc, nous analysons sa capacité à ouvrir une connexion et à maintenir cette connexion opérationnelle dans un réseau dont les liaisons physiques sont instables.

Dans une première phase, nous mesurons sur la plate-forme de tests les délais d'attente des acquittements TCP pendant les phases d'ouverture et de transfert des données. Les mesures sont réalisées avec les systèmes Linux Suse 8.0 et MS-Windows 2000 et les applications FTP et Telnet. Dans une seconde phase, nous simulons un scénario de transfert de données dans un réseau ad hoc formé d'un nombre variable de nœuds mobiles. Nous mesurons, en fonction du nombre nœuds présents dans le réseau, les délais d'établissement de la route ainsi que la durée de ses ruptures. Nous comparons ensuite ces valeurs aux mesures réalisées dans la première phase.

3.4.2 Mesures des délais d'attente des acquittements TCP

Nous relevons à l'analyseur réseau le nombre et les dates des réémissions des segments SYN et des segments de données non acquittés afin de mesurer les délais maximum des rémissions. Le principe d'ouverture d'une connexion TCP, basé sur l'acquittement des segments SYN, est présenté dans l'annexe H.

- **Délais d'acquittement d'un segment SYN :** nous constatons que les demandes d'ouvertures sont générées par TCP suite à l' exécution d'une commande Telnet ou FTP. Dans un environnement Linux, après 5 réémissions du segment SYN, soit après 189 secondes, la couche TCP suspend le processus d'ouverture de connexion et informe l'application.

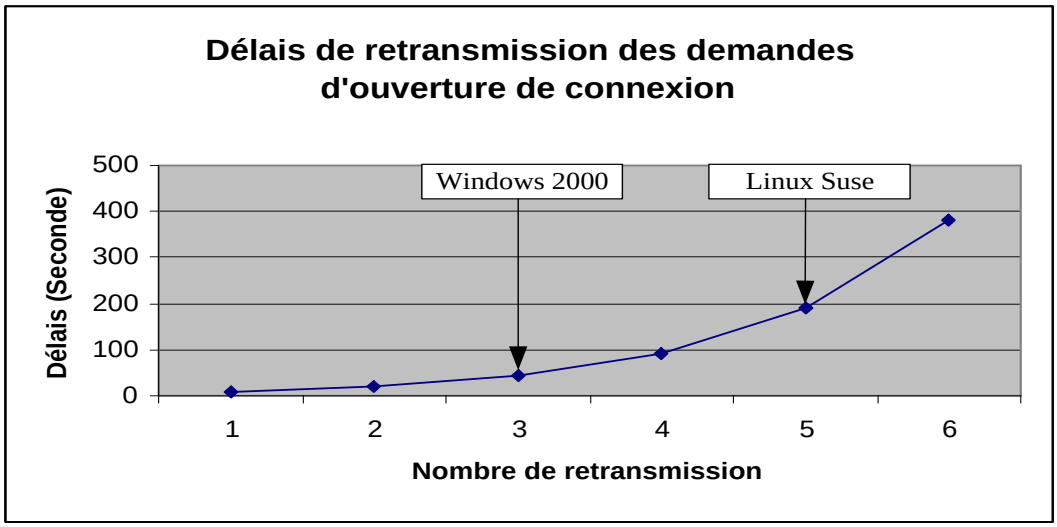


FIG. 3.4 – *Délais de retransmission des segments SYN non acquittés*

La valeur de la temporisation d'ouverture d'une connexion est fixée par la valeur du paramètre : *tcp_retries1*, et le nombre de retransmissions par celle du paramètre : *tcp_syn_retries*. Sur les systèmes de la plate-forme de tests, les valeurs par défaut de ces paramètres sont respectivement fixées 3 et 5. En fixant la valeur de ce dernier paramètre à 4, les réémissions sont suspendues après 93 secondes. Le système Windows 2000 suit le même processus, sur notre plate-forme les réémissions sont suspendues après 21 secondes.

Les évolutions des délais de réémission à l'ouverture d'une connexion sont représentées sur la figure 3.4, page 62.

– **Délais d'acquittement d'un segment de données :** nous analysons ici la retransmission des segments de données non acquittés. Nous constatons que l'application FTP ferme la connexion TCP si un segment n'a pas été acquitté au bout de 60 secondes, quel que soit l'environnement d'exécution de la commande (Linux ou Windows). Pour l'application Telnet exécutée dans l'environnement Linux, TCP libère la connexion au bout de 214 secondes. Ce qui correspond à une valeur de 9 pour le paramètre *tcp_retries2*. En fixant ce paramètre à la valeur 7, nous constatons que la connexion est interrompue après 53 secondes. Dans l'environnement Windows 2000 la connexion est fermée après une attente de 611 secondes. Les évolutions des délais de retransmission, dans les environnements Windows et Linux sont représentés sur la figure 3.5, page 63.

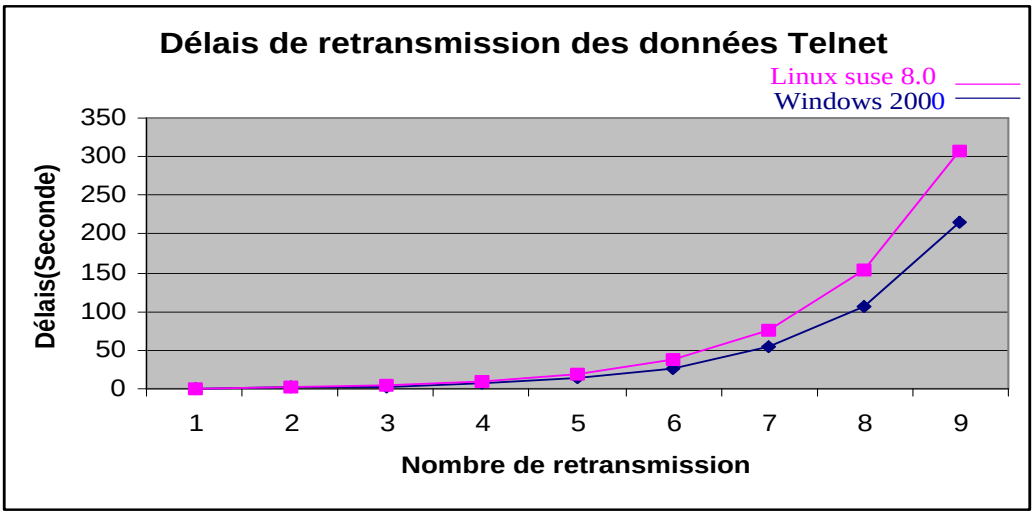


FIG. 3.5 – *Délais de retransmission des segments de données Telnet non acquittés*

3.4.3 Simulation des connexions TCP

Nous analysons ici les conséquences des ruptures d'un chemin, engendrées par le déplacement des nœuds, sur le délai d'établissement d'une connexion entre un client et un serveur. Nous disposons de deux outils de simulation réseaux : Qualnet et NS-2. L'outil Qualnet [2] est un produit commercial spécialisé pour la simulation des réseaux sans fil et NS-2 [3] est un logiciel *open-source* de simulation réseaux. Les contributions des membres de la communauté *open-source* permettent de disposer de nombreux modèles de protocoles, y compris de protocoles de routage ad hoc.

Nous utilisons le simulateur NS-2 avec les paramètres de scénarios suivants :

- réseaux 802.11 en mode ad hoc. La limite d'une connexion point à point est fixée à 250 mètres,
- établissement d'une connexion entre des nœuds client et serveur fixes,
- distance entre les nœuds client et serveur : 500 mètres,
- nombre de nœuds mobiles entre le client et le serveur : variables de 2 à N,
- type de mobilité des nœuds : RamdonWayPoint. (Vitesse de déplacement de 0,5 à 1,5 m/s - arrêts et changement de direction),

2. Scalable Network Technologie http://wwww.scalable-networks.com
3. Network Simulator-ns-2 http://www.isi.edu/nsnam/ns/

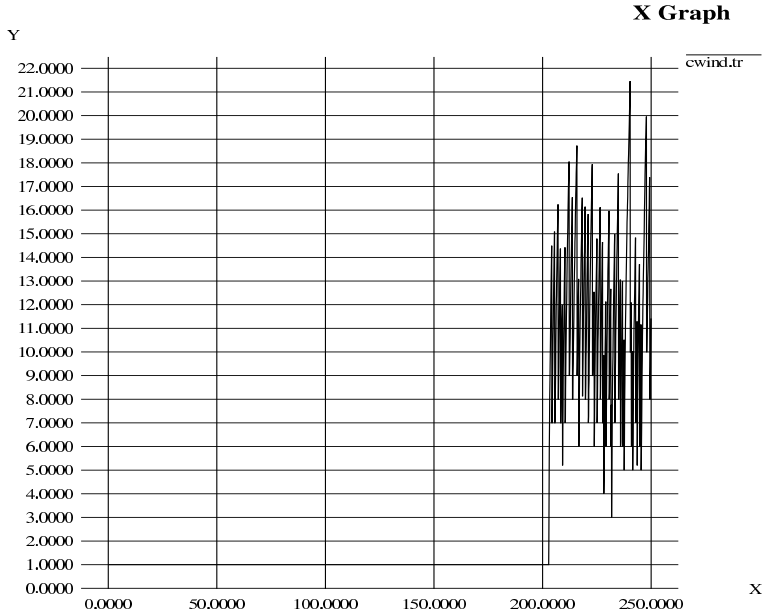


FIG. 3.6 – *Évolution de la fenêtre de congestion, cwnd, lors d'un transfert FTP. Y: cwnd (Segment) - X: t (Seconde)*

- aire de déplacement : 750 m X 750 m,
- protocoles de routage : DSDV - AODV.

Le modèle TCP NewReno, que nous utilisons pour nos simulations, est un modèle simplifié du protocole TCP. Il permet le transfert de données dans un seul sens, avec une gestion simplifiée des ouvertures de connexions. Le modèle TCP NewReno, implémenté sur le simulateur NS-2 commence les transferts de données dès qu'une connexion physique peut être établie, sans prendre en compte les délais de retransmission des segments SYN à l'ouverture de la connexion. La figure 3.6 de la page 64 représente l'évolution de la fenêtre de congestion, lors d'un transfert FTP entre deux nœuds commencé au temps t=10 secondes et une connexion physique établie après 200 secondes.

Nous représentons sur la figure 3.8 de la page 66 l'évolution de la fenêtre de congestion, avec 8 nœuds mobiles et le protocole de routage DSDV. Nous constatons que le transfert FTP entre deux nœuds est interrompu du temps t=90 secondes au temps t=380 secondes en raison de l'absence de route entre la source et la destination pendant cette période. Après t=380 secondes la position des nœuds permet à nouveau d'établir la route pour achever le transfert des données.

La figure 3.7 de la page 65 représente le temps d'établissement d'une route de bout en bout en fonction du nombre de nœuds mobiles présents dans le

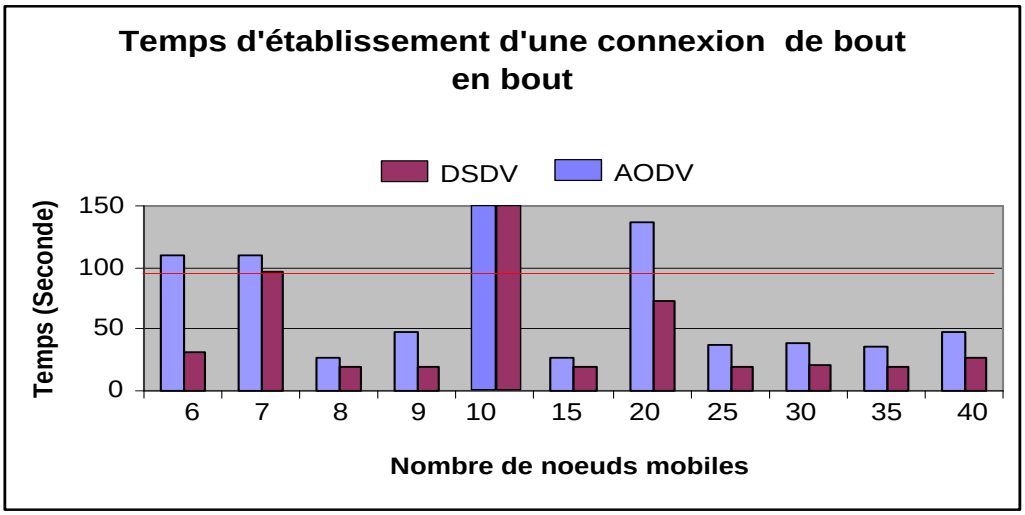


FIG. 3.7 – *Temps d'établissement d'une route, en fonction du nombre de nœuds mobiles*

réseau lors de la demande d'un transfert FTP.

3.4.4 Conclusions sur le comportement de TCP dans les réseaux ad hoc

Les résultats des simulations, réalisées avec différentes configurations du réseau, mettent en évidence que dans un réseau ad hoc un échange de type client/serveur basé sur une connexion TCP de bout en bout ne peut garantir la fiabilité et les performances des applications.

L'instabilité des connexions physiques a des conséquences sur le comportement des applications :

- Lorsqu'un client souhaite échanger des informations avec un serveur, le protocole TCP doit disposer d'une route pour initier une connexion avec le serveur. La figure 3.7 de la page 65 montre qu'en fonction du nombre de nœuds, et du protocole de routage, le délai maximal (représenté par la ligne rouge) fixé à 93 secondes pour l'ouverture de la connexion TCP peut être écoulé avant qu'une route ne soit disponible. Dans cette situation, seule l'application cliente peut prendre l'initiative de renouveler sa demande de connexion.
- Lorsqu'une route de bout en bout est établie, nous constatons, sur la Figure 3.8, que des interruptions dans les échanges entre les applications peuvent apparaître même si le protocole TCP conserve le contrôle

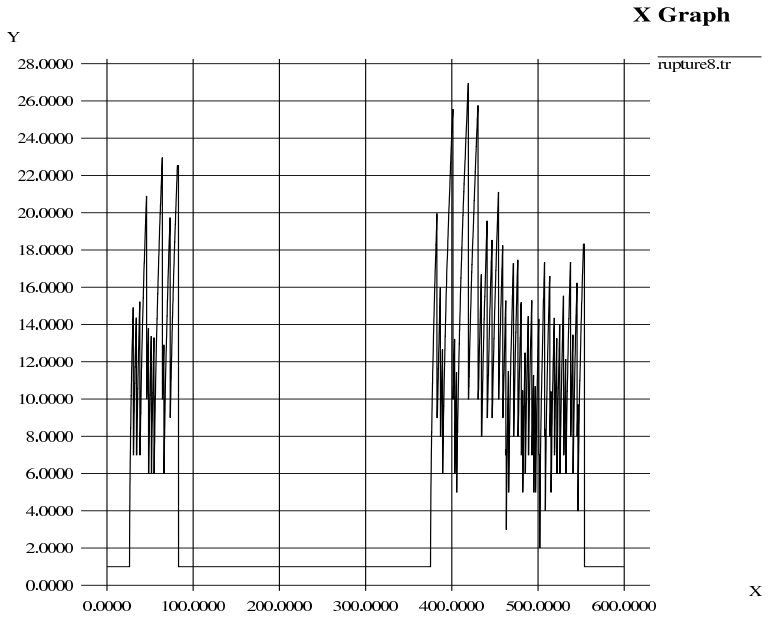


FIG. 3.8 – *Variation de la fenêtre de congestion pendant une phase de transfert. Y : cwnd (Segment) - X : t (Seconde)*

de la connexion. Dans cette situation, les applications peuvent se bloquer jusqu'au rétablissement d'une route de bout en bout, ou encore interrompre la connexion. Nous avons vérifié, par exemple, que l'application FTP ferme la connexion TCP quand les segments de données ne sont pas acquittés dans les 60 secondes qui suivent leur émission.

– Lorsqu'une route ne peut être établie entre le client et le serveur, cette situation apparaît dans le scénario avec 10 nœuds mobiles (figure 3.7 65), les échanges entre les applications restent impossibles jusqu'à ce qu'une nouvelle organisation des nœuds du réseau permettent d'établir une route de bout en bout. Pour s'affranchir de cette situation, l'application cliente devra renouveler périodiquement sa demande de connexion afin de bénéficier d'une configuration favorable du réseau.

En résumé, sur la base des mesures et des simulations réalisées, nous constatons que, dans un réseau ad hoc, le comportement et les performances d'une application client/serveur basée sur une connexion TCP dépendent :

– du nombre et de la mobilité des nœuds,
– de la valeur des paramètres (*TimeOut*) du protocole TCP,
– des fonctionnalités de l'application client/serveur,

- des performances du protocole de routage,
- des conditions de propagation.

Dans ces conditions, pour assurer une coopération fiable entre les systèmes, deux orientations sont selon nous envisageables :

- confier à l'application client/serveur le contrôle du mécanisme de coopération, par exemple pour renouveler les demandes d'ouverture de connexion quand celles-ci ont été abandonnées par TCP.
- confier à une application indépendante la gestion de la coopération.

La première proposition nécessite d'ouvrir une connexion TCP de bout en bout entre les applications, cliente et serveur. Les simulations réalisées montrent que l'application doit s'adapter aux comportements de TCP pour garantir des échanges fiables entre le client et le serveur. Une alternative à cette proposition consiste à utiliser un protocole TCP spécifique pour les réseaux ad hoc comme celui présenté dans [38]. Cette dernière solution réduit selon nous l'interopérabilité entre les systèmes, aussi nous ne souhaitons pas la retenir.

Dans l'architecture du système de détection, nous différencions la fonction de détection, réalisée sur chaque système par un IDS, et la fonction de coopération considérée ici comme un service, adapté au contexte des réseaux ad hoc, accessible et partagé par tous les LIDS. Nous proposons d'utiliser des agents mobiles pour réaliser la coopération entre les LIDS.

Dans la section suivante nous comparons, en fonction de la mobilité des nœuds le comportement d'une connexion client/serveur et d'un agent mobile qui se déplace du client vers le serveur.

3.5 Des agents mobiles pour une coopération fiable entre les LIDS

Dans cette section, nous montrons comment des agents mobiles peuvent contribuer à augmenter la fiabilité des échanges entre les nœuds d'un réseau sans fil ad hoc. A partir de différents scénarios de simulation, nous comparons le comportement d'un agent mobile qui se déplace de nœud en nœud et celui d'une connexion client/serveur établie de bout en bout.

Pour être opérationnelle, une application client/serveur doit établir et maintenir une connexion de bout en bout entre la source et la destination. Un agent mobile peut, lui, établir des connexions entre des nœuds voisins et

se déplacer de nœud en nœud pour atteindre sa destination finale.
Nous donnons ci-dessous les caractéristiques des agents mobiles qui, selon nous, peuvent contribuer à améliorer la fiabilité des échanges :

- leur mode de déplacement de nœud en nœud avec des ouvertures de connexion entre des nœuds voisins sont plus faciles à établir et à maintenir que les connexions établies de bout en bout par les applications client/serveur. Chaque nœud participe au routage et possède des informations relatives au chemin pour atteindre la destination;
- leur capacité à changer d'itinéraire ou à attendre le rétablissement d'une connexion;
- leur capacité à réduire le volume des données échangées sur le réseau.

Nous comparons, avec des scénarios de simulation identiques, le comportement d'un agent mobile et celui d'une connexion client/serveur pour réaliser des échanges de données. Pour réaliser la simulation d'agents mobiles dans un réseau sans fil ad hoc, nous utilisons l'extension du simulateur NS2 proposée par Kunal Shah dans [75]. Nous commençons par une analyse de ce modèle de simulation, puis nous réalisons les modifications nécessaires pour l'adapter aux scénarios de simulation dans les réseaux ad hoc. Nous terminons cette section par la présentation des résultats des simulations qui nous permettent de vérifier l'intérêt des agents mobiles pour améliorer la fiabilité des transferts de données entre les nœuds d'un MANET.

3.5.1 Analyse du modèle de simulation des agents mobiles

Le modèle de simulation des agents mobiles proposé par Kunal Shah est basé sur la plate forme Aglet d'IBM. Dans l'annexe D, nous présentons les modes de fonctionnement des agents mobiles de cette plate-forme, ainsi que leurs modes de communication et de déplacement, puis nous analysons les caractéristiques suivantes :

- les états d'un *aglet* et les méthodes associées,
- le modèle événementiel de l'*aglet*,
- le modèle de communication des *aglets*.

L'objectif des expérimentations réalisées par Kunal Shah est de montrer que pour certaines applications et dans certains contextes, les performances des agents mobiles sont supérieures à celles d'une application client/serveur. Il considère le cas d'une application d'extraction de données situées sur différents serveurs. L'accès aux réseaux des serveurs se fait à travers une seule

connexion sans fil dont le taux de perte des paquets est paramétrable. La présence des données sur l'un des serveurs est fixée de façon aléatoire. La méthode *run* de l'agent mobile implémente le déplacement de l'agent de serveur en serveur jusqu'à l'obtention des données. La méthode client/serveur effectue, elle, des requêtes successives vers chacun des serveurs.

Le modèle de simulation proposé fait les hypothèses suivantes :
- un agent utilise le protocole de transport TCP,
- la taille du code de l'agent et de ses données sont des paramètres de la simulation,
- les temps de traitement et de reconstruction de l'agent sont proportionnels à la taille de son code,
- le facteur de sélectivité permet de définir la taille de la réponse de l'agent,
- l'implémentation du modèle est réalisée pour effectuer des mesures de performances sans prendre en compte les aspects de sécurité,
- le modèle implémente une mobilité faible qui entraîne l'exécution d'une méthode après le déplacement de l'agent (voir l'annexe D).

L'extension[4] du simulateur NS-2 pour la simulation des agents mobiles est construite à partir des deux nouvelles classes : *contexte* et *agent mobile*.

La classe *contexte* est définie comme une extension de la classe TcpApp du simulateur NS-2. Elle dispose de méthodes nécessaires pour contrôler l'exécution et le déplacement des agents.
La classe *agent mobile*, appelée *Magent*, implémente les fonctionnalités de base de l'agent comme la création, l'expédition ou le retrait que nous expliquons dans l'annexe D. Chaque agent créé possède un identifiant unique utilisé par les objets *contextes* pour accéder à ses données propres.
L'implémentation du modèle proposé fait aussi l'objet d'une validation mathématique basée sur l'évaluation de la charge réseau et du temps d'exécution pour différentes valeurs du paramètre de sélectivité.

Les résultats des simulations réalisées par Kunal Shah montrent que pour un taux de perte faible du lien sans fil, les performances des agents et celles des requêtes client/serveur sont comparables. Par contre, quand ce taux progresse, la performance des agents n'est que peu affectée alors que la performance des requêtes client/serveur décroît de façon exponentielle.

4. http://vishnu.cs.lamar.edu/~kunals/mobile-agents.tar

Nous souhaitons utiliser ce modèle de simulation d'agents mobiles pour comparer la fiabilité d'un transfert de données effectué par des agents mobiles et par des connexions client/serveur dans des réseaux ad hoc dont les connexions physiques entre les nœuds sont instables.

3.5.2 Adaptation et validation du modèle de simulation des agents mobiles pour les réseaux ad hoc

Pour réaliser des échanges bidirectionnels, l'implémentation de la plateforme proposée par Kunal Shah est basée sur l'agent *FullTcp*. La classe *contexte* dérivée de la classe application, *TcpApp*, utilise cette fonctionnalité pour pouvoir émettre et recevoir simultanément. Les simulations effectuées avec l'agent *FullTcp* montrent un dysfonctionnement de son comportement. Pour corriger ce problème lié à l'implémentation de *FullTcp*, la classe *Application*, qui possède un pointeur vers un agent TCP, a été modifiée. Celle-ci utilise désormais deux pointeurs pour associer un agent de transport TCP à chaque sens de transmission.

Dans notre implémentation, nous utilisons l'agent *BayFullTcp* de la version 2.27 du simulateur NS2. Nous y apportons les modifications nécessaires pour obtenir des ouvertures et fermetures de sessions respectant les spécifications du protocole définies dans la RFC 793 [66]. De même, nous devons corriger un problème d'implémentation lié à la gestion des numéros d'acquittement. Les résultats des simulations effectuées pour valider le fonctionnement de la nouvelle implémentation de l'agent *BayFullTcp* sont représentés sur les figures 3.9 page 71 et 3.10 page 72. Nous constatons que le contrôle de la connexion est conforme aux spécifications du protocole [66].
La figure 3.9 représente les paquets TCP émis par les nœuds client et serveur lors du transfert d'un fichier de 1024 octets. Le transfert FTP est initialisé au temps t=30 secondes. Nous constatons que le protocole TCP tente d'établir une connexion (lignes 1 à 5), la valeur de la temporisation est doublée entre chaque réémission. Au temps t=100 secondes, le nœud destination s'est rapproché du nœud client et la connexion TCP commence alors sa phase d'établissement (lignes 5 à 6) pour transmettre la requête de 80 octets (ligne 7). Deux segments de données sont nécessaires pour échanger les 1024 octets (lignes 8 à 10). Lors de la transmission du dernier segment, le serveur positionne le bit F (Fin) pour initier la fermeture de la connexion (lignes 10 à 12).

La figure 3.10 de la page 72 représente les paquets TCP reçus par les différents nœuds quand l'agent mobile se déplace de nœud en nœud et retourne sur son nœud de départ. Nous constatons que l'agent utilise un nœud inter-

```
 1 <--[_o59] 30.000000 0:0>1:1 tcp flags S... seq:0 ack:0 len:0 [SYN_SENT]
 2 <--[_o59] 36.000000 0:0>1:1 tcp flags S... seq:0 ack:0 len:0 [SYN_SENT]
 3 <--[_o59] 48.000000 0:0>1:1 tcp flags S... seq:0 ack:0 len:0 [SYN_SENT]
 4 <--[_o59] 72.000000 0:0>1:1 tcp flags S... seq:0 ack:0 len:0 [SYN_SENT]
 5 <--[_o59] 120.000000 0:0>1:1 tcp flags S... seq:0 ack:0 len:0 [SYN_SENT]
 6 <--[_o68] 120.004512 1:1>0:0 tcp flags SA.. seq:0 ack:1 len:0 [SYN_RCVD]
 7 <--[_o59] 120.006570 0:0>1:1 tcp flags .AP. seq:1 ack:1 len:80 [ESTABLISHED]
 8 <--[_o68] 120.009149 1:1>0:0 tcp flags .A.. seq:1 ack:81 len:536 [ESTABLISHED]
 9 <--[_o59] 120.015435 0:0>1:1 tcp flags .A.. seq:81 ack:537 len:0 [ESTABLISHED]
10 <--[_o68] 120.017833 1:1>0:0 tcp flags .APF seq:537 ack:81 len:488 [FIN_WAIT_1]
11 <--[_o59] 120.023815 0:0>1:1 tcp flags .A.F seq:81 ack:1026 len:0 [LAST_ACK]
12 <--[_o68] 120.026033 1:1>0:0 tcp flags .A.. seq:1026 ack:82 len:0 [CLOSED]
```

FIG. 3.9 – *Ouverture de la connexion TCP et transfert FTP*

médiaire (_o84) pour atteindre sa destination, le nœud (_o88). Au début du déplacement, les connexions physiques entre les nœuds sont stables. L'agent, d'une taille de 100 octets, commence par se déplacer sur le nœud _o84. Les lignes 1 à 5 montrent les phases d'ouverture de la connexion, de transfert de l'agent et de fermeture de la connexion. Ensuite, l'agent se déplace de la même façon du nœud _o84 au nœud _o88, sa destination (lignes 6 à 10). Puis l'agent rejoint son nœud initial (_o80) en passant par le nœud intermédiaire _o84 (lignes 11 à 28) . Pour le retour, la taille de l'agent avec son code et ses données est fixée à 1300 octets.

Le mode de déplacement de l'agent de nœud en nœud est conforme à son modèle de programmation défini dans sa méthode *run* que nous donnons dans l'annexe D. Nous constatons à nouveau la capacité du protocole *BayFullTcp*, modifié, à gérer des connexions bidirectionnelles.

3.5.3 Analyse comparative de la fiabilité des échanges

Nous considérons un scénario d'interrogation d'une base de données et nous réalisons une première simulation basée sur l'échange d'une requête et de sa réponse entre un client et un serveur. La requête et sa réponse sont transmises aux destinataires suivant deux modes de transferts :

- une connexion client/serveur établie de bout en bout,
- un agent mobile qui se déplace de nœud en nœud.

Les simulations sont réalisées avec un modèle de propagation de type *shadowing*. Ce modèle présenté dans [34] définit la probabilité de stabilité d'une connexion physique en fonction de la distance séparant deux nœuds. La distribution pseudo-aléatoire de ce modèle de propagation permet de simuler les ruptures des connexions physiques générées par le mouvement des

```
 {1 -->[_o84] 14.020218 0:0>1:0 tcp flags S... seq:0 ack:0 len:0 [LISTEN]
  2 -->[_o80] 14.047387 1:0>0:0 tcp flags SA.. seq:0 ack:1 len:0 [SYN_SENT]
  3 -->[_o84] 14.071929 0:0>1:0 tcp flags .APF seq:1 ack:1 len:100 [SYN_RCVD]
  4 -->[_o80] 14.094811 1:0>0:0 tcp flags .A.F seq:1 ack:102 len:0 [FIN_WAIT_1]
  5 -->[_o84] 14.122555 0:0>1:0 tcp flags .A.. seq:102 ack:2 len:0 [LAST_ACK]

  6 -->[_o88] 15.093036 1:0>2:0 tcp flags S... seq:0 ack:0 len:0 [LISTEN]
  7 -->[_o84] 15.116118 2:0>1:0 tcp flags SA.. seq:0 ack:1 len:0 [SYN_SENT]
  8 -->[_o88] 15.140440 1:0>2:0 tcp flags .APF seq:1 ack:1 len:100 [SYN_RCVD]
  9 -->[_o84] 15.163082 2:0>1:0 tcp flags .A.F seq:1 ack:102 len:0 [LISTEN]
 10 -->[_o88] 15.185804 1:0>2:0 tcp flags .A.. seq:102 ack:2 len:0 [LAST_ACK]

 11 -->[_o84] 16.173125 2:0>1:0 tcp flags S... seq:0 ack:0 len:0 [LISTEN]
 12 -->[_o88] 16.195787 1:0>2:0 tcp flags SA.. seq:101 ack:1 len:0 [SYN_SENT]
 13 -->[_o84] 16.227565 2:0>1:0 tcp flags .A.. seq:1 ack:102 len:536 [SYN_RCVD]
 14 -->[_o88] 16.254861 1:0>2:0 tcp flags .A.. seq:102 ack:537 len:0 [ESTABLISHED]
 15 -->[_o84] 16.286339 2:0>1:0 tcp flags .A.. seq:537 ack:102 len:536 [ESTABLISHED]
 16 -->[_o88] 16.309241 1:0>2:0 tcp flags .A.. seq:102 ack:1073 len:0 [ESTABLISHED]
 17 -->[_o84] 16.351761 2:0>1:0 tcp flags .APF seq:1073 ack:102 len:228 [ESTABLISHED]
 18 -->[_o88] 16.374763 1:0>2:0 tcp flags .A.F seq:102 ack:1302 len:0 [FIN_WAIT_1]
 19 -->[_o84] 16.397525 2:0>1:0 tcp flags .A.. seq:1302 ack:103 len:0 [LAST_ACK]

 20 -->[_o80] 17.378503 1:0>0:0 tcp flags S... seq:101 ack:0 len:0 [LISTEN]
 21 -->[_o84] 17.403620 0:0>1:0 tcp flags SA.. seq:101 ack:102 len:0 [SYN_SENT]
 22 -->[_o80] 17.435118 1:0>0:0 tcp flags .A.. seq:102 ack:102 len:536 [SYN_RCVD]
 23 -->[_o84] 17.460055 0:0>1:0 tcp flags .A.. seq:102 ack:638 len:0 [ESTABLISHED]
 24 -->[_o80] 17.491233 1:0>0:0 tcp flags .A.. seq:638 ack:102 len:536 [ESTABLISHED]
 25 -->[_o84] 17.514355 0:0>1:0 tcp flags .A.. seq:102 ack:1174 len:0 [ESTABLISHED]
 26 -->[_o80] 17.541025 1:0>0:0 tcp flags .APF seq:1174 ack:102 len:228 [ESTABLISHED]
 27 -->[_o84] 17.563647 0:0>1:0 tcp flags .A.F seq:102 ack:1403 len:0 [FIN_WAIT_1]
 28 -->[_o80] 17.586509 1:0>0:0 tcp flags .A.. seq:1403 ack:103 len:0 [LAST_ACK]
```

FIG. 3.10 – *Trames TCP reçues lors du déplacement d'un agent mobile de nœud en nœud*

nœuds. La figure 3.11 de la page 73 représente la probabilité de stabilité de la connexion physique en fonction de la distance.
La figure 3.12 de la page 74 représente les variations dans le temps du trafic des données échangées sur un lien pour une stabilité de la connexion physique et une distance entre les nœuds, données. Nous constatons qu'avec les mêmes paramètres de simulation le trafic est distribué de façon pseudo-aléatoire dans le temps pour chaque simulation.

Les simulations sont effectuées avec les paramètres suivants :

- type d'application : transfert FTP,
- volume des données à transférer : variable,
- nombre de nœuds entre le client et le serveur : 18,
- protocole de routage : DSDV,
- type de propagation : shadowing (distribution pseudo-aléatoire),
- distance entre les nœuds : 250 mètres,
- durée de la simulation : 120 secondes.

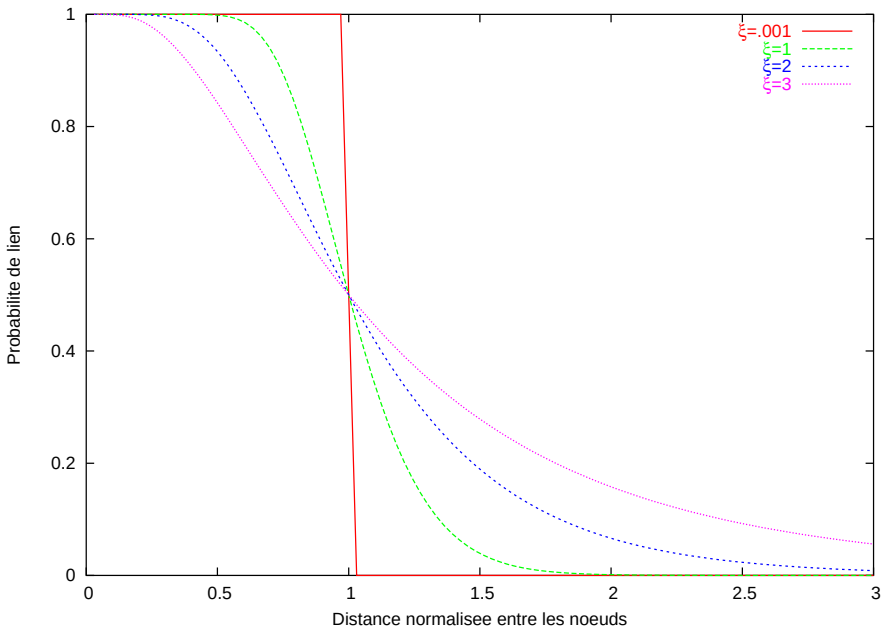


FIG. 3.11 – *Modèle de propagation de type shadowing*

Nous faisons varier de 1 à 48 le nombre de nœuds situés entre les nœuds source et destination. Puis, pour un nombre de nœuds donné, nous réalisons 20 essais avec une distribution pseudo-aléatoire de la stabilité des liens et nous calculons le pourcentage de transferts réussis. Nous considérons qu'un transfert est réussi quand la réponse est reçue par le client et que la connexion TCP est correctement fermée entre les nœuds client et serveur ou entre le client et le nœud précédent.
Nous réalisons plusieurs séries de simulations en faisant varier le volume des réponses :

- des requêtes et réponses de 100 octets, transportées dans un seul paquet TCP sont comparables par leurs tailles à des interrogations de bases de données,
- des réponses avec un volume de 64 Koctets sont comparables au transfert d'un fichier de log.

La figure 3.13 de la page 75 représente les pourcentages de transferts réussis par les agents mobiles et les connexions client/serveur établies de bout en bout. Pour une durée de simulation donnée, nous constatons que pour les mêmes conditions de stabilité des liens, un même nombre de nœuds intermédiaires, et une taille des requêtes et réponses fixée à 100 octets, les agents

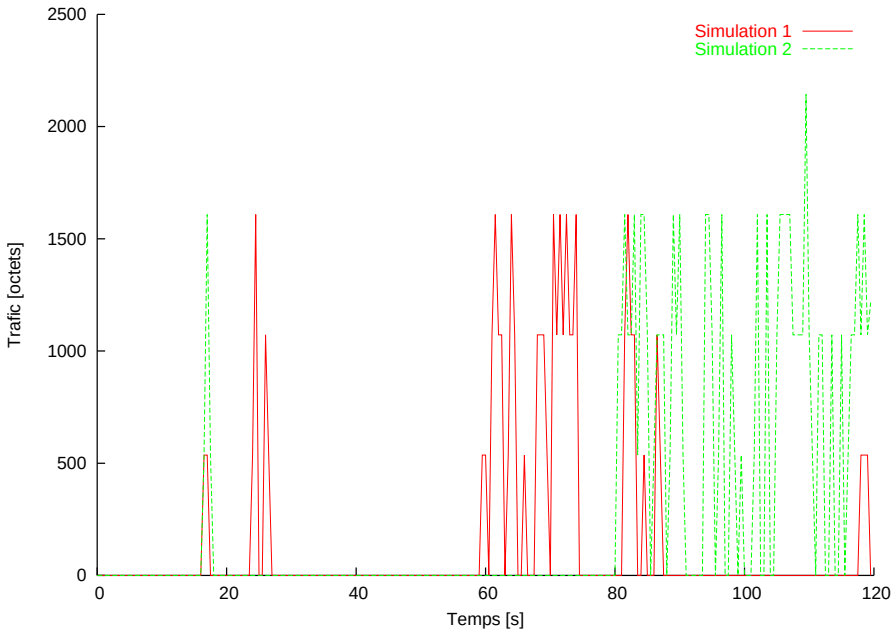


FIG. 3.12 – *Variation du trafic pour une connexion de longueur donnée*

mobiles ont un taux de transferts réussis supérieur à celui des connexions client/serveur :

- Les agents mobiles réussissent 100% des transferts jusqu'à 21 sauts, alors que la connexion client/serveur réussit 100% des transferts en traversant au maximum 4 nœuds.
- Les agents mobiles réussissent environ 50% des transferts avec 30 nœuds intermédiaires et au delà de 40 nœuds à traverser, plus aucun transfert n'est possible.
- La connexion client/serveur réussit, elle, à réaliser 50% des échanges dès que les paquets traversent 20 nœuds.

Sur la figure 3.15 de la page 76 nous représentons les résultats des simulations d'un transfert FTP réalisées avec un protocole de routage proactif ($DSDV$) et un protocole de routage réactif ($AODV$), les autres paramètres de simulations restant inchangés. Nous constatons que les courbes ont une allure similaire. Pour ce scénario particulier, le protocole de routage réactif est plus performant que le protocole de routage proactif. Dans nos simulations, nous ne nous intéressons pas aux performances des protocoles de routage ad hoc. Un certain nombre de travaux ont déjà été réalisés sur ce sujet, dont [18].

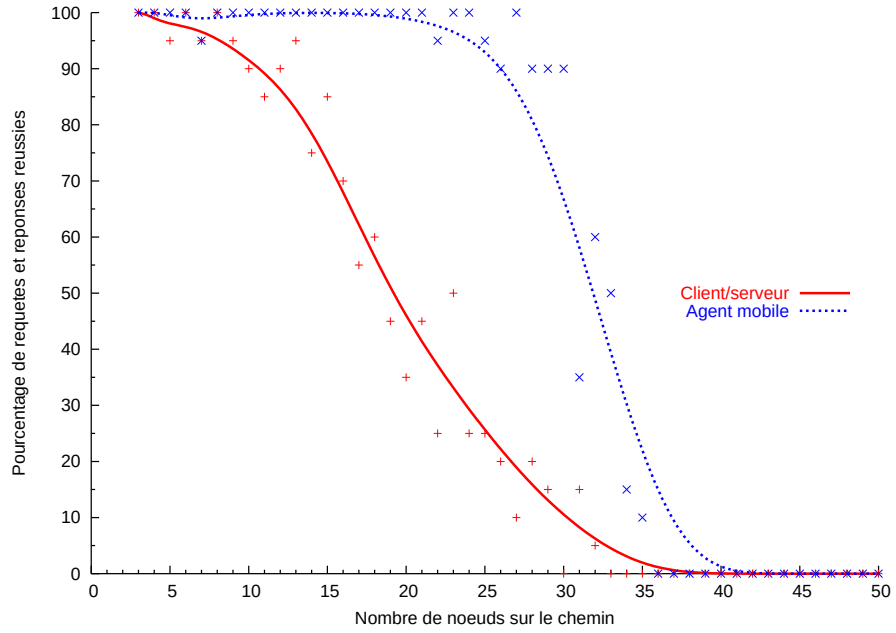


FIG. 3.13 – *Pourcentages de requêtes/réponses réussies*

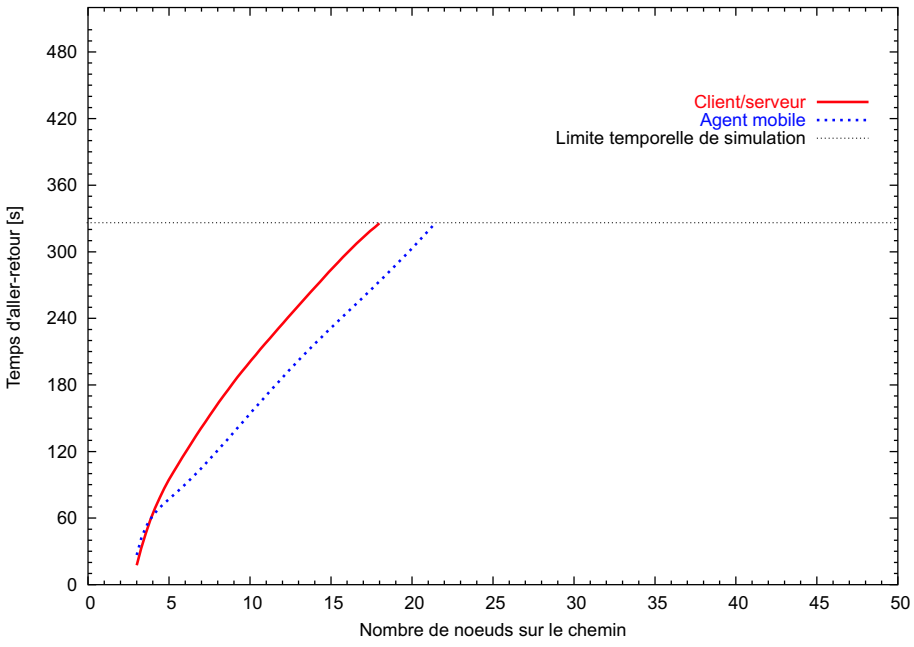


FIG. 3.14 – *Évolution du temps nécessaire pour effectuer une requête et sa réponse*

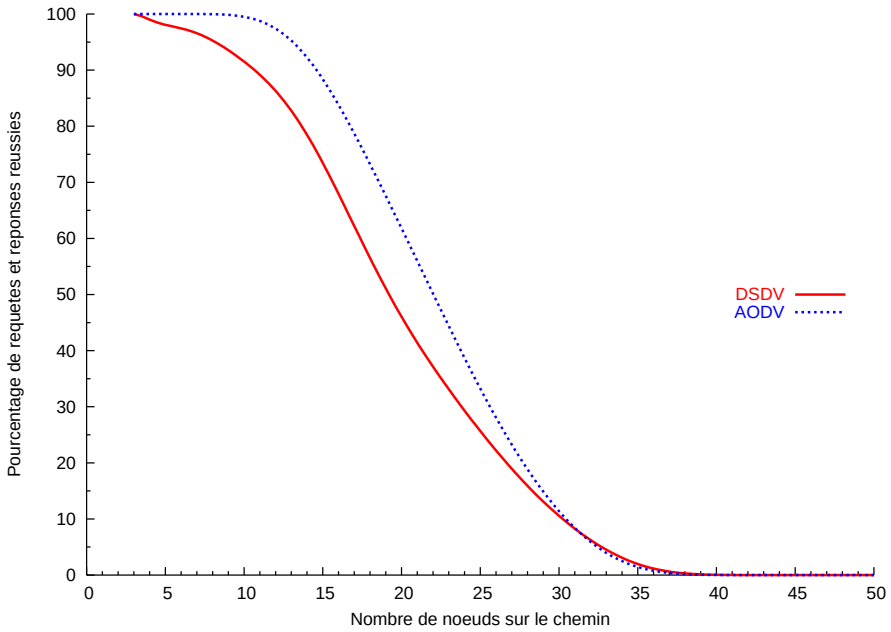


FIG. 3.15 – *Influence du protocole de routage sur le transfert des données*

Nous souhaitons comparer, dans un même environnement, le comportement d'un agent mobile et d'une connexion client/serveur pour réaliser des échanges de données.

Sur la courbe 3.14, de la page 75 nous relevons le temps moyen des transferts (question/réponse) réussis avec les deux modes d'échange des données. La durée de la simulation est fixée à 500 secondes. Pour les agents mobiles et les échanges client/serveur, nous constatons que le temps moyen d'exécution des requêtes réussies est proportionnel au nombre de sauts. Ce temps est limité par la durée de la simulation, fixée ici à 500 secondes.
La durée de la simulation ne permet pas d'obtenir des résultats significatifs au delà de 21 nœuds avec des agents mobiles et 18 nœuds avec le client/serveur.

Les courbes de la figure 3.16-a page 78 montrent que pour une connexion client/serveur le volume des données transférées a peu d'influence sur le temps de transfert. En effet, celui-ci dépend de l'état des liaisons physiques quand le protocole TCP tente d'ouvrir une connexion ou d'acquitter un paquet.

Sur les courbes de la figure 3.16-b page 78, obtenues avec un agent mobile, nous constatons que pour un même nombre de sauts, le temps de transfert augmente avec le volume des données transférées. La figure 3.18 de la page 80 montre que jusqu'à 32 Koctets de données transférées, les agents mobiles effectuent le transfert plus rapidement que la connexion client/serveur établie de bout en bout.

Nous analysons l'influence de la taille des réponses sur le pourcentage de transferts réussis. Sur la figure 3.17-a de la page 79, nous constatons qu'un volume de données compris entre 100 octets et 32 Koctets n'a pas d'influence sur le pourcentage de transferts réussis. Dans le cas d'un transfert réalisé par un agent mobile (figure 3.17-b de la page 79) nous constatons que le taux de transferts réussis baisse quand la taille des données augmente. Pour la même taille des requêtes/réponses et avec les mêmes conditions de transfert, les courbes de la figure 3.19 page 80 montrent que l'agent mobile réussit davantage de transferts qu'une connexion client/serveur.

Ces simulations nous permettent de conclure que dans un réseau sans fil ad hoc, avec des conditions de transferts de données identiques, un agent mobile qui se déplace de nœud en nœud réussit plus de transferts de données qu'une connexion client/serveur établie de bout en bout.

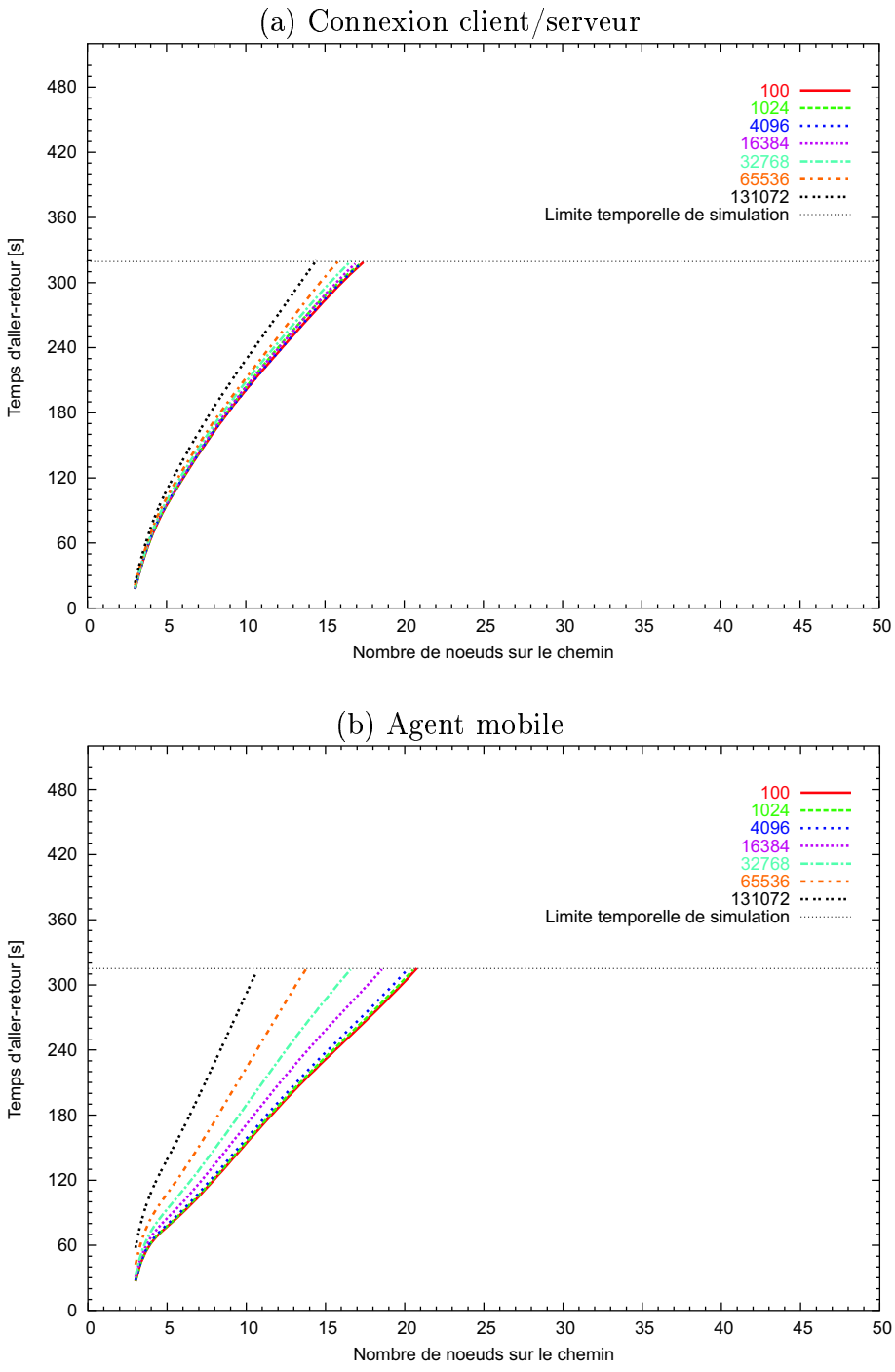
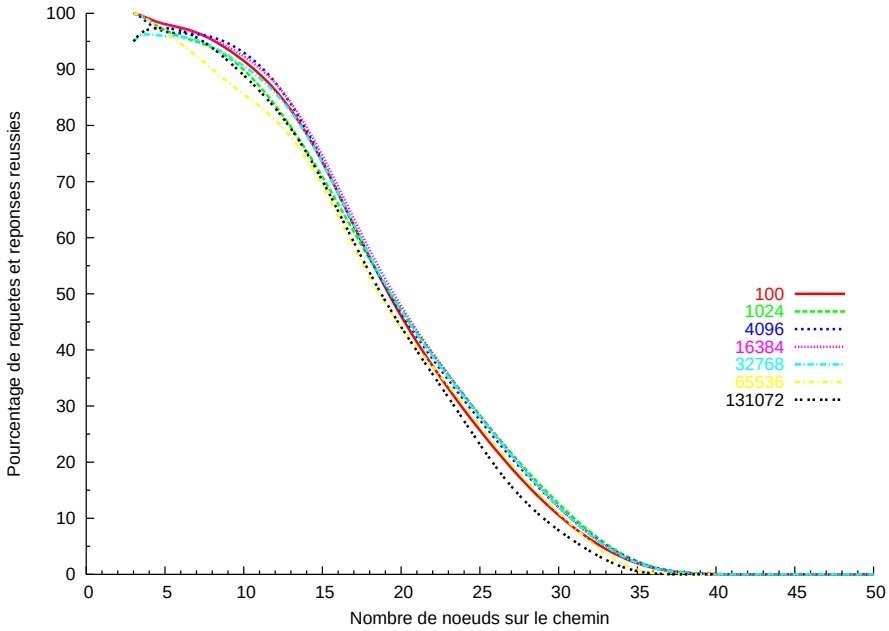


FIG. 3.16 – *Évolution du temps nécessaire pour effectuer une requête et sa réponse*

(a) Connexion client/serveur



(b) Agent mobile

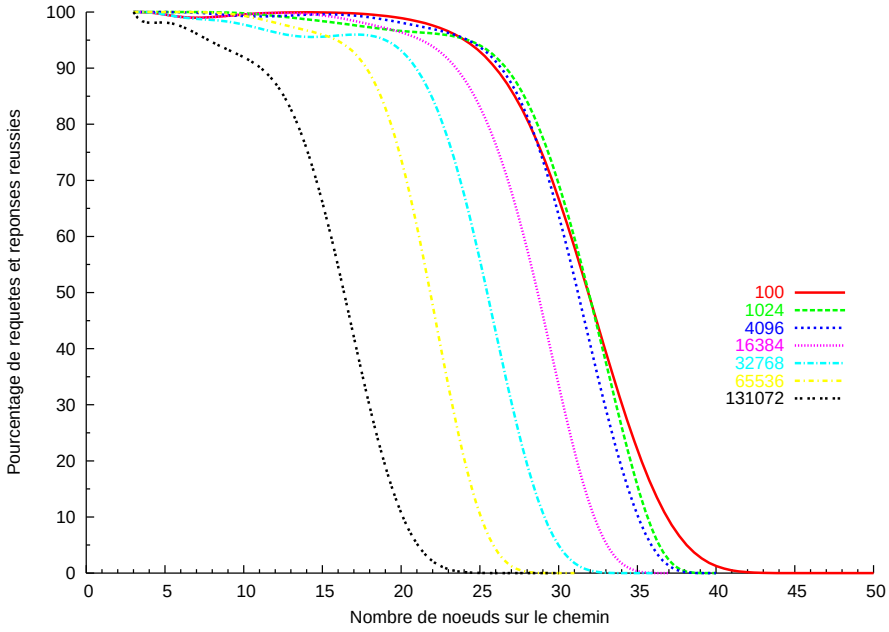


FIG. 3.17 – *Pourcentages de requêtes/réponses réussies en fonction de la taille de la réponse*

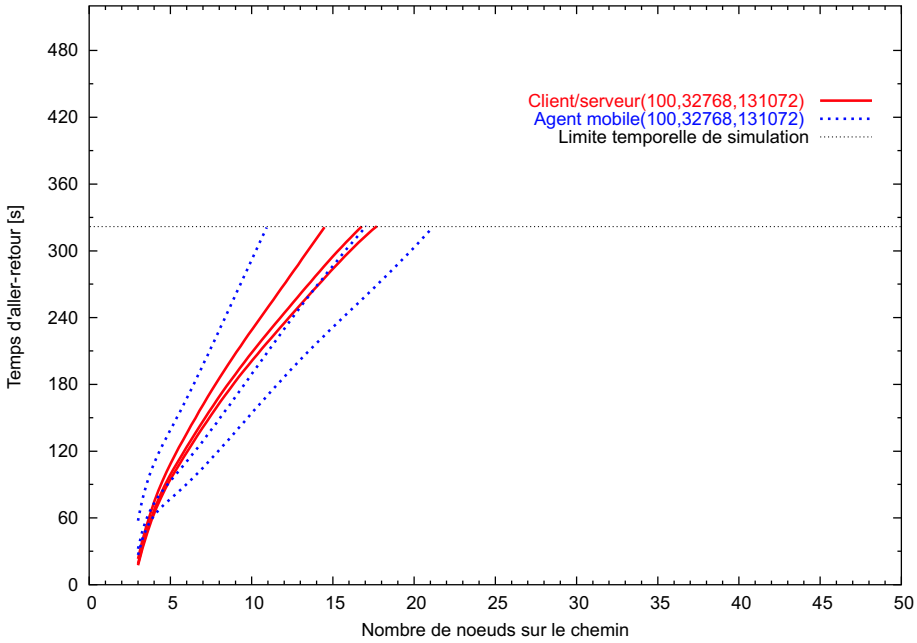


FIG. 3.18 – *Comparaison des temps moyens de requêtes/réponses selon la taille des données et le mode de transfert*

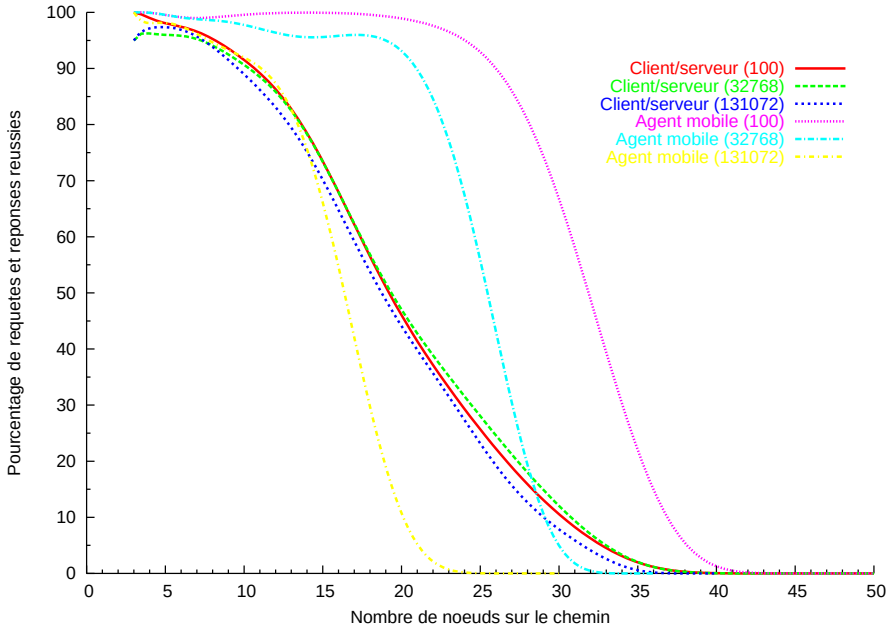


FIG. 3.19 – *Comportement d'un agent mobile et d'une connexion client/serveur en fonction de la taille de la réponse*

3.5.4 Conclusions sur le résultat des simulations effectuées pour vérifier la fiabilité des échanges par agents mobiles

Sur la base des simulations réalisées, nous montrons que dans un réseau ad hoc, avec les mêmes paramètres de simulations, un transfert de données est plus fiable quand il est réalisé par un agent mobile qui se déplace de nœud en nœud que quand il est réalisé au moyen de connexions client/serveur établies de bout en bout quel que soit le nombre de nœuds traversés. Nous montrons aussi que pour des volumes de données transférés importants, par exemple 32 Koctets dans nos conditions de simulations, un agent mobile reste plus performant.

3.6 L'architecture interne des LIDS

Chaque LIDS est conçu pour être autonome et exploiter les informations collectées localement ou sur un nœud distant. L'architecture retenue pour les LIDS, représentée figure 3.20 de la page 83, est une évolution de l'architecture que nous avons présentée dans [72] et [59].

Elle est construite à partir des trois composants de base du modèle fonctionnel proposé par l'IDWG : l'Analyseur, le Senseur et le Manager.
Nous utilisons, dans cette version, une approche par scénarios, que nous jugeons plus simple à implémenter et à tester qu'une approche comportementale. Notre objectif est d'évaluer l'intérêt des agents mobiles pour la coopération entre les LIDS.

Des tests comparatifs ultérieurs nous permettront de retenir l'approche la plus performante pour la détection, voire, à défaut, l'intégration des deux approches. Y. Zhang et al. proposent dans [82] un modèle d'IDS pour MANET, distribué et coopératif, basé sur une détection comportementale d'attaques contre plusieurs protocoles de routage. Les auteurs présentent les résultats des simulations réalisées sans toutefois chercher à comparer les performances des systèmes selon l'approche utilisée.

L'analyseur par *scénarios* utilise une base de signatures. Chaque signature d'attaque est décrite par un automate à états finis, ou FSD pour *Finite State Diagram*, codée avec une syntaxe XML[5] dans le fichier de signatures. La transition d'un état à un autre est la conséquence d'un événement. Cet événement est transmis à l'Analyseur par le module

5. eXtensible Markup Langage - http://www.w3.org/XML/

Gestion de la distribution, dont la fonction est aussi d'associer le ou les événements de bas niveau, générés par un ou plusieurs senseurs, à un événement abstrait traité par un automate à états finis. L'analyseur effectue des requêtes périodiques pour détecter les changements d'états.

Les senseurs ont pour fonction de traiter les requêtes transmises par le module *Gestion de la distribution*. Selon le type d'information demandée, les senseurs effectuent les requêtes sur le nœud local ou sur un nœud distant via la plate-forme à Agents Mobiles.

Le Manager d'alertes transmet aux autres IDS les alertes détectées par le LIDS. Une alerte générée par un autre IDS peut aussi être prise en compte localement par le Manager.

Pour permettre au LIDS d'opérer, des modules externes sont nécessaires :

- Un fichier de signatures d'attaques, *XML Signatures*,
- Un agent SNMP, chargé d'effectuer les requêtes locales sur la MIB II,
- Une IHM qui assure l'interface de commandes et de visualisation avec l'opérateur,
- Un module de communication, formé des couches de protocoles des différents niveaux et de l'interface physique.

3.7 Conclusions sur le modèle d'IDS distribué pour les réseaux ad hoc

Dans ce chapitre, nous proposons un modèle d'IDS distribué pour les MANET. Celui-ci est formé d'IDS autonomes, que nous appelons LIDS, installés sur chaque nœud et d'un système de coopération fiable et autonome pour gérer les échanges entre les LIDS.

Nous montrons par des simulations que dans un réseau ad hoc, les transferts de données sont plus fiables lorsqu'ils sont contrôlés par des agents mobiles que par des connexions client/serveur. Leur utilisation pour contrôler les échanges entre les LIDS permettra de doter l'architecture de l'IDS d'un système de coopération fiable et autonome.

Nous proposons et nous validons des adaptations au modèle de simulation du protocole *BayFullTcp* de la version 2.9 du simulateur NS-2. Cette nouvelle implémentation sera mise à la disposition de la communauté des utilisateurs du simulateur NS-2.

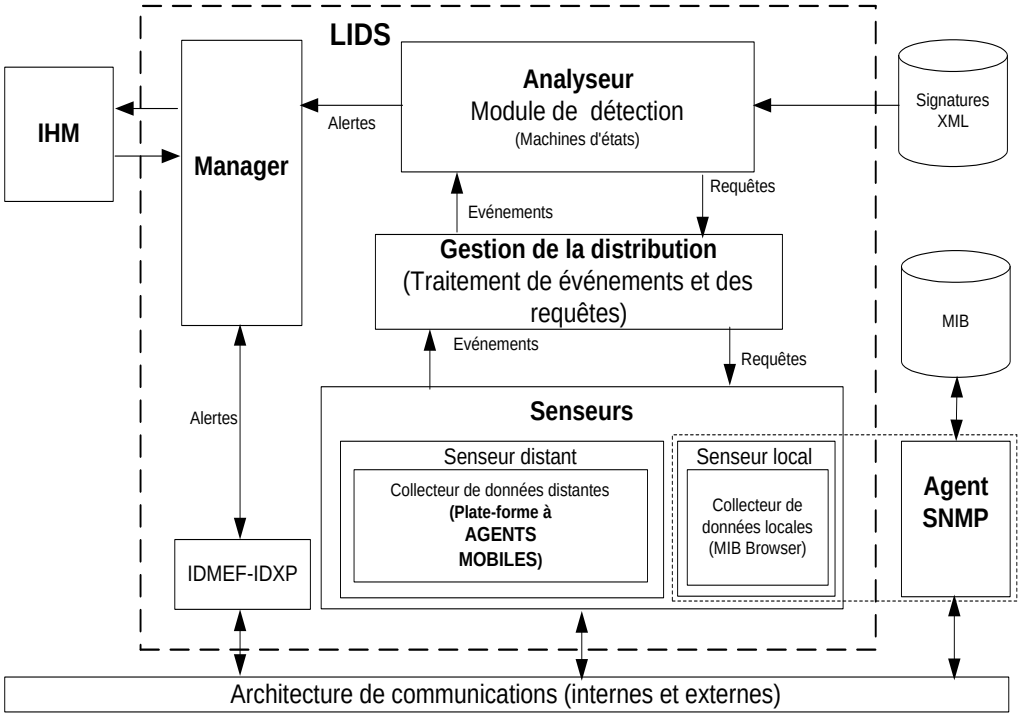


FIG. 3.20 – *Architecture interne d'un LIDS*

Nous présentons aussi l'architecture interne des LIDS. Celle-ci met en évidence le choix des MIB comme source de données.

L'architecture d'IDS distribuée, le modèle de coopération par agents mobiles et l'architecture interne des LIDS que nous proposons ont été présentés dans cinq communications [2],[71],[72],[59],[13].

Chapitre 4

Implémentation et validation expérimentale de l'architecture

4.1 Objectifs des expérimentations et méthodologie de validation

Dans le chapitre précédent, nous proposons, sur la base des résultats obtenus par simulation, d'utiliser des agents mobiles pour réaliser la coopération entre des LIDS installés sur les nœuds mobiles d'un MANET. Nous poursuivons nos expérimentations par une validation fonctionnelle et une évaluation des performances de l'architecture proposée.

Dans ce chapitre, nous présentons d'abord le prototype, la plate-forme de tests et les expérimentations.
Nous évaluons l'aspect fonctionnel de la détection d'intrusions et les performances du prototype avec différents scénarios d'attaques.
Dans un environnement ad hoc expérimental, nous analysons alors les aspects qualitatifs et quantitatifs des échanges entre LIDS, basés sur le mode client/serveur et sur des agents mobiles.
Nous terminons cette section en évaluant les actions complémentaires d'un LIDS et d'un autre IDS installés sur le même nœud.

4.2 Description du prototype

La finalité du prototype est de valider et d'évaluer les performances du système de détection d'intrusions distribuée que nous proposons.

Pour tester et optimiser les fonctions critiques des LIDS, comme la détection et la coopération, nous réalisons une conception modulaire en suivant le

schéma de l'architecture présentée sur la figure 3.20 page 83.

L'architecture interne des LIDS, organisée en modules, permet de valider et d'optimiser chaque module sans remettre en cause l'ensemble de l'architecture. Cette approche est importante pour permettre d'évaluer les performances des différentes méthodes de détection dans un réseau sans fil ad hoc. Les caractéristiques de l'architecture suivent les spécifications des LIDS établies dans la section 3.3.2. L'implémentation réalisée en java et l'utilisation des variables des MIB permettent aussi au prototype d'être indépendant de la plate-forme matérielle utilisée.

Dans cette section nous présentons les principaux diagrammes UML [26] utilisés pour la conception des LIDS.

4.2.1 Les diagrammes de classes

Les trois classes *Manager*, *Analyser* et *EventAbstractor* sont respectivement associées aux modules Manager, Analyseur et Gestion de la Distribution /Senseurs de l'architecture interne des LIDS.

Une classe principale, appelée *IDSKernel*, a pour fonction de créer l'ensemble des composants (*objets*) de l'IDS.

L'interface de visualisation est représentée par le composant IHM.

La classe *Analyser* réalise la détection des intrusions. Chaque attaque, décrite par une signature dans un fichier XML, est représentée par un automate à états finis ou FSD (*Finite State Diagram*) géré par l'analyseur.

- A l'initialisation, l'*Analyser* crée un *FSDIterator* pour chaque FSD résultant de l'extraction d'une signature par l'*XMLExtractor*.
  Un objet de la classe *FSDIterator* est un FSD dont les attributs, de type vecteur, mémorisent les états et les transitions. Ses méthodes traitent les événements et les changements d'états afin de détecter les attaques. L'ensemble des *FSDIterator*, résultant de l'extraction des signatures, est regroupé dans un vecteur, appelé *IteratorList*.
- Pendant la détection, l'*Analyser* effectue des requêtes périodiques, appelées *PeriodicalQueries*, ou à la demande, les *Queries*, pour détecter les événements responsables des changements d'états. Un nouveau FSD est créé lors du premier changement d'état pour permettre la détection de nouvelles attaques du même type.

Les objets *Event*, *Query*, *PeriodicalQuery* et *Alarm* représentent les messages échangés entre les différents modules. Ils sont regroupés dans une liste appelée *IDSObjectList*, et distribués périodiquement à chacun

des modules destinataires.

La classe *EventAbstractor* implémente les fonctions des modules Gestion de la Distribution/Senseurs. Cette classe réalise l'association entre les événements des FSD et les résultats des requêtes effectuées par les senseurs. Quand une demande de requête est transmise par l'analyseur au module gestion de la distribution, l'*EventAbstractor*, celui-ci reçoit, analyse et transforme la requête en requêtes élémentaires dans une syntaxe adaptée aux caractéristiques des senseurs utilisés.

Sur notre prototype, les requêtes élémentaires sont des requêtes SNMP réalisées par le *MIBBrowser*. Une ou plusieurs variables MIB peuvent être associées à un événement d'une FSD.

L'*EventAbstractor* est aussi chargé de transmettre les demandes de requête, appelées *MIBQuery* directement au *MIBBrowser* local ou à un agent mobile, afin de transporter la requête sur un nœud distant.

Parmi les différentes plates-formes disponibles, présentées dans la section 2.3.2, page 44, nous utilisons la plate-forme Aglet d'IBM [48]. Ses caractéristiques de mobilité et portabilité sont compatibles avec nos besoins. De plus, nous disposons des sources et d'une documentation complète pour réaliser les adaptations nécessaires.

La classe *IHM* met en forme les alarmes générées par le LIDS. Par exemple, dans le cas d'une attaque de type *Stepstone*, après la détection d'une connexion entrante, c'est-à-dire lorsqu'une nouvelle ligne, avec le port destination Telnet, apparaît dans la table *TCPconnEntry*, une nouvelle fenêtre affiche la chaîne des connexions. La table des connexions TCP de la MIB est représentée dans l'Annexe F.

Une fenêtre de compte-rendu est ouverte pour chaque nouvelle attaque détectée. Une fenêtre appelée *Topologie* affiche en permanence l'état des connexions entre tous les nœuds équipés d'un LIDS.

4.2.1.1 Les objets

Le diagramme de la figure 4.1, page 88, représente les classes dont les instances sont les objets des LIDS. Les classes *Analyser, FiniteSateDiagram, FsdIterator* et *State* héritent de la classe *IDSModule*.

Pour chaque FSD, instance de la classe *FiniteSateDiagram*, les attributs *messageList* et *stateList* sont des vecteurs qui représentent respectivement les messages à envoyer et l'ensemble de ses états. Les méthodes de cette classe permettent d'ajouter un message ou état. Les messages échangés entre les

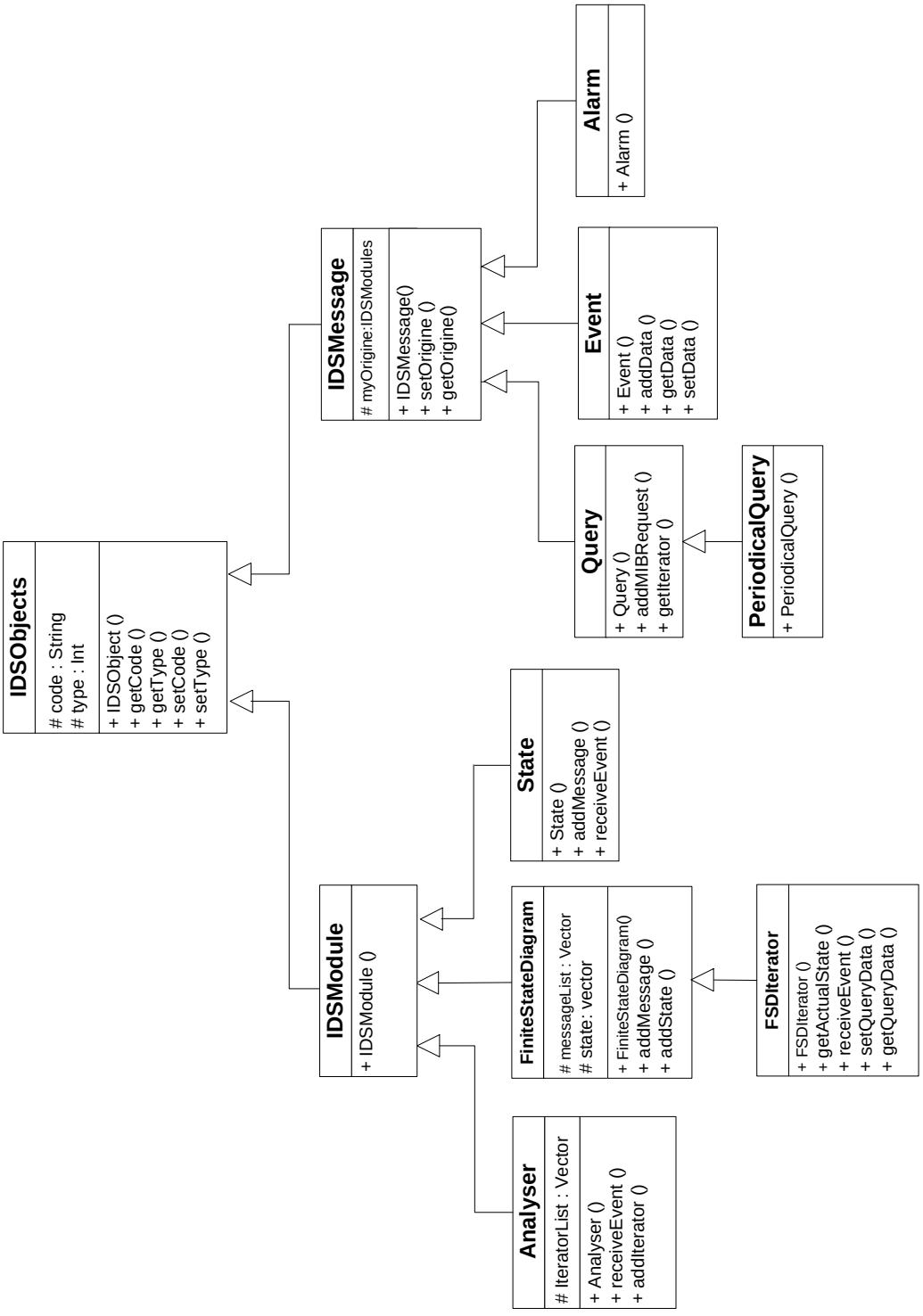


FIG. 4.1 – *Les composants des LIDS*

modules sont des instances des classes *Query, Event, Alarm.*

La classe *Event* est un conteneur d'événements dont les méthodes *setData, getData, AddData* permettent de manipuler les données de l'événement représentées par un vecteur. Les *Queries* sont des messages utilisés pour collecter les données dans les MIB.

Les alarmes représentent les messages générés lorsqu'une attaque est détectée. Dans la version actuelle du prototype, le module *Manager* transmet directement les alarmes à l'IHM.

La classe *FSDIterator* étend la classe *FiniteSateDiagram*, ses méthodes lui permettent de générer des messages de type *Query* pour effectuer des requêtes, ou de traiter les messages de type *Event* reçus pour modifier l'état courant du FSD.
L'état d'un FSD est défini par la classe *State* dont les méthodes permettent de connaître l'état suivant, les types de messages à générer ainsi que les événements qui entraînent un changement d'état.

4.2.1.2 Les FSD - FiniteSateDiagram

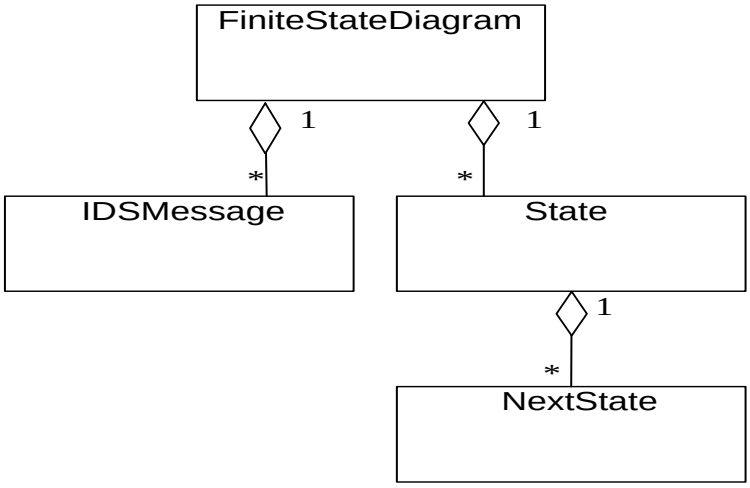


FIG. 4.2 – *Les Finite State Diagram*

Un FSD représente une signature d'attaque. Il est composé d'états et de messages, instances des classes *State*, *NextState* et *IDSMessage* (cf. figure 4.2). Les méthodes de cette classe (*addState* et *addMessage*) permettent, à l'initialisation, de construire les vecteurs *messagesList* et *stateListe*. Pour la détection des attaques l'*Analyser* utilise des *FSDIterator*, construits à partir

de chaque FSD et placés dans l'*IterartorList*. Les méthodes *receiveEvent* et *getQueryData* sont chargées de gérer les changements d'états.

4.2.1.3 L'IDS Kernel

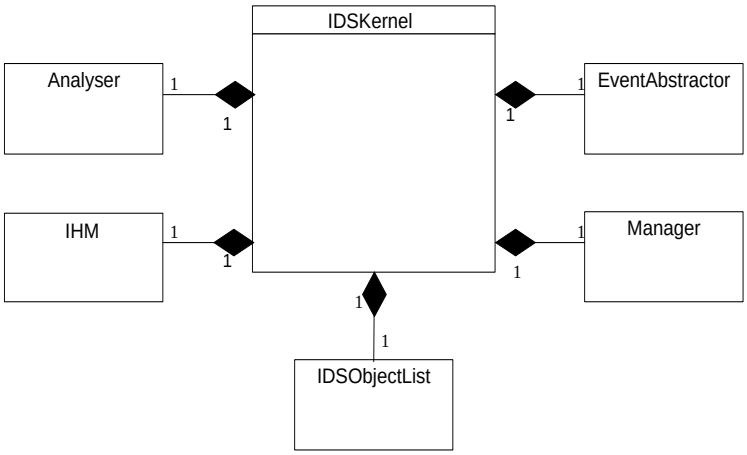


FIG. 4.3 – *L'IDSKernel*

A l'initialisation, l'instance de cette classe construit les composants du LIDS représentés sur la figure 4.3 : *le Manager, l'Analyser, l'IHM, l'EventAbstractor, l'IDSObjectList*. Périodiquement, les méthodes *Distribute* et *send-(IDSobject)* de la classe *IDSKernel* sont invoquées pour distribuer les messages, instances des classes *Event, Query, PeriodicalQuery, Alarm*, de l'*IDSObjectList*, à leur destinataire *l'Analyser, l'EventAbstractor et l'IHM.*

4.2.1.4 L'Analyser

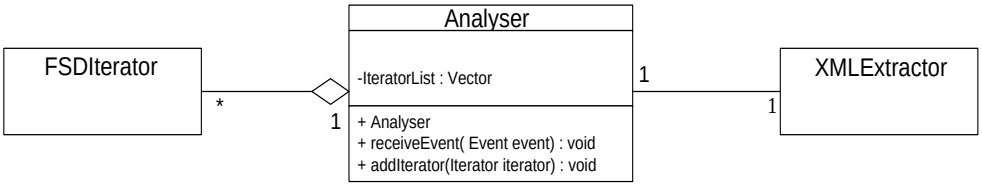


FIG. 4.4 – *L'Analyser*

L'*Analyser*, représenté avec ses principaux composants sur la figure 4.4 de la page 90, est le composant central de l'architecture des LIDS, sa fonction

principale est d'initialiser et de gérer l'ensemble des *FSDItérator* à partir des événements reçus.

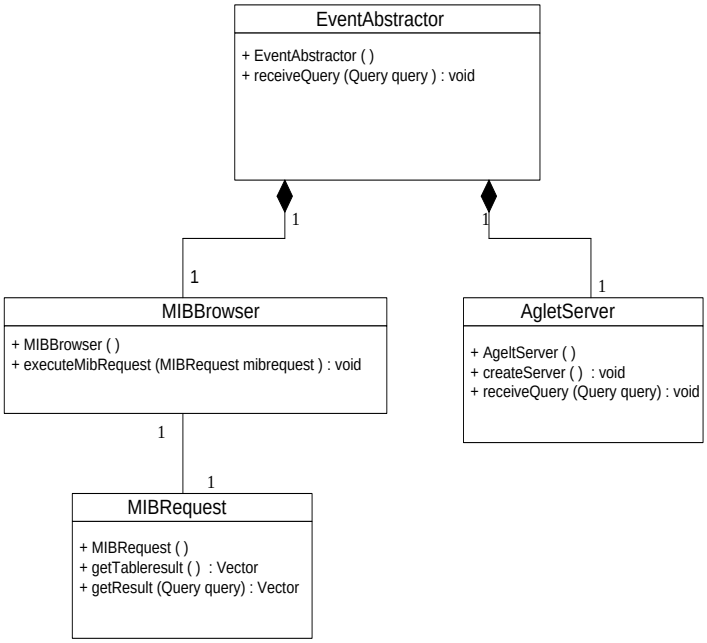


FIG. 4.5 – *L'EventAbstractor*

4.2.1.5 L'EventAbstractor

L'*EventAbstractor* reçoit et traite les messages de type *Query* et *PeriodicalQuery*. Selon les arguments du message, l'*EventAbstractor* transmet la requête au *MIBBrowser* local ou à *AgletServer* si la requête doit être réalisée sur un autre nœud.

Les deux classes *MIBBrowser* et *AgletServer*, sont associées à la classe *EventAbstractor*. La première s'interface avec les agents SNMP locaux alors que la deuxième contrôle les agents mobiles.

4.2.2 Diagramme de séquence

La figure 4.6 de la page 93, représente les messages échangés entre les différents objets. L'*Analyser*, selon les événements associés au changement d'états des FSD, enregistre les *Queries* à réaliser dans la liste *IDSObjectList*. Celles-ci sont périodiquement reçues et traitées par l'*EventAbstractor*. Les

événements générés par le résultat des requêtes effectuées sur la MIB locale sont ajoutés *IDSObjectList* et reçus par l'*Analyser*. Quand un SFD change d'état courant, la liste des *IDSObjectList* est mise à jour afin qu'à la prochaine distribution, le nouveau message (*Event, Query, Alarm*) soit transmis à son destinataire.

Si la requête est destinée à un système distant, alors celle-ci est transmise à *l'AgletServer* local avec ses paramètres. Celui-ci mandatera un agent mobile pour aller effectuer la requête SNMP sur le système distant. L'agent mobile créé pour cette mission se déplace selon l'état du réseau et arrivé à destination, en fonction de l'objectif de sa mission, il peut poursuivre ses déplacements sur d'autres nœuds.

A la fin de sa mission, l'agent mobile revient avec ses données sur son système et se détruit après avoir transmis ses données à l'IHM.

4.2.3 Caractéristiques de l'implémentation du prototype

Nous avons retenu la plate-forme à agents Aglet[1] car ses fonctionnalités sont compatibles avec les tests à réaliser. De plus, elle est écrite en java, ce qui la rend portable. Son code source est stable, la documentation complète et disponible.

Le code du prototype, réalisé en java 1.4.1, possède les caractéristiques suivantes :

- Code source du prototype : 110 Ko
- Code source de la plate forme Aglets : 580 Ko
- Code source d'un agent mobile : 4 Ko

Le taille code de la plate-forme java pour l'exécution (JRE - Java Run Time) est de 28 Mo.

4.3 Description de la plate-forme de mesures

Nous utilisons l'architecture réseau représentée sur la figure 4.7. Le nœud A possède des filtres *Iptables* pour bloquer les trames MAC générées par le nœud C. Le nœud B est alors élu MPR des nœuds A et C quelles que soient leurs positions.

De même nous simulons l'instabilité des liaisons physiques, caractéristique des WLAN, par l'application de filtres IP sur les différentes interfaces des nœuds.

1. IBM Aglet Class Library 2.1.0 - Aglet : 1.2

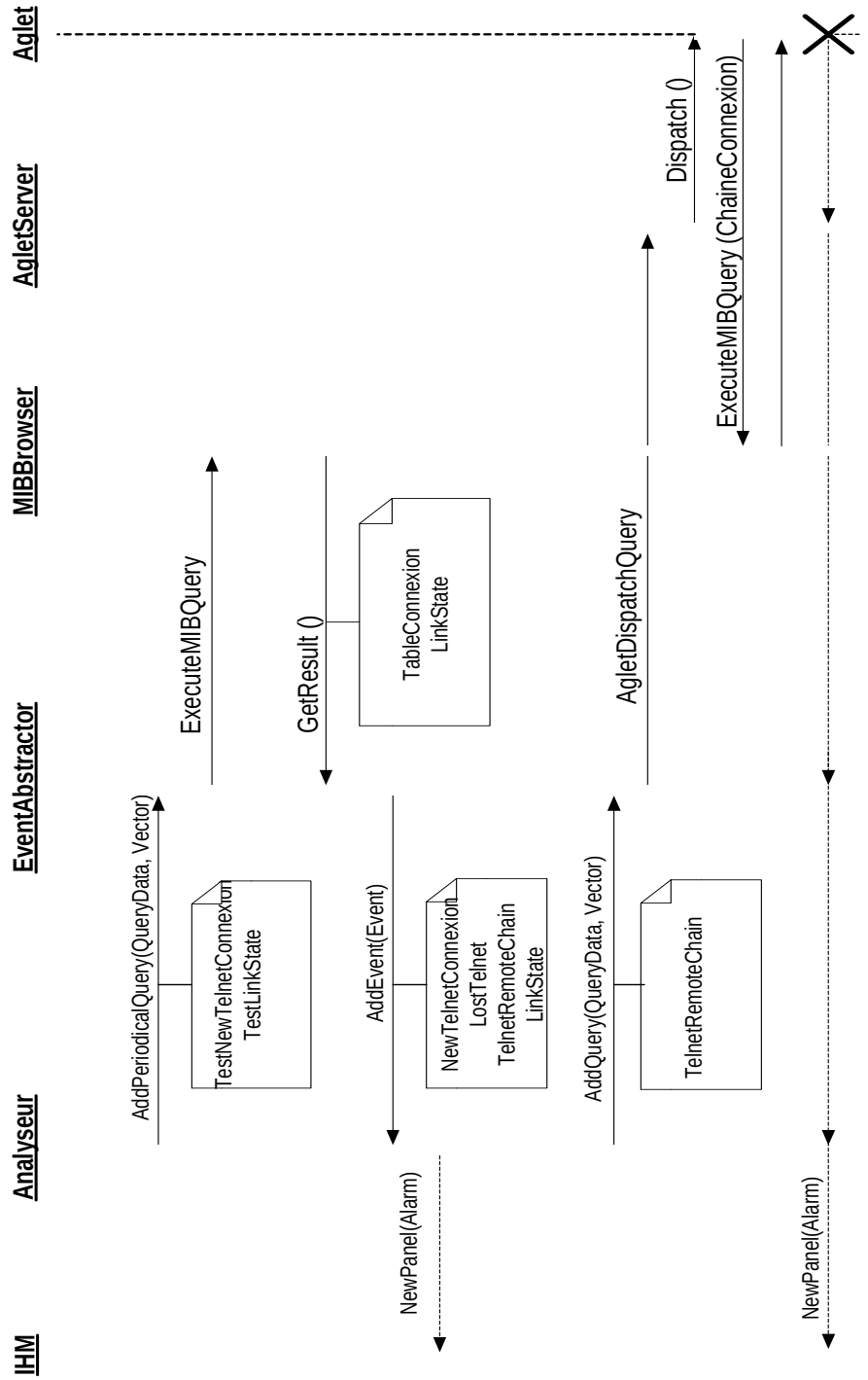


FIG. 4.6 – *Diagramme de séquence d'un LIDS*

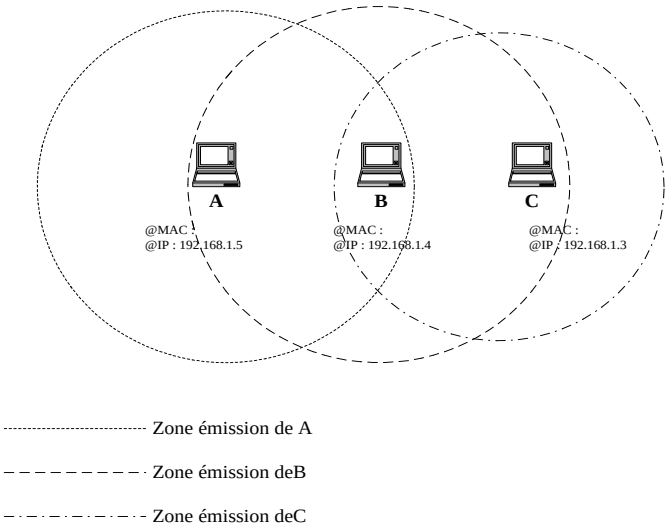


FIG. 4.7 – *Réseau 802.11b en mode ad hoc - routage OLSR*

Nœud	Plate-forme matérielle	Interface 802.11b	Adresse MAC	Adresse IP
A	NEC Power Mate	CISCO PCI-AiroNet 350	00 :0A :8A :A2 :C7:BB	192.168.1.4
B	NEC VersaFC140-256M	CISCO PCM-AiroNet	00 :0A :8A :A2 :AB: 8E	192.168.1.5
C	NEC Power Mate	CISCO PCI-AiroNet 350	00 :0A :8A :A2 :C9:45	192.168.1.3

TAB. 4.1 – *Configuration des nœuds*

Tous les nœuds sont configurés avec le système Linux SuSE 8.2 - Noyau 2.4.20, un Analyseur[2] , un agent SNMP[3], la plate-forme java 1.4.1, la plate-forme à agents Aglet, l'IDS SNORT 2 et le prototype du LIDS.
Le Tableau 4.1 présente les configurations matérielles des différents nœuds.

4.4 Validation fonctionnelle

A chaque attaque est associée une signature représentée par un automate à états finis. Nous présentons ici deux attaques courantes dans un MA-

2. Ethereal 0.9.10- Libpcap 0.7
3. ucd-SNMP 4.2.6

NET : la première, appelée *N_HOP*, cible le protocole de routage OLSR et la deuxième, appelée *Setpstone*, est plus générale et cible l'application Telnet. Dans [72] et [71] nous présentons d'autres types d'attaques sur le protocole de routage.

4.4.1 Description et détection d'une attaque sur le protocole de routage OLSR

4.4.1.1 Description de l'attaque

Il est facile de générer des attaques de déni de service ou d'écoute passive par compromission d'un nœud MPR qui émettra des messages TC modifiés. En outre, il est aussi possible de casser un lien symétrique par des messages *HELLO* erronés. De cette manière, on interrompt les communications avec les voisins immédiats et éloignés, le routage utilisant seulement les liens symétriques. Cette vulnérabilité est exploitée dans l'attaque présentée ci-après.

Ces vulnérabilités, qui reposent sur l'émission de messages modifiés par un nœud malveillant inséré dans le réseau sont, dans leur principe, communes aux autres protocoles de routage ad hoc.

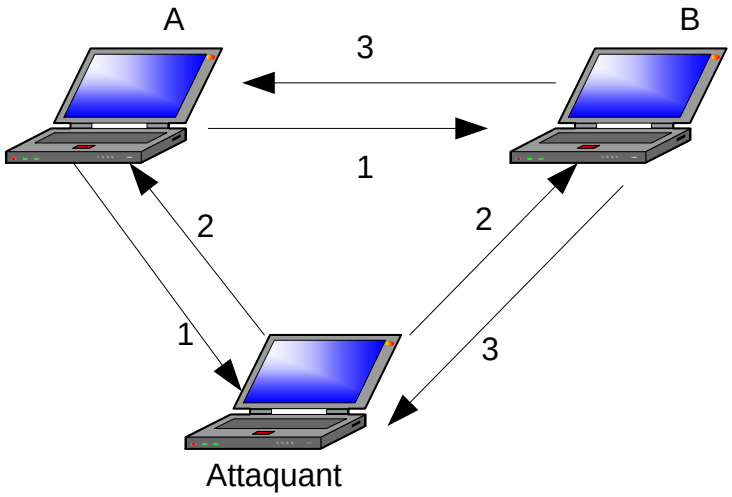


FIG. 4.8 – *Attaque par déni de service du protocole de routage*

La figure 4.8 présente le scénario de l'attaque. Avant l'attaque, les nœuds A et B ont établi un lien symétrique.

1. Le nœud A diffuse un message *HELLO* contenant l'adresse IP (*Internet Protocol*) du nœud B (@IP_B) dans la liste de ses liens symétriques.

2. A la réception de ce message, l'attaquant envoie un message erroné semblant issu de A avec @IP_B dans la liste des liens marqués perdus. B doit changer l'état du lien avec @IP_A en *Heard* et ne peut plus envoyer de messages vers A. A reçoit aussi le message erroné, mais comme rien n'est prévu dans OLSR pour traiter ce problème, un correctif ne sera transmis que par le message *HELLO* suivant.
3. B diffuse un message *HELLO* avec @IP_A dans la liste des liens asymétriques.
4. L'attaquant devra continuer à générer un message erroné à la réception du message de B. Cela aura pour effet de casser la symétrie du lien de A avec B, stoppant ainsi tout trafic de A vers B. B recevra aussi le message erroné.

Le nœud dont le message *HELLO* a été transformé devra détecter la réception d'un message *HELLO* contenant l'adresse de l'une de ses interfaces. L'attaque est difficile à parer : corriger la situation en renvoyant un message *HELLO* correct conduirait le réseau dans un état instable si l'attaquant persiste à renvoyer des messages. L'authentification peut éliminer les attaques générées par des intrus, mais pas celles opérées par des stations légitimes mais corrompues.

4.4.1.2 Signature de l'attaque

Nous présentons sur la figure 4.9 de la page 97 la description de l'automate à états finis associé à la signature de l'attaque.

Description des états :

Nsaut_S0 : état normal, pas d'état *ASYM* du lien

Nsaut_S1 : état *ASYM* du lien déclaré.

Nsaut_S2 : détection possible d'une attaque-de type *Déni de Service*, décision selon les états du lien pendant le prochain *HELLO_INTERVAL*.

Description des événements :

NSaut_E0 : le lien n'a pas été déclaré *ASYM* pendant le dernier *HELLO_INTERVAL*.

NSaut_E1 : le lien a été déclaré *ASYM* puis *SYM* ou *MPR* pendant un ou plusieurs *HELLO_INTERVAL*.

NSaut_E2 : le lien a été déclaré *ASYM* pendant un ou plusieurs *HELLO_INTERVAL*.

NSaut_Attaque : détection de l'attaque.

Les changements d'états se font sur l'apparition d'événements détectés lors de la lecture périodique de variables dans la MIB. Le test des variables de

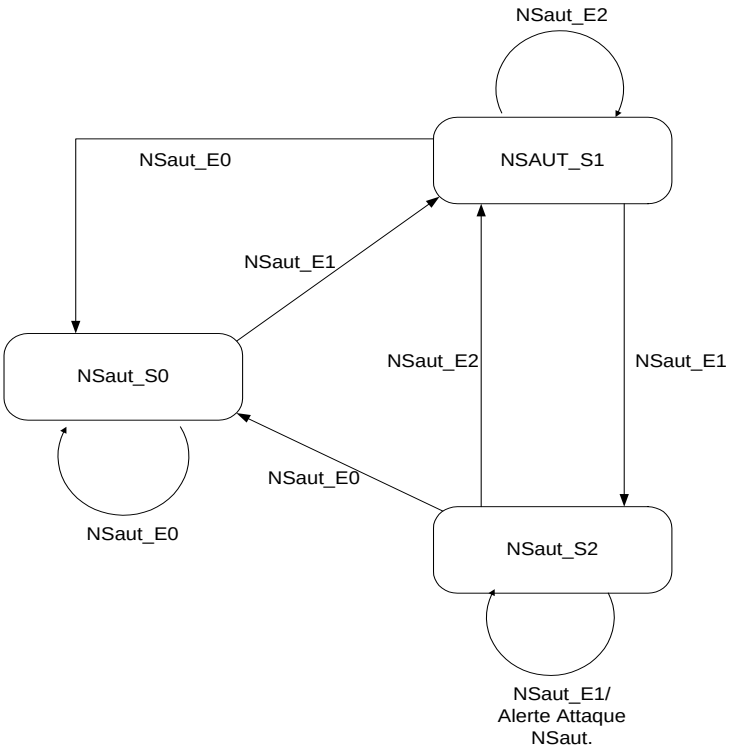


FIG. 4.9 – *Signature de l'attaque par déni de service*

la MIB est réalisé périodiquement toutes les 2 secondes (Valeur du paramètre *HELLO_INTERVAL* préconisée par OLSR).

4.4.1.3 Description des variables de la MIB utilisées pour la détection de l'attaque

Nous avons réalisé une extension de la MIB standard (voir annexe G). Il s'agit d'une table dont chaque élément est formé des trois variables suivantes :

OlsrNeighborState : état courant du lien avec le voisin (Sym, Asym, MPR, Lost),

OlslrPreviousNeighborState : état précédent du lien,

OlsrNeighborAddress : adresse IP du voisin.

Ces variables sont mises à jour à chaque réception ou émission d'un message *HELLO*. Les événements résultent du changement d'état des variables, qui sont testées périodiquement toutes les 2 secondes (*HELLO_INTERVAL*). Le fichier texte de définition des variables *OlsrNeighborState, OlslrPrevious-NeighborState, OlsrNeighborAddress* est donné dans l'annexe G.

4.4.2 Détection de l'attaque *Setpstone*

L'attaque *Stepstone* [78] correspond à la création d'une chaîne de connexions (*telnet* par exemple) vers un terminal distant. Une attaque de ce type précède généralement des actions plus graves car elle rend plus difficile la localisation de l'attaquant. Cette technique est couramment utilisée par les attaquants pour masquer leur machine, et donc l'origine réelle de l'attaque.

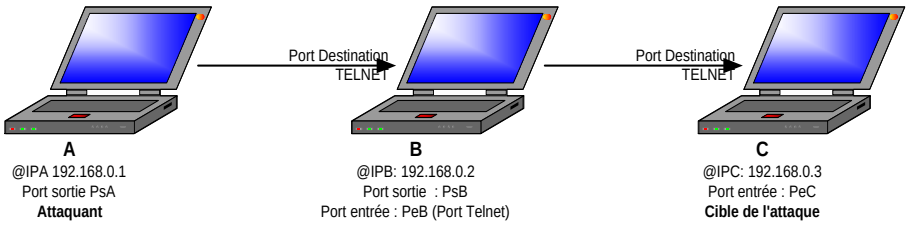


FIG. 4.10 – *Descrption de l'attaque Stepstone*

La figure 4.10 illustre le principe de l'attaque *stepstone*. L'attaquant, initie une connexion avec un autre nœud du réseau *(Nœud B)* . Depuis ce nœud, l'attaquant poursuit son chemin vers un autre nœud *(Nœud C)* et ainsi de suite jusqu'à atteindre l'hôte cible de l'attaque.

4.4.2.1 Signature de l'attaque

La détection d'une attaque de type *stepstone* se décompose en deux phases distinctes.

Tout d'abord, dès qu'un nœud est la cible d'une connexion *Telnet* entrante (événement local STEPSTONE_E0), il interroge l'hôte initiateur de la connexion afin de connaître son origine (requête distante STEPSTONE_Q1) :

- Si l'hôte interrogé ne gère aucune connexion entrante associée au port de sortie (c'est donc que le nœud est à l'origine de la chaîne), il le signale à l'initiateur de la requête en lui transmettant l'événement STEPSTONE_E1. Un nœud recevant un tel événement doit mémoriser la chaîne correspondante. Elle est composée d'exactement deux éléments : le receveur et l'expéditeur de l'événement, ce dernier y figurant en tant que source de la chaîne.
- Dans le cas contraire, l'hôte interrogé transmettra à l'initiateur de la requête, la chaîne existante (événement STEPSTONE_2). Sur réception de ce type d'événement, l'initiateur de la requête mémorise la chaîne envoyée et y ajoute l'émetteur du message en tant que dernier maillon. Une chaîne est éliminée quand la connexion correspondant à

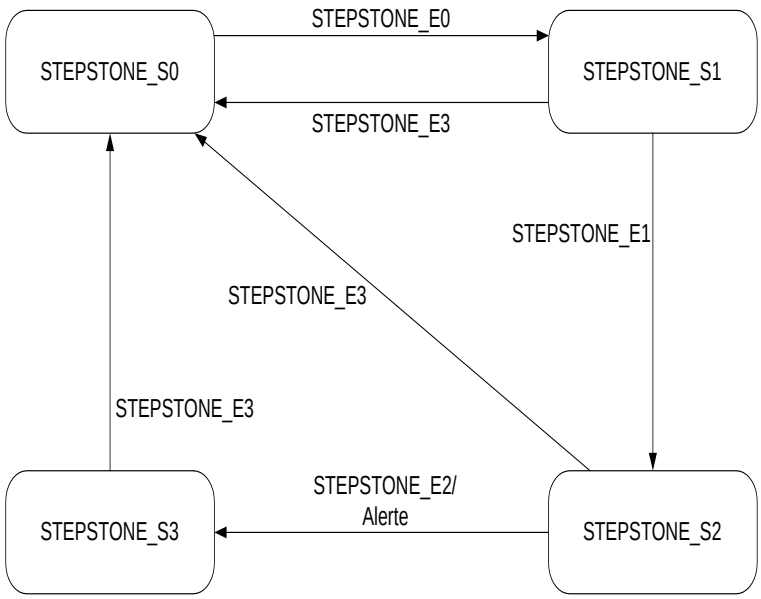


FIG. 4.11 – *Signature de l'attaque Stepstone (Chaîne de connexion Telnet)*

son dernier maillon se ferme. Ceci est signalé par l'événement local STEPSTONE_E3.

La réception d'un événement STEPSTONE_E2 indique la présence d'une chaîne de connexions, comprenant au moins trois nœuds.

La requête STEPSTONE_Q0 (en attente d'un des événements STEPSTONE_E0 ou STEPSTONE_E3) doit être exécutée de manière périodique. En revanche, on remarque que la soumission de la requête STEPSTONE_Q1 (en attente d'un des événements STEPSTONE_E1 ou STEPSTONE_E2) est, quant à elle, déclenchée en réaction à une transition d'état (apparition ou disparition d'une connexion TCP).

L'automate de la figure 4.11 représente la machine d'état de la signature de cette attaque.

La table *tcpConnTable* de la MIB SNMP standard (voir l'annexe F) contient les informations relatives aux connexions TCP actives. Pour chacune d'elles, elle indique l'adresse IP de la machine locale, le port TCP sur la machine locale, l'adresse IP de la machine distante, et le port TCP sur la machine distante. Dans la version actuelle de notre prototype, sur chaque nœud, un module spécifique est chargé de détecter les associations entre les connexions entrantes et sortantes.

4.4.2.2 Détection de l'origine de la chaîne des connexions Telnet

La détection de l'attaque *Stepstone*, décrite ci-dessus, permet d'illustrer le rôle des agents mobiles dans l'architecture présentée sur la figure 3.20. Nous considérons ici une connexion Telnet réalisée en cascade sur trois nœuds équipés d'un même LIDS et d'un agent SNMP. Cette chaîne de connexions est représentée sur la figure 4.10.

Nous nous intéressons au comportement du LIDS du nœud C.

- Première étape : sur C, l'*Analyseur* est informé de l'arrivée d'une nouvelle connexion TCP. Dans notre exemple cette dernière est identifiée de manière unique par ses adresse et port sources (@IP_B et PsB).
- Deuxième étape : sur C, l'*Analyseur* doit maintenant déterminer si B est le premier nœud de la chaîne. Cette requête est transmise au gestionnaire de distribution qui envoie alors un agent mobile sur B pour collecter cette information.
- Troisième étape : sur B, l'agent émet une requête au module de *Gestion de la distribution* pour obtenir la chaîne de connexion dont le port de sortie est PsB. Si l'IDS de B ne possède pas encore cette information (i.e. si son propre agent chargé de la collecter n'est pas rentré), l'agent attendra qu'elle soit disponible. Dans notre exemple c'est la chaîne de connexion dont l'origine est @IP_A et PsA qui est identifiée.
- Quatrième étape : l'agent retourne sur le nœud C avec les données collectées sur B et les transmet ensuite à l'Analyseur.
- Cinquième étape : l'IHM affiche la chaîne des connexions. La réponse à la détection d'une attaque *Stepstone*, i.e. la fermeture de la connexion, peut-être réalisée à l'initiative de l'opérateur.

4.4.3 Tests de détection

Dans l'environnement de tests, les deux attaques que nous venons de présenter sont correctement détectées à leur exécution. La première attaque *N_HOP* est détectée localement sans qu'il soit nécessaire de faire appel à un agent mobile. La détection de l'attaque *StepStone* fait appel à la coopération entre LIDS. Des agents mobiles sont créés par les LIDS des nœuds détectant une connexion Telnet entrante et se déplacent de nœud en nœud pour identifier la chaîne des connexions.

La détection de ces attaques montre la capacité de notre architecture à détecter localement et de manière distribuée des attaques de niveau réseau ou application. Les détections des attaques sont effectuées sur la base des données collectées par les senseurs dans les MIB.

4.5 Évaluation qualitative de la coopération

Dans cette section, nous analysons le comportement des agents mobiles et celui des connexions client/serveur lorsque les liens physiques du réseau sont instables

Nous utilisons un nouveau scénario d'attaque, basé sur le balayage de ports (*ports scanning*). Pour effectuer un diagnostic plus précis de l'attaque, quand un balayage de ports est détecté sur son nœud, le LIDS recherche les attaques du même type réalisées sur les autres nœuds de sa communauté.

Le balayage de ports a pour objectif d'identifier les ports actifs d'un nœud. Bien que cette opération ne représente pas une attaque immédiate contre un service offert par le nœud, elle constitue très fréquemment l'étape préalable au lancement d'attaques plus sérieuses. Les nœuds d'un MANET directement accessibles par des nœuds étrangers à leur communauté sont particulièrement exposés à ces actions d'investigation. Si le résultat d'un balayage des ports permet d'identifier que le port Telnet est ouvert, une attaque du type *Stepstone* présentée dans la section 4.4.2 peut, par exemple, être lancée.

Pour ce test, nous n'utilisons pas les données des MIB pour détecter le balayage de ports car les requêtes SNMP effectuées sur des MIB distantes sont transportées par le protocole UDP qui, contrairement à TCP, n'apporte aucune fiabilité aux échanges client/serveur. Afin de disposer sur chaque nœud d'une base de données accessible via des connexions TCP, nous utilisons l'IDS SNORT. Ce dernier utilise les données collectées sur le réseau. Il est configuré pour enregistrer ses alertes dans une base de données MySQL locale.

Les expérimentations ont pour objectif d'analyser la collecte de ces alertes réalisée par des requêtes MySQL distantes en mode client/serveur et par des requêtes locales réalisées par des agents mobiles préalablement déplacés.
Les expérimentations sont réalisées en suivant les étapes décrites ci-dessous :

- **Première étape** : analyse du comportement des échanges client/serveur à travers le réseau.

  Sur une interface, nous simulons une rupture de la connexion physique lors de l'envoi de requêtes SQL et capturons avec l'outil ETHEREAL [4] les trames réseaux générées.

  Sur chaque nœud, SNORT est configuré pour enregistrer les attaques détectées dans une base de données MySQL locale. Les traces, représentées sur la figure 4.12, ont été capturées lors d'une requête MySQL effectuée depuis le nœud A (192.168.1.3) vers la base MySQL située sur

4. http://www.ethereal.com

le nœud C (192.168.1.5).

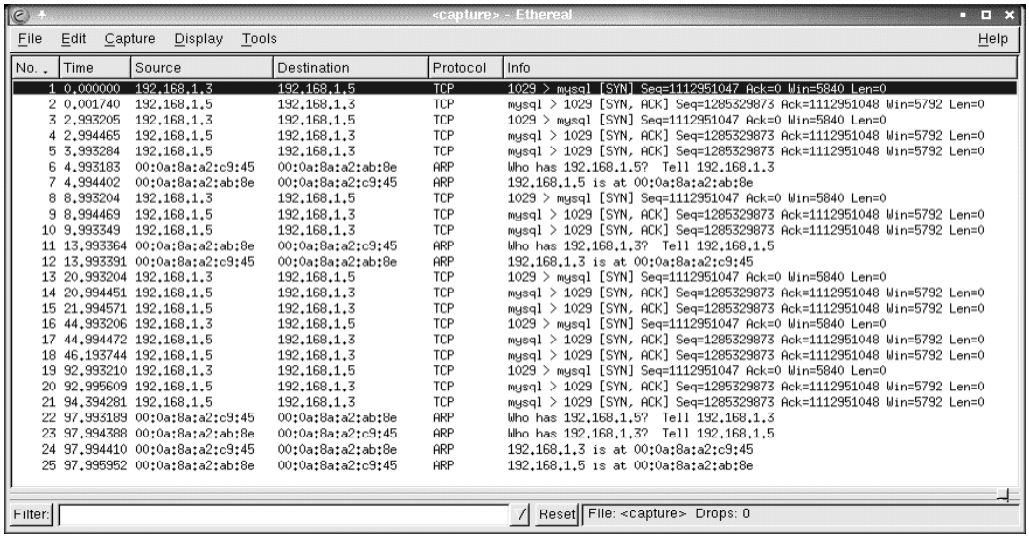


FIG. 4.12 – *Requêtes client/serveur avec des connexions réseau instables*

Nous constatons sur la figure 4.12 que suite à une demande de requête, la couche TCP tente d'ouvrir une connexion pendant 98 secondes par l'émission de segments TCP avec le drapeau SYN.
Au delà de ce délai, la demande d'ouverture de la connexion TCP est abandonnée, de même que l'interrogation de la base MySQL distante. Une nouvelle tentative de connexion à la base MySQL ne peut être réalisée qu'en générant une nouvelle requête.

- **Deuxième étape** : analyse du comportement des agents mobiles. Nous utilisons un aglet qui se déplace sur les différents nœuds du réseau pour interroger les bases MySQL de SNORT et ramener les informations. Le LIDS crée un agent mobile autonome et asynchrone programmé pour attendre qu'une route vers la destination soit établie avant d'ouvrir une connexion TCP.
  A l'émission de l'agent, nous simulons une rupture de la connexion d'une durée supérieure à celle mesurée dans le scénario de la première étape. Nous constatons, sur les traces de la figure 4.13, que l'agent mobile utilise la connexion dès que celle-ci est de nouveau disponible sans limite de temps.

- **Troisième étape** : analyse des expérimentations réalisées dans les deux étapes précédentes. Nous constatons que le comportement autonome et

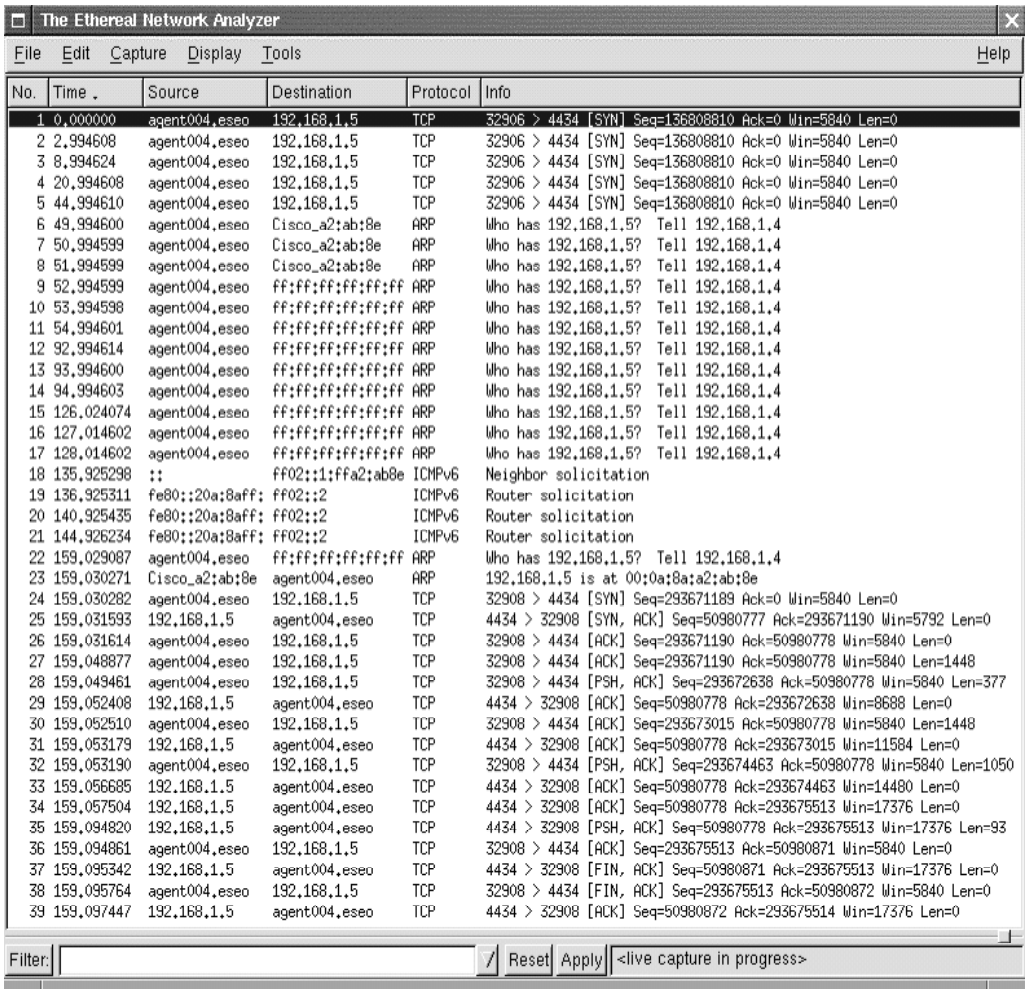


FIG. 4.13 – *Déplacement d'un agent avec des connexions physiques instables*

asynchrone des agents mobiles leur permet de s'adapter aux évolutions de la topologie d'un MANET pour atteindre leur destination. De ce fait, dans un réseau ad hoc, les échanges d'information par le biais d'agents mobiles sont plus fiables que des échanges client/serveur à travers le réseau et décharge l'application qui les a créés de tout contrôle.

4.6 Évaluations quantitatives des agents mobiles

Lors de son déplacement pour s'exécuter sur un nœud distant, le comportement d'un aglet est différent selon les ressources logicielles (classes java) disponibles sur ce nœud. Ce comportement à une influence importante sur la charge réseau induite et sur les temps de réponse.

4.6.1 Les modes de déplacement d'un aglet

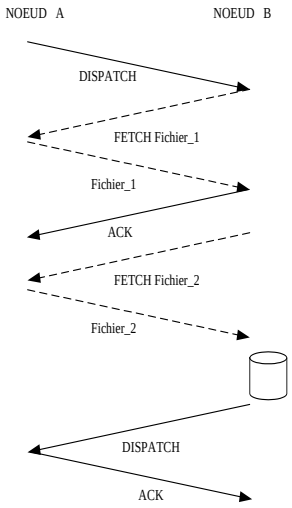


FIG. 4.14 – *Échanges pour le déplacement d'un aglet*

La plate-forme Aglet transfère les agents en trois phases, représentées sur la figure 4.14. Tout d'abord l'aglet est envoyé par une commande DISPATCH. Si la plate-forme destination ne possède pas les définitions des classes nécessaires à l'exécution de l'aglet, elle en fait la demande, par une commande FETCH, à l'émetteur de l'aglet. Un accusé de réception (DISPATCH) est transmis pour confirmer la disponibilité de toutes les classes demandées.

4.6.2 Mesure du temps d'accès aux variables d'une MIB par le déplacement d'un aglet à un saut.

Compte tenu du mode de déplacement d'un aglet, nous réalisons trois scénarios de mesures pour effectuer des requêtes MIB sur un nœud situé à

un saut :

- Déplacement d'un aglet sans code pré-installé, avec retour de toutes les variables SNMP collectées.
- Déplacement d'un aglet avec code pré-installé, avec retour de toutes les variables SNMP collectées.
- Déplacement d'un aglet avec code pré-installé, sans retour des variables SNMP collectées.

Nombre de requêtes SNMP	10	25	50	100	200
Requêtes directes	*43,5*	*109*	*217*	*435*	*870*
Sans code pré-installé	1697	1747	1825	1990	2570
Avec code pré-installé	300	330	390	540	**740**
Avec code pré-installé et sans retour des variables	175	200	260	**410**	**580**

TAB. 4.2 – *Temps de collecte de variables MIB (en milliseconde)*

Sur le tableau 4.2 et sur la figure 4.15, nous constatons qu'il faut collecter un grand nombre de variables, entre 80 et 140 selon le type de traitement à effectuer sur ces variables, pour que les aglets deviennent plus performants que des requêtes directes si nous considérons uniquement le temps de collecte. Dans un processus de détection d'intrusions basé sur la collecte d'information dans une MIB, ce seuil est selon nous trop élevé pour pouvoir justifier, à lui seul, le choix de la plate-forme Aglet pour collecter les informations situées sur d'autres nœuds.

4.6.3 Évaluation de la charge réseau induite par le déplacement d'un aglet

Nous évaluons les charges réseau induites par la collecte d'information dans la MIB effectuée à distance par des requêtes SNMP et localement par un aglet préalablement déplacé. Pour le premier déplacement de l'aglet, le code associé nécessite deux commandes DISPATCH et deux commandes FETCH (cf. figure 4.14, page 104). Lors d'un déplacement de l'aglet sans les fichiers associés, deux commandes DISPATCH sont nécessaires. Le nombre d'octets échangés selon le mode de transfert de l'aglet est représenté dans le Tableau 4.3.

Nous pouvons comparer ces chiffres au nombre de données échangées sur le réseau lors d'une requête SNMP et de sa réponse (voir le Tableau 4.4).

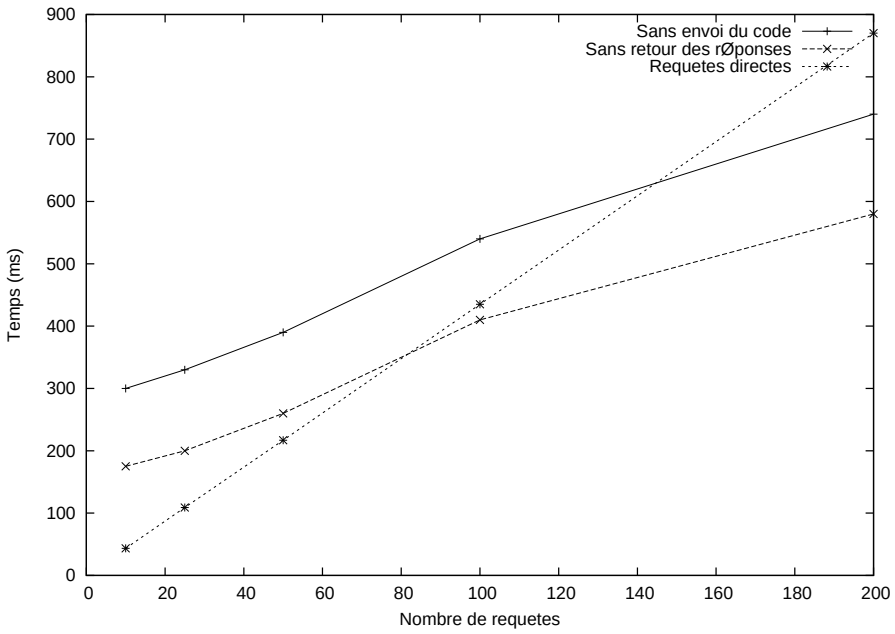


FIG. 4.15 – *Temps d'obtention des informations avec un Aglet*

	Déplacement du code de l'aglet	Sans déplacement du code de l'aglet
Échanges ATP	26 695	19 208
Échanges TCP/IP	5476	3 197
Total	**32 284**	**22 405**

TAB. 4.3 – *Charge réseau induite par le déplacement d'un aglet (en octet)*

802.11b	IP	UDP	SNMP	Variables Binding
30	20	8	41	2 à N

TAB. 4.4 – *Format d'une requête SNMP (en octet)*

Nous représentons sur la figure 4.16 de la page 107 les droites représentant le volume des données échangés sur le réseau selon différents modes d'accès.

Nous constatons à la lecture des courbes de la figure 4.16 qu'il faut effectuer plus de 120 requêtes de variables simples SNMP dans une MIB pour générer sur le réseau un volume de données équivalent à celui du déplacement d'un aglet sans son code.

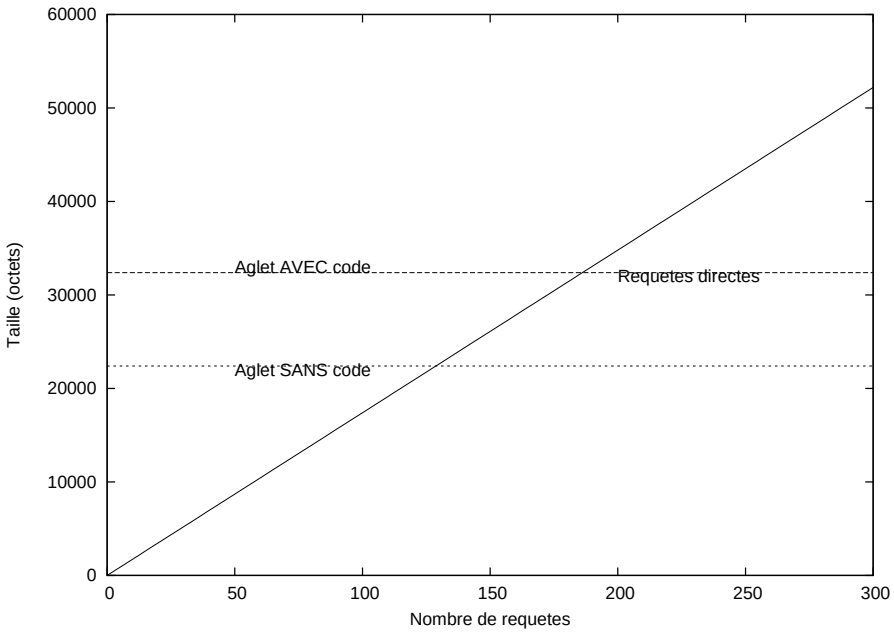


FIG. 4.16 – *Occupation du réseau en fonction du nombre de requêtes*

En conclusion nous constatons que la collecte de données avec des agents mobiles, Aglet, dans une MIB située sur un nœud distant, génère un trafic réseau important comparé à celui d'une requête SNMP simple.

4.7 Analyse des mesures de performances

Nous rappelons que si un LIDS utilise des requêtes SNMP/UDP à travers un réseau, ces dernières n'ont alors aucune garantie de fiabilité en raison du mode datagramme propre à UDP. Les évaluations quantitatives réalisées nous montrent que la collecte d'information dans une MIB située sur un nœud distant est plus lente et charge plus le réseau quand celle-ci est réalisée par un aglet que par une requête de type SNMP à travers le réseau.

Toutefois, il convient de noter que sur des critères quantitatifs, la performance des agents mobiles dépend du rapport entre le volume du code et le volume des données à déplacer. Dans nos mesures, le volume des données déplacées représente quelques octets. De plus, la plate-forme à agents mobiles générique Aglet utilise des protocoles de transfert complexes (cf. Tableau 4.3). Ce contexte propre à notre implémentation explique qu'il faut réaliser un nombre élevé de requêtes SNMP pour commencer à tirer profit des agents mobiles si l'on ne considère que les critères quantitatifs (cf. le Tableau 4.3).

Une évaluation de performance entre deux plate-formes à agents mobiles, dont la plate-forme Aglet, et un modèle client/serveur basé sur RMI est présentée dans [25]. On constate que des gains d'efficacité significatifs peuvent être obtenus en utilisant des agents mobiles. Plus précisément la plate-forme optimisée construite spécialement a permis de réduire par 7 les temps de migration d'un agent sur un LAN ETHERNET. Cette différence s'explique par la complexité du protocole ATP (Aglet Transfert Protocol) qui nécessite plusieurs envois de messages pour déplacer un agent.

Si nous considérons uniquement les critères quantitatifs, il sera nécessaire, selon nous, d'utiliser une plate-forme à agents mobiles légère, spécifique et adaptée aux environnements sans fil.

Les expérimentations montrent aussi que les agents mobiles, programmés pour se déplacer de nœud en nœud, adaptent leur comportement aux modifications de la topologie. L'autonomie des agents mobiles est une caractéristique qui permet aussi de libérer une application communicante du contrôle d'une connexion TCP établie de bout en bout.

4.8 Coopération locale d'un LIDS

L'architecture d'IDS distribuée que nous proposons est construite autour de LIDS installés sur chaque nœud du MANET. Certains nœuds peuvent aussi disposer d'un IDS local, autre que le LIDS, pour assurer leur protection. Nous étudions ici la capacité de notre architecture à intégrer ces systèmes.

Nous proposons dans cette situation, si les ressources du nœud le permettent, d'utiliser conjointement et de façon complémentaire l'action des deux IDS et le système de coopération à base d'agents mobiles.

Cette coopération locale, présentée dans [60], offre au LIDS, tout en conservant ses caractéristiques propres, la possibilité :

- d'étendre immédiatement le nombre d'attaques détectées sans concevoir de nouvelles signatures,
- de bénéficier du renfort d'IDS optimisés pour la détection de certains types d'attaques indépendantes du contexte ad hoc,
- de cibler la détection d'attaques spécifiques au contexte ad hoc,
- de bénéficier de nouvelles données, sans développer de nouveaux senseurs,
- de disposer de différentes méthodes de détection,
- d'élargir ses capacités de détection, voire de corréler les alertes locales.

Pour réaliser cette coopération locale entre un LIDS et un IDS, nous proposons de faire évoluer l'architecture interne des LIDS et de considérer, dans ce cas, un IDS comme un nouveau type de senseur local. A cette extension de senseur, nous associons une signature générique dont un événement est associé à l'alerte générée par l'IDS. La prise en compte par le LIDS du type d'alerte lui permet ensuite d'effectuer un traitement spécifique, comme d'utiliser les agents mobiles pour collecter de nouvelles informations sur d'autres nœuds.

Pour valider cette approche, nous menons des expérimentations avec des nœuds équipés d'un LIDS dont la nouvelle architecture interne est représentée sur la figure 4.17 de la page 110 et du NIDS SNORT (*Network Intrusion Detection System*) [5] [46].

Dans cette section, nous commençons par une brève description des caractéristiques de l'IDS SNORT. Nous présentons ensuite les expérimentations réalisées pour valider cette nouvelle capacité de coopération des LIDS. Pour cela, nous testons la détection d'une attaque de type *StepStone* dirigée contre un nœud équipé des IDS SORT et LIDS, et nous terminons par une analyse des résultats obtenus.

4.8.1 Coopération entre un LIDS et SNORT

Avec plus de 100 000 installations recensées dans le monde, SNORT est aujourd'hui l'IDS le plus répandu. Ce logiciel *Open Source* a été créé en 1998 par Marty Roesch comme un outil personnel d'aide à l'analyse du trafic réseau dont les évolutions bénéficient régulièrement des contributions de nombreux développeurs de la communauté Open-Source.
SNORT est indépendant des plate-formes, entièrement paramétrable, et dispose d'une base de signatures conséquente. De plus, pour nous, le fait de disposer d'un code source public représente une plus grande garantie de fiabilité et d'évolution que les logiciels commerciaux dont le code source est généralement inaccessible aux utilisateurs.

SNORT peut être utilisé comme un enregistreur de paquets (*sniffer*) ou comme un Système de Détection d'Intrusions basé sur le réseau (*NIDS - Network Intrusion Detection System*). Pour cette dernière fonction, il analyse les données collectées sur le réseau et émet une alerte si le trafic est jugé suspect.
Actuellement, SNORT peut détecter une très grande variété d'attaques et

5. http://www.snort.org

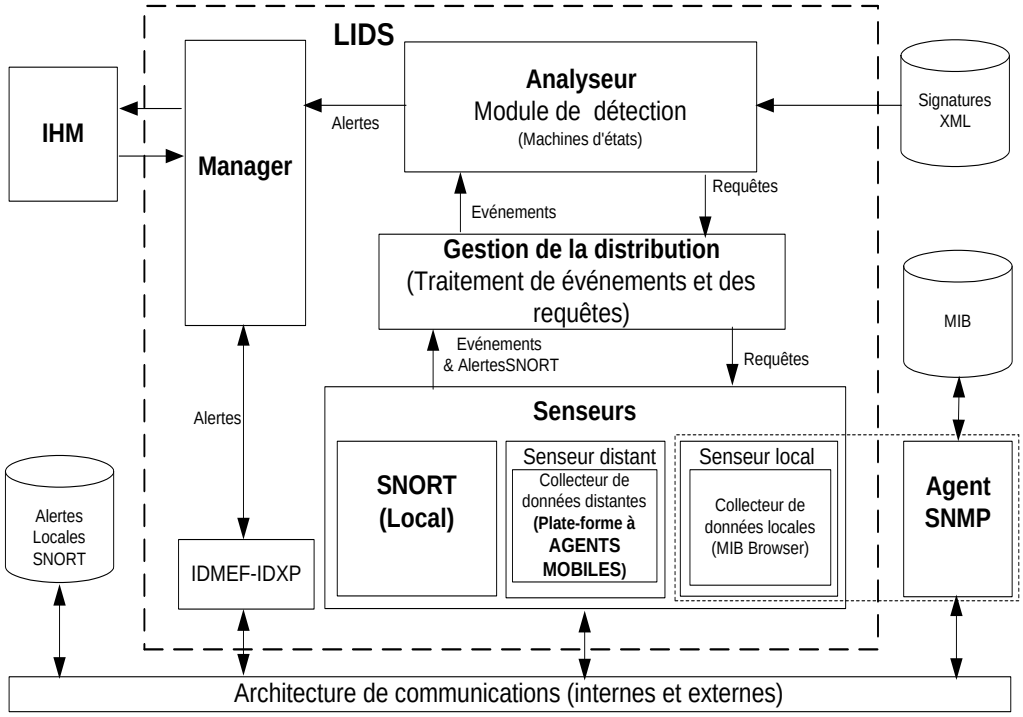


FIG. 4.17 – *Nouvelle architecture interne d'un LIDS*

d'événements tels que les débordements de tampons, les balayages de ports furtifs, les attaques CGI, les tentatives d'identification d'OS et bien d'autres encore.

4.8.1.1 Principales caractéristiques de SNORT

L'architecture interne de SNORT présentée dans [46] est donnée sur la figure 4.18. Elle est construite autour de cinq composants principaux :

- La bibliothèque **libpcap** dont la fonction est de collecter les paquets véhiculés par le réseau. Elle est indépendante des plate-formes matérielles et des systèmes d'exploitation. Elle est chargée d'extraire les paquets directement de la carte réseau.
- Le **décodeur de paquets** analyse les protocoles de la couche 2 à la couche 4 pour stocker toutes les données d'un paquet dans une structure de données qui sera analysée par les préprocesseurs.
- Les **préprocesseurs** examinent le trafic à la recherche d'activités suspectes et préparent les paquets pour que le **moteur de détection** puisse les interpréter correctement.

Certaines attaques ne peuvent pas être identifiées par une signature, il est donc nécessaire d'utiliser des préprocesseurs spécifiques pour détecter ces activités suspectes. A ce titre, SNORT dispose de préprocesseurs pour la détection d'attaques courantes, comme *Frag2*, pour détecter les attaques basées sur la fragmentation de paquets IP.

- Le **moteur de détection** est le composant principal de SNORT. Il réalise deux fonctions majeures : l'analyse des règles et la détection des signatures. Le moteur de détection construit ses signatures d'attaques en analysant les règles de SNORT lors du démarrage. Puis, il applique ce système de règles dans l'ordre de chargement en mémoire.
  Le moteur de détection construit un arbre de décision sous la forme de listes chaînées. Les nœuds recherchent dans chaque paquet des éléments de signature de plus en plus précis. Par exemple, le moteur commence par détecter si un paquet est de type TCP, puis il vérifie l'adresse source etc.. Dès que la signature d'une attaque est identifiée, SNORT écrit les données alertes définies dans la règle.
- Les **modules de sorties** enregistrent les données d'alertes dans un fichier simple ou dans une base de données. Différents modules de sortie sont disponibles, comme par exemple :

  alert_fast : pour l'enregistrement rapide et basique d'une ligne dans un fichier.

  alert_full : pour l'enregistrement complet dans des fichiers contenus dans un fichier portant le nom de l'IP.

  alert_syslog : pour transmettre les rapports d'alertes au serveur syslog.

  log_tcpdump : pour des enregistrements au format TcpDump.

4.8.1.2 Les bases de la coopération entre un LIDS et SNORT

Les nombreux modules de sortie disponibles offrent différentes possibilités pour intégrer SNORT comme un *senseur* dans l'architecture des LIDS.

Pour étendre l'interopérabilité des LIDS à d'autres IDS que SNORT, nous retenons la solution basée sur IDMEF (*Intrusion Detection Message Exchange Format*) [6] qui propose une structure générique pour les messages d'alertes échangés entre les *Managers*. Les messages IDMEF sont décrits au format XML (*eXtensible Markup Langage*[7]) dont nous donnons les principales caractéristiques dans l'annexe C.

6. www.ietf.org/ internet-drafts/draft-ietf-idwg-idmef-xml-11.txt
7. http://www.w3.org/XML/

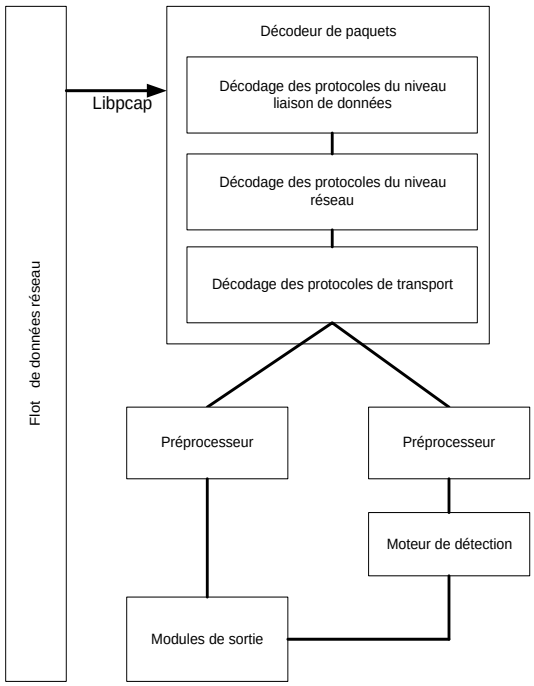


FIG. 4.18 – *Structure interne de Snort*

Un message IDMEF est composé de deux structures, représentées sur la figure 4.19 : la structure **Alert** et la structure **Heartbeat**.

La structure **Alert** représente un ou plusieurs[8] événements levés par l'IDS. On y retrouve les principales informations concernant :

- l'analyseur à l'origine de l'alerte,
- le moment où l'alerte a été créée,
- le moment où l'alerte a été détectée,
- l'heure de l'analyseur,
- les sources des présumées attaques,
- les destinations des présumées attaques,
- le nom de l'alerte,
- l'impact de l'événement,
- des informations diverses.

La structure **HeartBeat** est utilisée par l'analyseur pour indiquer son statut au *Manager*. L'analyseur envoie ces messages régulièrement avec un intervalle de temps défini pour indiquer qu'il est actif. L'absence de certains messages peut alors indiquer un problème de transmission sur le réseau.

8. en fonction de l'IDS

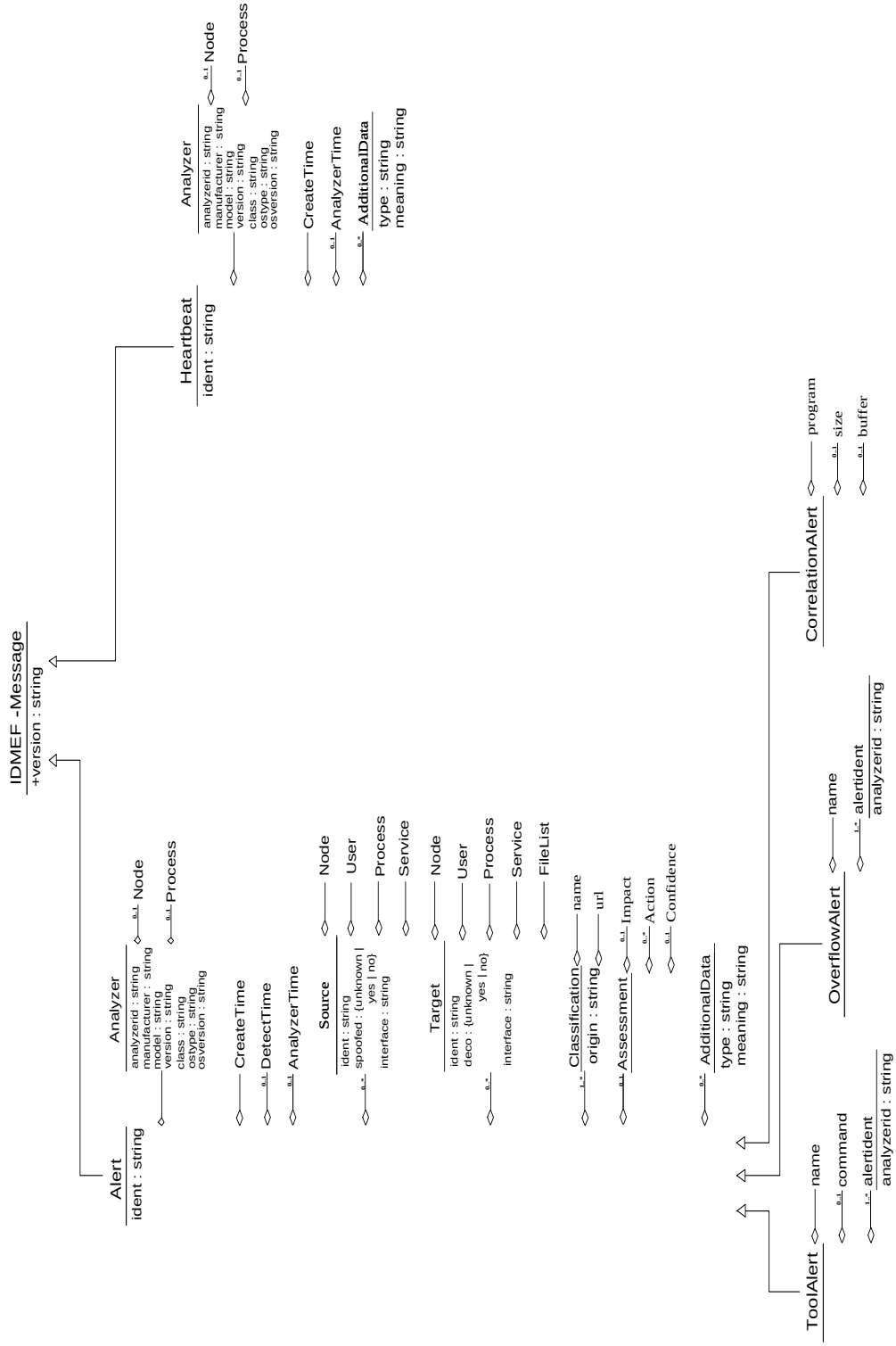


FIG. 4.19 – *Format d'alerte IDMEF*

Nous enregistrons les messages d'alerte générés par SNORT au format IDMEF/XML dans un fichier de log et nous définissons une nouvelle signature d'attaque générique pour toutes les alertes générées par SNORT. Suite à l'enregistrement d'une alerte dans le fichier de log, un événement est transmis au module *Gestion de la Distribution* pour permettre au LIDS de prendre en compte l'attaque détectée.

4.8.2 Validation expérimentale : détection de l'attaque Stepstone

Nous reprenons le scénario de l'attaque *Stepstone* présentée dans la section 4.4.2 page 98 et la plate-forme d'expérimentation présentée dans la section 4.3 page 92.

La configuration et les versions des logiciels utilisés pour ces expérimentations sont données dans l'annexe E.

Pour détecter une attaque de type *Stepstone*, **SNORT** est configuré avec la signature ci-dessous pour détecter les connexions Telnet entrantes :

```
alert tcp $EXTERNAL_NET any -> $TELNET_SERVERS 23
    (flags:S; msg:"TELNET open session"; idmef:default;)
```

Un message d'alerte, au format présenté ci-dessous, est enregistré dans le fichier de log, à chaque ouverture d'une nouvelle connexion Telnet :

```
<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD IDMEF
          v1.0//EN" "idmef-message.dtd">
<IDMEF-Message version="1.0">
  <Alert ident="1">
    <Analyzer analyzerid="-----" model="snort" version="1.9.0">
      <Node>
        <name>linux</name>
      </Node>
    </Analyzer>
    <CreateTime ntpstamp="0xc30afd1e.0xd49e3433">
                2003-09-11T13:54:38Z</CreateTime>
    <Source>
      <Node>
        <Address category="ipv4-addr">
          <address>192.168.4.16</address>
        </Address>
      </Node>
      <Service>
```

```
        <port>23</port>
      </Service>
    </Source>
    <Target>
      <Node>
        <Address category="ipv4-addr">
          <address>192.168.4.18</address>
        </Address>
      </Node>
      <Service>
        <port>1107</port>
      </Service>
    </Target>
    <Classification origin="vendor-specific">
      <name>TELNET login incorrect</name>
      <url>No URL available</url>
    </Classification>
  </Alert>
</IDMEF-Message>
```

La prise en compte par le LIDS du message d'alerte généré par SNORT lui permet d'être informé de l'arrivée d'une connexion Telnet et d'envoyer un agent mobile pour rechercher le nœud à l'origine de la connexion Telnet.

4.8.3 Analyse de la coopération d'un LIDS avec un autre IDS

Pour la détection de l'attaque *Stepstone* sur le nœud disposant d'un LIDS et de SNORT, le fichier XML de définition des signatures permet de sélectionner le senseur (MIB SNMP ou paquets réseaux) actif pour la détection de l'arrivée d'une nouvelle connexion Telnet. La mise à jour des variables de la MIB doit rester active sur chaque nœud du réseau. Car si ces variables ne sont pas utilisées par le LIDS du nœud, elles restent accessibles aux requêtes effectuées par les agents mobiles à la demande des autres LIDS.

Nous constatons que l'extension proposée pour prendre en compte les capacités de détection d'un IDS dans l'architecture globale est fonctionnelle. L'intégration de l'IDS SNORT sur notre palte-forme de tests, au sein de l'architecture, permet au LIDS de cibler les attaques spécifiques des réseaux ad hoc, comme celles dirigées contre le protocole de routage.

4.8.4 Conclusions sur les expérimentations réalisées dans l'environnement de tests

Les expérimentations réalisées avec le prototype et la plate-forme de tests permettent de valider les aspects fonctionnels de la détection des intrusions et du système de coopération par agent mobile.

L'ensemble de l'architecture dispose d'un mécanisme de coopération fiable basé sur les agents mobiles dont le mode de déplacement de nœud en nœud lui permet de s'adapter à la stabilité des connexions (voir 3.13 page 75).

Nous montrons aussi sur la plate-forme de tests que l'occupation du réseau résultant du déplacement des agents est réduite par une préinstallation sur chaque nœud des ressources logicielles nécessaires à leur activité. Bien que fonctionnelle dans les réseaux ad-hoc la plate forme Aglet induit une charge importante pour gérer le déplacement d'un agent mobile. Nous montrons par les simulations que quand la taille des données déplacées par un agent croit le taux de transferts réussis diminue (Voir 3.17 page 79).

L'utilisation des données des MIB pour la détection des intrusions permet aux LIDS et aux agents mobiles de disposer d'information dans une syntaxe commune quel que soit le système utilisé. Nous avons aussi montré que les MIB peuvent être étendues pour collecter de nouvelles données.

Suite aux expérimentations réalisées pour analyser la coopération entre deux IDS situés sur un même nœud, si les ressources du système sont limitées, le LIDS peut alors être configuré pour cibler les attaques les plus critiques pour le réseau et le nœud. Une configuration plus complète peut être obtenue par la mise en œuvre de plusieurs IDS spécialisés sur le même nœud.

Nous nous orientons vers une architecture d'IDS distribuée et coopérative dans laquelle la configuration du système de détection d'intrusions actif sur chaque nœud pourrait être adaptée aux ressources disponibles.

L'architecture d'IDS distribuée, le modèle de coopération par agents mobiles, l'architecture interne des LIDS et les résultats des expérimentations menées sur la plate-forme de tests ont été présentés dans la revue TSI [60].

Le chapitre suivant est une synthèse des résultats obtenus et présente des axes de travail pour améliorer les performances de l'IDS que nous proposons.

Chapitre 5

Conclusions et perspectives

Dans les MANET, les mécanismes de sécurité préventifs doivent être adaptés pour pallier l'absence d'infrastructure permanente. Devant ce nouveau challenge, nous avons proposé un modèle de sécurité qui associe les actions des mécanismes de sécurité préventifs et celles d'un système de détection d'intrusions (IDS).

Dans cette thèse, nous nous sommes intéressés au problème spécifique de la détection des intrusions dans les réseaux sans fil ad hoc.

Nous avons commencé par établir les caractéristiques d'un IDS répondant aux contraintes imposées par les MANET. Puis nous avons étudié, parmi les IDS existants, ceux compatibles avec les caractéristiques établies. A l'issue de cette analyse, nous avons constaté qu'il n'existait pas d'IDS possédant l'ensemble des caractéristiques recherchées.
Nous avons alors proposé notre propre modèle d'IDS distribué et coopératif. Dans ce modèle, chaque nœud est équipé d'un IDS, appelé LIDS, associé à un système de coopération basé sur des agents mobiles. Chaque LIDS exploite les données collectées dans les MIB SNMP situées sur les différents nœuds. L'architecture d'IDS distribuée, le modèle de coopération par agents mobiles, et l'architecture interne des LIDS ont été présentés dans cinq communications [2],[71],[72],[59] et [13]. Des résultats de mesures ont, en outre, été présentés dans la revue TSI [60].

La validation de notre modèle a été réalisée en deux étapes. Tout d'abord, après avoir défini l'architecture globale du système, nous nous sommes intéressés au sous-système de coopération et ensuite à la détection des intrusions. Nous avons particulièrement étudié la fiabilité du sous-système de coopération. Nous avons montré, par des mesures réalisées sur notre plate-forme de

tests, que dans un MANET, le protocole TCP ne permet pas d'assurer seul une coopération fiable entre les LIDS. Nous avons ensuite montré par des simulations que des agents mobiles se déplaçant de nœud en nœud sont plus fiables qu'une connexion client/serveur établie de bout en bout et que l'autonomie des agents mobiles permet de décharger les LIDS de tout contrôle de la coopération.
Les résultats des simulations que nous avons réalisées montrent que :

- les agents mobiles apportent plus de fiabilité que les connexions client/serveur établies de bout en bout,
- le taux de réussite d'un transfert avec un agent est toujours supérieur ou égal à celui d'une connexion client/serveur.

Pour les simulations réalisées avec les agents mobiles, nous avons été amenés faire évoluer le modèle du protocole TCP du simulateur NS-2 de façon à obtenir des phases d'ouverture, de transfert et de fermeture conformes aux spécifications du protocole. Ce modèle du protocole TCP est maintenant disponible pour la communauté d'utilisateurs du simulateur NS-2.

Dans la seconde partie de nos recherches, nous nous sommes attachés à valider l'aspect fonctionnel de la détection d'intrusions. Nous avons réalisé un prototype conforme au modèle présenté dans cette thèse. Les expérimentations réalisées dans un environnement de tests et un nombre limité de signatures ont montré que la détection basée sur la recherche de signatures d'attaques à partir des données collectées dans des MIB SNMP est fonctionnelle.
Pour une plus grande fiabilité, nous avons réalisé uniquement des requêtes SNMP locales. Les requêtes des variables situées sur des nœuds distants sont acheminées de façon fiable par le système de coopération afin de pallier le manque de fiabilité du protocole UDP associé à l'instabilité des connexions sans fil. L'utilisation des MIB permet de disposer de données avec une syntaxe commune sur tous les nœuds et simplifie ainsi la coopération entre les nœuds.

Le système de coopération, basé sur des agents mobiles, a pour objectifs :

- de limiter le volume de trafic échangé sur le réseau,
- de décharger l'application LIDS du contrôle de la coopération,
- de s'adapter à la stabilité des liens pour réaliser des échanges fiables.

La plate-forme Aglet utilisée avec le prototype nous a permis de valider les aspects fonctionnels de la coopération entre les LIDS. La capacité d'un agent mobile à analyser les données sur place sans les déplacer et à poursuivre un itinéraire de façon autonome peuvent contribuer à réduire le trafic réseau.

Nous avons constaté que les caractéristiques de déplacement d'un agent de la plate-forme Aglet n'étaient pas adaptées aux performances des réseaux sans fil.

Nous pensons pouvoir améliorer les performances de notre modèle d'IDS pour réseaux sans fil ad hoc en suivant trois axes de travail :

- *l'étude d'une plate forme à agents mobiles pour les MANET*: les services de cette plate-forme pourraient aussi être accessibles à d'autres applications comme le protocole de routage ou la découverte de la topologie ;

- *l'optimisation de l'algorithme de détection*: nous validons dans notre prototype une détection basée sur la recherche de signatures d'attaques. Toutefois, une détection de type comportementale peut aussi être envisagée. Sur la base de notre travail, des tests comparatifs permettraient alors de retenir la méthode de détection la plus performante dans le contexte des MANET ;

- *l'enrichissement de la base de signatures d'attaques spécifiques aux MANET*: nous pensons aussi réaliser une étude sur la corrélation entre les types d'attaques et les variations des variables de la MIB standard. Une approche similaire a déjà été réalisée dans [12] pour une attaque de type DDOS.

Bibliographie

[1] K. Al Agha, G. Pujolle, and G. Vivier. *Réseaux de mobiles et réseaux sans fil.* EYROLLES, 2001.

[2] P. Albers, O. Camp, J-M Percher, B. Jouga, L. Mé, and R. Puttini. Security in ad hoc networks: a general intrusion detection architecture enhancing trust based approaches. In *Proceedings of the international workshop on Wireless Information Systems-WIS2002,Ciudad Real,Spain*, April 2002.

[3] J.P Anderson. Computer security threat monitoring and surveillance. Technical report, James P Anderson Co., Fort Washington, Pennsylvania, April 1980.

[4] M. Asaka, Atsushi Taguchi, and Shigeki Goto. Ida-the implementation of ida : An intrusion detection agent system. Technical report, IPA Waseda University, 1999.

[5] B. S Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan. Improving performance of tcp over wireless networks. In *Proceedings of the 17th International Conference of Distributed Computing Systems*, pages 365–373, Mai 1997.

[6] J. Sundar Balasubraniyan, J. Omar Garcia-Fernandez, D.Isacoff, E. Spafford, and D. Zamboni. Aafid - autonomous agents for intrusion detection. Technical report 98/05, COAST Laboratory Purdue University, June 1998.

[7] W. Binder and V. Roth. Secure mobile agent systems using java: Where are we heading? In *Proceedings ACM Symposium on Applied Computing-SAC 2002, Madrid, Spain*, March 2002.

[8] E. Blanton, M. Allman, K. Fall, and L. Wang. A conservative selective acknowledgment (sack)-based loss recovery algorithm for tcp -proposed standard. ftp://ftp.rfc-editor.org/in-notes/rfc3517.txt.

[9] L. Blunk and J. Vollbrecht. Ppp extensible authentication protocol (eap). ftp://ftp.rfc-editor.org/in-notes/rfc2284.txt.

[10] N. Boukhatem. Les agents mobiles et applications. *Actes du colloque DNAC'99-De Nouvelles Achitectures pour les Communications, Paris, France*, 1999.

[11] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97. ACM Press, 1998.

[12] J. B. D. Cabrera, L. Lewis, X. Qin, W. Lee, R. Prasanth, B. Ravichandran, and R. Mehra. Proactive dectection of distributed denial of service attacks using mib traffic variables-a feasibility study. *Proceedings of The Seventh IFIP/IEEE International Symposium on Integrated Network Management-IM2001, Seattle, WA*, May 2001.

[13] E. Carmès, J-M Guérin, C.Devic, P. Nguyen, J-M. Percher, O.Camp, B.Jouga, and R. Puttini. Rahms : Réseaux ad hoc multiservices sécurisés. *Actes DNAC 2002-De Nouvelles Architectures pour les Communications, Paris*, Décembre 2002.

[14] T.-W. Chen and M. Gerla. Global state routing: A new routing scheme for ad-hoc wireless networks. In *Proceedings of IEEE International Conference on Communications- ICC-IEEE*, pages 171–175, June 1998.

[15] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik. Itinerant agents for mobile computing. *IEEE Personal Communications*, 2(5):34–49, 1995.

[16] C. C. Chiang, H.K. Wu, W. Liu, and M. Gerla. Routing in clustered multihop, mobile wireless networks. In *Proceedings of the IEEE Singapore International Conference on Networks*, pages 197–211,, 1997.

[17] T. Clausen and P. Jacquet. Optimized link state routing protocol. http.ietf.org/internet-drafts/draft-ietf-manet-olsr-11.txt, July 2003.

[18] Thomas Clausen. Rr-5135 - comparative study of routing protocols for mobile ad-hoc networks. Rapport de recherche 5135, INRIA- Institut National de Recherche en Informatique et Automatique, 2004.

[19] W. R. Cockayne and M. Zyda. *Mobile Agents*. MANNING, 1998.

[20] S. Corson and J. Macker. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation consideration. Request for Comments (Informational) 2501, IETF, 1999.

[21] J. Duarte de Queiroz, L. Fernando Rust da Costa Carmo, and L. Pirmez. Micael: An autonomous mobile agent system to protect new generation networked applications. In *Proceedings of RAID'98-Research Advances in Intrusion Detection*, June 1998.

[22] H. Debar, M. Dacier, and A. Wespi. A revised taxonomy for intrusion detection systems. Technical Report rz 3176, IBM Zurich Research Laboratory, October 1999.

[23] D.E. Denning and P.G. Neumann. Requirements and model for ides a real-time intrusion-detection expert system. Technical report : Document a005, Computer Science Laboratory, SRI International, Menlo Park, California, 1985.

[24] Dorothy E. Denning. An intrusion detection model. *IEEE Transactions on software engineering*, SE-13(NO.2):222–232, 1987.

[25] D. Hagimont et L. Ismail. *Agents mobiles et client/serveur : évaluation de performance et comparaison*, volume 19 of *9*. Technique et science informatiques, 2000.

[26] P. Muller et N. Gaether. *Modelisation objet avec UML*. Eyrolles, 2001.

[27] Laura Marie Feeney. A taxonomy for routing protocols in mobile ad hoc networks. Technical Report T99-07, 1, 1999.

[28] S. Floyd and T. Henderson. The newreno modification to tcp's fast recovery algorithm -experimental. ftp://ftp.rfc-editor.org/in-notes/rfc2582.txt.

[29] Y.F. Fou, F. Gong, C. Sargor, X. Wu, S. F. Wu, H. C. Chang, and F. Wang. Jinao-design and implementation of a scalable intrusion detection system for the ospf routing protocol. Technical report, Advanced Networking Research, MCNC Computer Science Dept, NC State University, February 1999.

[30] Matthew S. Gast. *802.11 Wireless Networks - The definitive Guide*. O'REILLY, 2002.

[31] Z. Haas. A new routing protocol for the reconfigurable wireless networks. In *Proceedings of the IEEE International Conference on Universal Personal Communications*, 1997.

[32] D. Harrington, R. Presuhn, and B. Wijnen. An architecture for describing simple network management protocol (snmp) management frameworks. ftp://ftp.rfc-editor.org/in-notes/rfc4311.txt.

[33] C. G. Harrison, D. M. Chess, and A. Kershenbaum. Mobile agents: Are they a good idea? Research report, IBM Research Division, T. J Watson Research Center, Yorktown Heights, NY 10598, March 1995.

[34] R. Hekmat and P. Van Mieghem. Degree distribution and hopcount in wireless ad-hoc networks. In *Proceedings of the 11th IEEE International Conference on Networks (ICON 2003)-Sydney, Australia*, pages 603–609, Sept 2003.

[35] G. Holland and N. Vaidaya. Analysis of tcp performance over mobile ad hoc networks. In *MobiCom 99- Proceedings of the Fifth Annual*

ACM/IEEE International Conference on Mobile Computing and Network, Seattle, Washigton, August 1999.

[36] Y.C. Hu, D.B. Johnson, and A.Perrig. Secure efficient distance vector routing in mobile wireless ad hoc networks. In *Proceedings of the 4th IEEE Workshop on Mobile Systems and applications - WMCSA*, June 2002.

[37] Y. Huang and W. Lee. A cooperative intrusion detection system for ad hoc networks. In *Proceedings of 1st ACM Workshop on Security of Ad Hoc and Sensor Networks, Fairfax, VA, USA*, October 2003.

[38] J. Liu J. and S. Singh. Atcp: Tcp for mobile ad hoc networks. *IEEE journal on Selected Areas in Communications*, (J-AC)(19 (7)):p. 1300–1315, July 2001.

[39] Ph. Jacquet, P. Muhlethaler, and A. Qayyum. Optimized link state routing protocol, draft-ietf-manet-olsr-00.txt. Internet Draft, IETF MANET Working Group, November 1998.

[40] D. B Johnson and D. A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[41] O. Kachirski and R. Guha. Intrusion detection using mobiles agents in wireless ad hoc networks. In *Proceedings of the IEEE Workshop on Knowledge Media Networkig (KMN'02)*, July 2002.

[42] O. Kachirski and R. Guha. Effective intrusion detection using multiple sensors in wireless ad hoc networks. In *Proceedings of HICSS'03 -36th Annual Hawaii International Conference on System Sciences*, January 2003.

[43] G. Karjoth, D. B. Lange, and M. Oshima. A security model for aglets. *IEEE Internet Computing*, July 1997.

[44] T. Karygiannis and L. Owens. Wireless network security. NIST Special Publication 800-48, National Institute of Standards and Technology-NIST - Technology Administration U.S Departement of Commerce, November 2002.

[45] C. Ko, M. Ruschitzka, and K. Levitt. Execution monitoring of security-critical programs in distributed systems: A specification-based approach. In *Proceedings of IEEE Symposium on Security and Privacy*, 1997.

[46] J. Koziol. *Snort 2.* CampusPress, 2003.

[47] C. Krugel and T.Toth. Flexible, mobile agent based intrusion detection for dynamic networks. In *Proceedings of European Wireless, Florence, Italy*, 2002.

[48] D. B. Lange and M. Oshima. *Programming and Deploying Java Mobile Agnets with Aglets.* ADDISON-WESLEY, 1998.

[49] l'IDWG Intrusion Detection Working Group de l'IETF. http://www.ietf.org/html.charters/idwg-charter.html.

[50] Stefano Martino. A mobile agent approach to intrusion detection. Technical report, Joint Research Center Institute for Systems, Informatics and Safety, Italy, June 1999.

[51] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. Masif: The omg mobile agent system interoperability facility. In *Proceedings of Mobile Agents, pages 50–67, Stuttgart, Germany*, Sept. 1998.

[52] P. Muhlethaler. *802.11 et les réseaux sans fil.* EYROLLES, 2002.

[53] S. Murthy and J. J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *Mob. Netw. Appl.*, 1(2):183–197, 1996.

[54] M.Wood and M.Erlinger. Intrusion detection message exchange requirements. *http://www.ietf.org/internet-drafts/draft-ietf-idwg-requirements-10.txt*, 2002.

[55] IEEE Institute of Electrical and Electronics Engineers. http://standards.ieee.org/getieee802/802.11.html.

[56] International Standards Organization. Information processing systems - osi reference model - part 2 : Security architecture. Technical Report 7498-2, 1989.

[57] G. Pei, M. Gerla, and T.-W. Chen. Fisheye state routing: A routing scheme for ad hoc wireless networks. In *Proceedings of IEEE International Conference on Communications- ICC-IEEE*, pages 70–74, 2000.

[58] G. Pei, M. Gerla, X. Hong, and C.C Chiang. A wireless hierarchical routing protocol with group mobility. In *Proceedings of the IEEE Wireless Communications and Networking Conference-WCNC*, September 1999.

[59] J-M. Percher and B.Jouga. Détection d'intrusions dans les réseaux ad hoc. *Actes du Symposium sur la Sécurité des Technologies de l'Information et des Communications-SSTIC2003, Rennes, France*, Juin 2003.

[60] J-M. Percher, R. Puttini, L. Mé, O. Camp, B. Jouga, and P. Albers. Un système de détection d'intrusions distribué pour réseaux ad hoc. *Sécurité Informatique TSI,vol 23,N° 3/2004*, 2004.

[61] C. Perkins. Ad-hoc on-demand distance vector routing. In *Proceedings IEEE MILCOM '97, Monterey, California*, October 1997.

[62] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on demand distance vector (aodv) routing. Request for Comments (Experimental) 3561, http://www.ietf.org/rfc/rfc3561.txt, July 2003.

[63] Ch. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications-ACM/SIGCOMM'94*, pages 234–244, 1994.

[64] V. Pham and A. Karmouch. Mobile software agents: An overview. *IEEE Communications Magazine*, pages 26–38, July 1998.

[65] J. Postel. Internet protocol. ftp://ftp.rfc-editor.org/in-notes/rfc791.txt.

[66] J. Postel. Transmission control protocol. ftp://ftp.rfc-editor.org/in-notes/rfc793.txt.

[67] P.Papadimitratos and Z.J. Haas. Secure routing for mobile ad hoc networks. In *Proceedings of the Communication Networks and Distributed Systems Modeleing and Simulation - CNDS*, January 2002.

[68] R. Presuhn and Ed. Management information base (mib) for the simple network management protocol (snmp). ftp://ftp.rfc-editor.org/in-notes/rfc4318.txt.

[69] N. Prigent, C. Bidan, J.P Andreaux, and O. Heen. Secure long term communities in ad hoc network. In *Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks-Fairfax, VA, USA*, October 2003.

[70] R. Puttini, L. Mé, and R. de Sousa. Manet authentication extension for securing manet routing protocols. In *Proceedings of the 5th IEEE International Conference on Mobile and Wireless Communications Networks (MWCN)*, October 2003.

[71] R. Puttini, J-M Percher, L. Mé, and R. de Sousa. A fully distributed ids for manet. In *Proceedings of the Ninth IEEE Symposium on Computers and Communications-ISSC2004, Alexandria, Egypte*, June 2004.

[72] R.S. Puttini, J-M. Percher, L.Mé, O.Camp, R. de Sousa Jr., C.J Barenco Abbas, and L.J. Garcia Villalba. A modular architecture for distributed ids in manet. In *Proceedings of the International Conference on Computational Science and Its Applications-ICCSA2003, Montreal, Canada*, May 2003.

[73] G. Ratman and S.Singh. Wtcp: an efficient mechanism for improving tcp performance over wireless links. In *Proceedings of the 3rd IEEE Symposium on Computers and communications*, pages 74–78, June 1998.

[74] E. Royer and C. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, 1999.

[75] Kunal Shah. *Performance analysis of mobile agents in wireless internet applications using simulation.* PhD thesis, The Faculty of the College of Graduate Studies, Lamar University, August 2003.

[76] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. Todd Heberlein, Che-Lin Ho, K. N. Levitt, B. Mukkherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur. Dids-distributed intrusion detection system. Technical report, Computer Security Laboratory, Department of Computer Science, University of California, Davis, June 1992.

[77] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. Grids- a graph based intrusion detection system for large networks. Technical report, Computer Security Laboratory, Department of Computer Science, University of California, Davis, 1996.

[78] S. Staniford-Chen and Todd Heberlien. Holding intruders accountable on the internet. In *Proceedings of the IEEE symposium on security and privacy, p 39-49, Oakland*, May 1995.

[79] W. W. Streilein, D. J. Frie, and R. K. Cunningham. Detectin flood-based denial of service attacks with snmp/rmon. *MIT Lincoln Laboratory*, 2002.

[80] G. B. White, E. A. Fish, and Udo Pooch. Csm - cooperating security managers: a peer based intrusion detection system. *IEEE Networks*, pages 20–23, January 1996.

[81] Y. Zhang and W. Lee. Intrusion detection in wireless ad hoc networks. In *Proceedings of the sixth annual international conference on Mobile computing and networking, MOBICOM'2000.* pages 275–283. ACM Press New York, USA, 2000.

[82] Y. Zhang, W. Lee, and Y. Huang. Intrusion detection techniques for mobile wireless networks. *ACM/Kluwer Wireless Networks Journal*, Vol. 9(No. 5):545–556, September 2003.

[83] L. Zhou and Z. Haas. Securing ad hoc networks. *IEEE Network Magazine*, 13(6), November-December 1999.

Annexes

Annexe A

Les normes IEEE 802.11

A.1 Principales caractéristiques des normes IEEE 802.11

Les différentes versions de la norme IEEE 802.11 se différencient principalement par leur débit et la bande de fréquences utilisée. Nous présentons dans la figure A.1 les caractéristiques des normes IEEE 802.11, IEEE 802.11a, IEEE 802.11b et IEEE 802.11g.

MAC IEEE 802.11			
IEEE 802.11 2,4 GHz 2 Mps	IEEE 802.11a 5 GHz 54 Mps	IEEE 802.11b 2,4 GHz 11 Mps	IEEE 802.11g 2,4 GHz 54 Mps

FIG. A.1 – *Caractéristiques des principales normes IEEE 802.11*

A.2 Les trames IEEE 802.11 du mode ad hoc

Il existe trois types de trames IEEE 802.11 :

– *Les trames de données*, utilisées pour la transmission des données utilisateur ;

- *Les trames de contrôle*, utilisées pour contrôler l'accès au support (RTS, CTS, ACK,...) ;
- *Les trames de gestion*, utilisées les opérations d'association, de synchronisation et d'authentification.

Toutes ces trames sont construites suivant le modèle représenté sur la figure A.2.

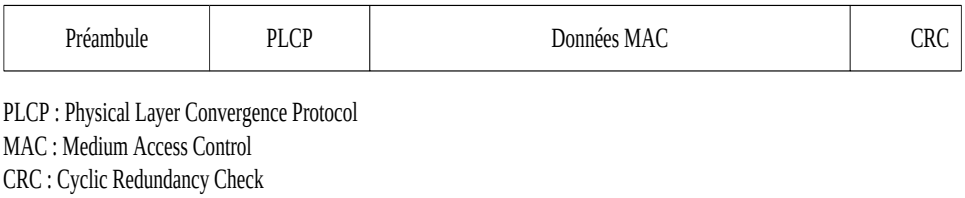


FIG. A.2 – *Format d'une trame IEEE 802.11*

- Le préambule est composé d'un champ synchronisation formé d'une séquence de 80 bits alternant 0 et 1 et d'un champ d'identification de début de trame, SFD (*Start Frame Delimiter*), de 16 bits.
- L'en-tête PLCP contient des informations utilisées par la couche physique pour décoder la trame.
- Le champ données MAC est représenté sur la figure A.3. Nous donnons

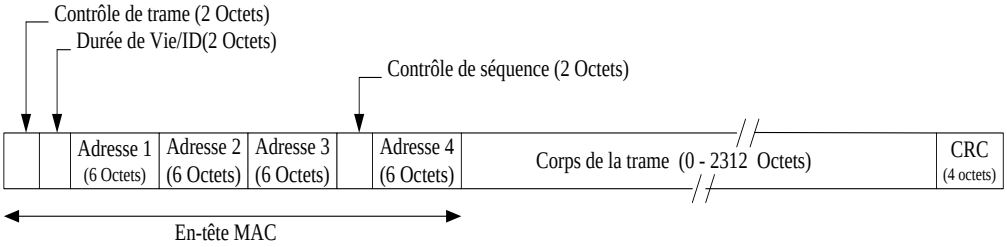


FIG. A.3 – *Format de la partie MAC d'une trame IEEE 802.11*

dans cette section la fonction des différents champs en mode ad hoc uniquement. Certains champs peuvent ne pas être présents selon les types de trames.

- Le champ de *contrôle de trame*, représenté sur la figure A.4 contient les informations suivantes :
  - Le champ *version du protocole* est positionné à 00 pour les versions actuelles de la norme.

- Les champs *Types et Sous-types* indiquent la nature de la trame. En mode ad hoc les trames suivantes sont utilisées :

  Trames de gestion : Requête/réponse d'association, Balise (*Beacon*), ATIM (*Ad hoc Traffic Information Map*).

  Trames de contrôle : RTS/CTS, ACK.

  Trames de données utilisées pour la transmission des données.
- Les champs *ToDS et FromDS* sont positionnés à zéro en mode ad hoc pour indiquer que les échanges de trames se font entre deux stations.
- Le champs *More Fragments* est positionné à un pour indiquer que d'autres fragments suivent le fragment en cours.
- Le champs *Retry* à 1 indique que le fragment est une retransmission d'un fragment précédemment transmis.
- Le champs *Power Management* indique que la station passera en mode d'économie d'énergie après la transmission de cette trame.
- Le champs *More Data* est utilisé en mode d'économie d'énergie et indique à la station que d'autres données, en attente, lui sont destinées.
- Le champs *WEP* indique que le corps de la trame est chiffré suivant l'algorithme WEP (*Wire Equivalent Privacy*).
- Le champs *Order* indique que cette trame est envoyée en utilisant la classe de service strictement ordonné.

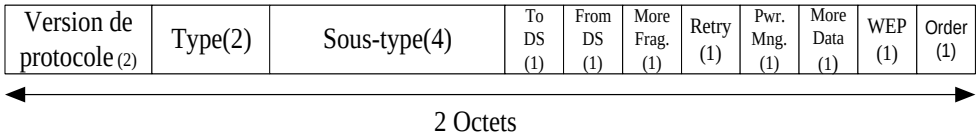


FIG. A.4 – *Format du champ contrôle de l'en-tête MAC 802.11*

- Le champ *Durée/ID*, selon le type de la trame, indique soit l'identifiant de la station dans les trames de polling en mode d'économie d'énergie, soit la valeur utilisée pour le calcul du NAV dans les autres trames.
- Les *Champs Adresses* dépendent de la valeur des bits *ToDS et FromDS* de la trame. En mode ad hoc l'utilisation de ces champs

est présentée dans le tableau A.1. Ces champs indiquent l'identifiant du BSS (*BSSID*), les adresses sources (*SA*)et destination (*DA*). Le champ *Adresse 4* est inutilisé en mode ad hoc.

ToDS	FromDS	Adresse 1	Adresse 2	Adresse 3	Adresse 4
0	0	DA	SA	BSSID	Impossile

TAB. A.1 – *Les champs adresses de la trame MAC - IEEE 802.11*

– Le champ *contrôle de séquence* spécifie l'ordre des fragments d'une trame fragmentée. Il est constitué du numéro de fragment dans la trame et du numéro de séquence qui le numéro de la trame.

Annexe B

Principaux protocoles de routages ad hoc

Les protocoles de routage ad hoc se différencient entre eux par le niveau d'implication des nœuds dans le routage. Ils peuvent ainsi être uniformes si tous les nœuds du réseau jouent le même rôle dans le routage et que cette fonction de routage est distribuée de façon égaliataire. Ils peuvent à l'inverse être non-uniformes si une structure hiérarchique est donnée au réseau et que seuls certains nœuds assurent le routage. Ainsi, les protocoles à sélection de voisins s'appuient sur routage réalisé par leurs voisins pour transmettre leur données à travers le réseau, tandis que les protocoles à partitionnement découpent le réseau en zones dans lesquelles le routage sera assuré par un unique nœud maître. Les protocoles de routage uniformes peuvent également être regroupés selon les données qu'ils utilisent pour effectuer leur tâche. Pour les protocoles orientés topologie, plus connus sous le nom de *link-state protocols*, chaque nœud se base sur l'état des connexions qu'il peut avoir avec d'autres nœuds. Cette information est ensuite transmise aux autres nœuds du réseau pour leur donner une connaissance plus précise de la topologie du réseau. Les protocoles orientés destinations, plus connus sous le nom de *distance vector protocols*, maintiennent pour chaque nœud destination une information sur le nombre de nœuds qui les en séparent (la distance) et éventuellement sur la première direction à emprunter pour y arriver (le vecteur).

AODV (*Ad hoc On demand Distance Vector*) [61] est un protocole réactif, uniforme et orienté destination. La route retenue est bidirectionnelle et correspond au plus court chemin (en nombre de nœuds) entre la source et la destination. Chaque nœud maintient une table de routage dont les entrées mémorisent, pour une destination :

- l'identifiant de cette destination,

- l'identifiant du prochain nœuds vers cette destination,
- le nombre de nœuds jusqu'à cette destination.

La demande de route est diffusée par la source à travers tout le réseau. Celle-ci permet à tous les nœuds de mémoriser une route vers la source. Quand la destination reçoit cette demande, la transmission de sa réponse permet aux nœuds intermédiaires de mémoriser la route recherchée. Pour maintenir les routes, chaque nœud vérifie périodiquement la présence de ses voisins. De plus, un numéro de séquence date chaque requête ou réponse pour éviter les multiples traitements et actualiser les routes.

TORA (*Temporally Ordered Routing Algorithm*)[11] est un protocole réactif, uniforme et orienté destination. Le choix de la route est réalisé à partir de la construction d'un graphe acyclique direct entre la source et la destination. Ce graphe représente l'ensemble des routes possibles vers une destination. Pour déterminer une route, la source diffuse vers tous les nœuds une demande de route. Quand celle-ci est reçue par la destination, la réponse permet d'attribuer un poids à tous les nœuds impliqués. Ce poids représente la distance minimale entre le nœud et la destination et permet de construire le graphe des chemins de la source vers la destination. Ensuite, la route utilisée est déterminée à chaque nœud par la sélection du nœud voisin qui possède le poids le plus faible. Un mécanisme permet d'actualiser le graphe dès qu'une modification apparaît dans le réseau (suppression d'un lien). De plus, il existe un mécanisme de destruction de route utilisé lorsque cette dernière n'est plus employée.

DSDV (*Destination Sequence-Distance Vector*)[63] est un protocole proactif, uniforme, orienté destination et basé sur l'algorithme de *Bellman-Ford* distribué. Chaque nœud possède en permanence dans sa table de routage le plus récent chemin découvert vers chaque nœud du réseau. Parmi plusieurs chemins de même âge, DSDV privilégie le chemin le plus court. Chaque nœud maintient une table de routage dont les entrées mémorisent pour une destination :

- l'identifiant du prochain nœud vers cette destination,
- le nombre de nœud jusqu'à cette destination (la distance),
- le plus grand numéro de séquence reçu pour cette destination.

Chaque nœud diffuse périodiquement sa table de routage à travers le réseau. Lors d'une nouvelle diffusion, le nœud incrémente un numéro de séquence et le transmet avec sa table de routage. Celui-ci est utilisé par les autres nœuds pour valider la mise à jour de leur table de routage et éviter les boucles. Pour limiter les informations échangées, des mises à

jours incrémentales sont générées pour informer les nœuds des derniers changements intervenus (nouvelles destinations- routes perdues) depuis la dernière mise à jour complète. Cette dernière est générée périodiquement par chaque nœud.

WRP (*Wireless Routing Protocol*)[53] est un protocole proactif, uniforme, et orienté destination. Ce protocole reprend les mêmes principes de fonctionnement que le protocole AODV, mais utilise quatre tables pour gérer le routage :

- la table des distances qui contient une entrée par voisin pour chaque nœud destination du réseau. Chaque entrée mémorise le nombre de sauts ainsi que le nœud situé avant le nœud destination.
- la table de routage qui contient une entrée pour chaque destination, mémorise pour chaque entrée, la distance, le prochain nœud et le nœud précédent.
- la table de coût des liens qui recense le coût des liens avec tous les voisins du nœud, ainsi que temps écoulé depuis le dernier message correct quand un nœud voisin ne répond plus.
- la table des retransmissions de messages qui est utilisée pour vérifier que les voisins ont bien acquitté les messages de mise à jour.

Par la diffusion des tables de routage, chaque nœud possède toutes les informations de routage vers toutes les destinations du réseau. Pour suivre les évolutions de la topologie du réseau, chaque nœud échange avec ses voisins des paquets de mise à jour. L'information "nœud précédent" permet d'éviter les boucles et accélère ainsi la convergence.

DSR (*Dynamic Source Routing*)[40] est un protocole réactif, uniforme et orienté topologie. Ce protocole reprend le principe de routage d'IPv6. La route retenue correspond au plus court chemin, en nombre de nœuds, entre la source et la destination. La demande de route diffusée sur le réseau mémorise les nœuds qu'elle traverse. Lorsqu'une demande atteint la destination, elle contient la liste complète des nœuds traversés. La destination peut ainsi choisir la route optimale et la retransmettre à la source en empruntant le chemin inverse. Quatre types de messages de routage sont utilisés : *demande de route*, *réponse à une demande*, *erreur* et *information*. Le cumul des options de demande et de réponse permet, par exemple, à une destination d'initier une demande de route vers le nœud demandeur tout en lui transmettant la réponse. Ce protocole exploite ainsi les liens unidirectionnels du réseau. Une fois la route établie, les paquets contiennent dans leurs entêtes la liste complète des nœuds à traverser. Les nœuds intermédiaires n'ont pas besoin de maintenir d'information sur le routage. Cependant, les demandes de

routes qui traversent le réseau actualisent les informations de routage stockées dans les nœuds traversés afin d'optimiser les demandes de découverte de routes. Lorsque l'impossibilité de joindre un nœud d'une route est détectée, un message d'erreurs est généré afin de mettre à jours les informations stockées dans les caches. Pour des raisons d'économie d'énergie, les nœuds peuvent se mettre en veille. Ils signalent leur réapparition sur le réseau en diffusant un message d'information pour rétablir les liens avec leurs voisins. Certaines optimisations apportent une composante proactive à cet algorithme.

GSR (*Global State Routing*)[14] est un protocole uniforme, proactif, et orienté topologie qui s'inspire de DSDV. Dans chaque nœud, les informations sur la topologie complète du réseau (l'état des liens, le nombre de sauts vers chaque destination, la liste des nœuds voisins, le prochain saut..) sont mémorisées dans des tables. L'ensemble de ces informations sont propagées uniquement aux nœuds voisins à intervalles réguliers, et chaque fois qu'un changement de topologie est détecté. Il n'y a pas de mise à jour par diffusion à travers le réseau.

FSR (*Fisheye State Routing*)[57] est un protocole uniforme, proactif, et orienté topologie. Ce protocole fonctionne sur le même principe que le protocole GSR, mais les messages de mise à jour ne sont pas adressés à l'ensemble des nœuds du réseau. La fréquence de rafraîchissement est aussi fonction de la distance entre les nœuds. Même si un nœud n'a pas une vision précise de la topologie lointaine, les messages qu'il enverra seront routés correctement, la précision étant d'autant plus grande qu'ils s'approcheront de la destination.

CGSR (*Clusterhead Gateway Switch Routing Protocol*)[16] est un protocole proactif, non-uniforme et orienté partitionnement. Il subdivise le réseau en grappes, appelées *clusters*, et détermine un maître, le *clusterhead*, pour chacune d'elles. Le routage entre deux nœuds se fait via les maîtres des différentes grappes à traverser. CGSR définit également des passerelles dont le rôle est de transférer les paquets du maître d'une grappe vers celui de sa voisine. Une grappe est définie par la zone d'émission/réception de son maître. Le routage entre les différents maîtres se fait à l'aide de l'algorithme DSDV. Tous les nœuds tiennent à jour une table d'appartenance aux grappes, appelée *Cluster Member Table*. L'algorithme DSDV permet de maintenir cette table à jour pour l'ensemble des nœuds du réseau. Par ailleurs, les nœuds doivent aussi maintenir une table de routage. Celle-ci est utilisée par les maîtres pour se contacter. A chaque maître de grappe, la table de routage associe : la première passerelle à utiliser, appelée le *next hop* ainsi que le nombre

de sauts pour l'atteindre. Le déplacement des nœuds peut entraîner des désignations fréquentes des maîtres de grappes. L'algorithme LCC (*Least Cluster Change*) minimise le nombre de ces opérations. Avec cet algorithme, un nœud devient maître, et définit ainsi sa propre grappe lorsqu'il se retrouve hors de la zone d'émission/réception d'un autre maître. Par ailleurs, lorsque deux maîtres se rencontrent, l'algorithme LCC permet de n'en retenir qu'un seul.

HSR (*Hierarchical State Routing*)[58] est un protocole réactif, non-uniforme et orienté partitionnement. Le réseau est subdivisé en grappes et un maître est élu pour chacune d'elles. HSR s'appuie sur des niveaux hiérarchiques, formés par le regroupement de nœuds ou de maîtres, de façon à organiser la diffusion des informations. Chaque nœud diffuse les informations sur les liens qu'il entretient avec ses voisins aux autres nœuds du réseau. Les maîtres de grappes résument cette information et la transmettent aux maîtres voisins (via des nœuds passerelles). Les niveaux supérieurs procèdent de manière analogue et s'appuient en plus sur l'information résumée venant du niveau inférieur. Quand ces informations arrivent au niveau le plus haut, elles sont rediffusées vers les niveaux inférieurs. Le partitionnement physique permet d'associer une adresse hiérarchique, HID (*Hierarchical ID*), à chacun des nœuds. Cet identificateur suffit pour transmettre de l'information vers un autre nœud quelconque du réseau dont on connaît également le HID. En plus du regroupement physique effectué au travers des grappes (subdivision géographique), HSR propose également un partitionnement logique du réseau en groupes fonctionnels.

OLSR (*Optimized Link State Routing Protocol*)[39],[17] est un protocole proactif, non uniforme et basé sur la sélection de voisins. OLSR s'appuie sur le concept de Relais Multi Point , appelés MPR pour (*Multi Point Relay*). Les MPR d'un nœud correspondent à l'ensemble des voisins qui permettent d'atteindre tous les nœuds situés à deux sauts. La diffusion des différents messages de contrôle ne se fera que vers les MPR, réduisant ainsi les répétitions inutiles. OLSR tient compte et exploite les liens unidirectionnels du réseau. Chaque nœud maintient de l'information sur les nœuds qui l'ont élu en tant que MPR. Ceci est fait grâce à des messages de présence, appelés *hello messages*, envoyés périodiquement par chaque nœud à ses voisins. Ces messages contiennent :

- la liste de nœuds avec lesquels l'émetteur entretien des liens bi-directionnels,
- la liste de nœuds que l'émetteur peut entendre (ils entretiennent un lien uni-directionnel vers lui),

– la liste de nœuds que l'émetteur a choisi en tant que MPR.

La diffusion de ces messages de présence permet aux nœuds du réseau de stocker, dans leur table de voisins, une vision à deux sauts de leur voisinage et de calculer l'ensemble de leur MPR. Cet ensemble est recalculé dès qu'un changement est détecté dans le voisinage à deux sauts. La diffusion sur la totalité du réseau (via les MPR) de messages de contrôle de topologie, appelés TC pour *Topology Control messages*, par les MPR donne l'information topologique nécessaire au routage. Ils donnent, pour chaque MPR, la liste des nœuds qui l'ont choisi. Grâce à ces messages, les nœuds peuvent maintenir une table de topologie (*Topology Table*), indiquant le dernier saut pour chaque destination. Un algorithme de plus court chemin appliqué à la table des voisins et à la table de topologie permet de construire la table de routage de chaque nœud. Cette table mémorise, pour tous les nœuds du réseau, le nombre de sauts, et le premier saut pour l'atteindre. Elle doit être recalculée dès que l'une des deux tables sources est modifiée.

ZRP (*Zone Routing Protocol*)[31] est un protocole non uniforme, à sélection de voisins et hybride (actif et proactif). Pour identifier une route, ZRP s'appuie sur deux protocoles : IARP (*IntrAzone Routing Protocol*) et IERP (*IntErzone Routing Protocol*). ZRP définit pour chaque nœud une zone dont l'étendue est définie par le nombre maximal de sauts autour de ce nœud. Les noeuds situés à cette distance maximale du nœud source sont appelés les nœuds périphériques. IARP est défini comme un protocole de routage proactif, de type *link state* ou *distance vector*. Il permet à chaque nœud de connaître et de diffuser les routes vers chacun des nœuds de sa propre zone. Les demandes de routes au-delà de la zone sont transmises uniquement aux nœuds périphériques. Cette procédure se répète de zone en zone jusqu'à la localisation du nœud destination. Après avoir reçu la requête, celui-ci répond à la source en lui indiquant la route à suivre pour le joindre. IERP est un protocole de routage réactif chargé d'établir et de maintenir les routes au delà de la zone du nœud source.

Annexe C

Caractéristiques des fichiers XML

XML a été mis au point en 1996 par le XML Working Group sous l'égide du World Wide Web Consortium[1]. Depuis le 10 février 1998, les spécifications XML 1.0 ont été reconnues par le W3C.
XML est un langage qui permet de structurer les documents grâce à l'utilisation de balises *markup*, il peut être considéré comme un métalangage[2] qui, à ce titre, permet de créer à volonté de nouvelles balises pour isoler toutes les informations élémentaires.
En effet, sa puissance est due à sa capacité à pouvoir décrire n'importe quel domaine de données grâce à son extensibilité. Il va permettre de structurer le document, de définir le vocabulaire et la syntaxe des données qu'il va contenir.

Un document XML est structuré en 3 parties:

La première partie, identifie la version de la norme XML utilisée pour créer le document (mention obligatoire), ainsi que son jeu de caractères (*encoding* - attribut facultatif - ici on spécifie qu'il s'agit du jeu ISO-8859-1, jeu LATIN, pour permettre de prendre en compte les accents français). Ainsi, le prologue est une ligne du type:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Le prologue se poursuit avec des informations facultatives sur des instructions de traitement à destination d'applications particulières. Leur syntaxe est la suivante:

```
<?instruction de traitement?>
```

1. W3C - www.w3.org
2. Langage permettant de définir d'autres langages

La seconde partie, définit le type de document, à l'aide d'un fichier annexe appelé DTD - *Document Type Definition.*

La troisième partie, est l'arbre des éléments, dont la description suivante représente par exemple une signature simplifiée de l'attaque Steptone.

```
<FSDLIST>
<FSD CODE="STEPSTONE">
<PERSISTENT/>

<STATE NUMBER="0">
<PERIODICALQUERY CODE="TELNETNEWCONN" PERIOD="5000"/>

<NEXTSTATE NUMBER="1" EVENTCODE="NEWTELNETCONN"/>
</STATE>

<STATE NUMBER="1">
<QUERY CODE="TELNETREMOTECHAIN"/>
<NEXTSTATE NUMBER="2" EVENTCODE="NORMALCHAIN"/>
<NEXTSTATE NUMBER="3" EVENTCODE="ANORMALCHAIN"/>
</STATE>

<STATE NUMBER="2"/>


<STATE NUMBER="3">
<ALAsdfRM CODE="ANORMALTCPCONN"/>
<PERIODICAfsdLQUERY CODE="REMOTETCPCHAIN" PERIOD="8000"/>
<NEXTSTATE NUMBER="2" EVENTCODE="NORMALCHAIN"/>
<NEXTSTATE NUMBER="3" EVENTCODE="ANORMALCHAIN"/>
</STATE>

</FSD>

<DISABLEDFSD CODE="NHOP">
<PERIODICALccQUERY CODE="STATUS" PERIOD="2000"/>
<STATE NUMBER="0">
<NEXTSTATE NUMBER="1" EVENTCODE="ASYMSYM"/>
</STATE>

<STATE NUMBER="1">
```

```
<NEXTSTATE NUMBER="2" EVENT="SYMSYM"/>
<NEXTSTATE NUMBER="3" EVENT="ASYMSYM"/>
</STATE>

<STATE NUMBER="2">
<QUERY CODE="REMOTETCPCHAIN"/>
<NEXTSTATE NUMBER="2" EVENT="SYMSYM"/>
<NEXTSTATE NUMBER="3" EVENT="ASYMSYM"/>
</STATE>

</DISABLEDFSD>


</FSDLIST>
```

Annexe D

La plate-forme à agents mobiles: AGLET

Un Aglet (**Ag**ent App**let**) est un objet mobile, écrit en Java, qui peut se déplacer de machine en machine avec son code et ses données.

L'API Aglet a été développée en 1995 par une équipe de chercheurs du laboratoire de recherche d'IBM de Tokyo. Elle offre aux programmeurs les fonctionnalités nécessaires au développement d'applications basées sur des agents mobiles avec les avantages liés à Java. La figure D.1 représente les principaux objets et interfaces définis par l'API Aglet.

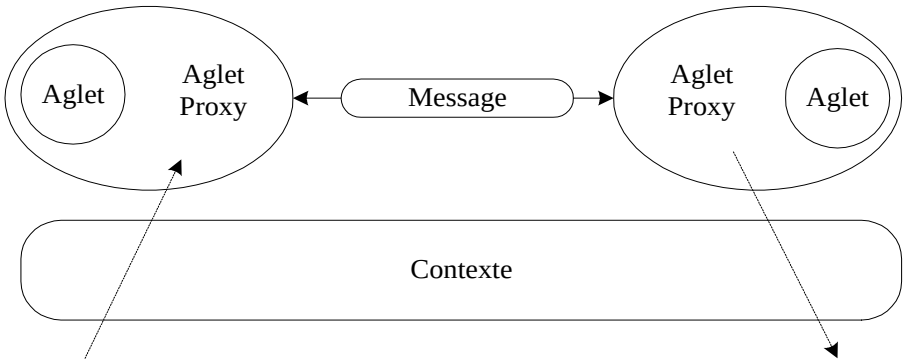


FIG. D.1 – *Le modèle objet de l'API Aglet*

- L'*aglet* est un objet mobile qui possède un *proxy* et qui se déplace de *contexte* en contexte.
- Le *contexte* est un objet statique qui représente l'environnement d'exécution d'un aglet sur un système à un instant donné. Il est aussi appelé

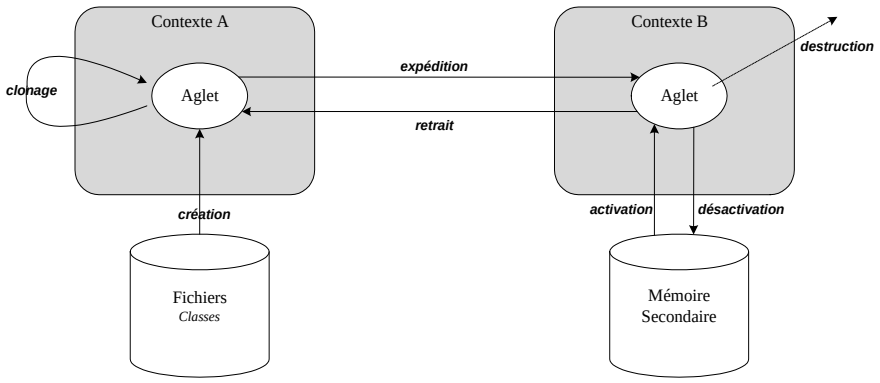


FIG. D.2 – *Le modèle du cycle de vie d'un Aglet*

une *place*. Le contexte permet d'une part de gérer et de contrôler les aglets, et d'autre part de protéger le système hôte des agents malveillants. Un même système peut aussi héberger plusieurs contextes.

- Le *proxy* est le représentant d'un aglet, il permet de masquer sa localisation réelle. Il reste dans le contexte initial de l'aglet à sa création et constitue l'intermédiaire obligatoire de tous les échanges avec son aglet.
- Le *message* est un objet échangé entre deux aglets. La communication par messages, synchrones et asynchrones, permet aux aglets de collaborer et d'échanger des informations.

Après sa création, l'état d'un aglet peut être modifié par l'exécution de différentes opérations. Ainsi un aglet peut, par exemple, être *cloné*, *déplacé* dans un autre contexte, ou *suspendu temporairement*.

La figure D.2, page 146, représente les différents états du cycle de vie d'un aglet. Les principales opérations relatives à la vie d'un aglet sont :

La création : la création d'un aglet s'effectue dans un contexte. Le nouvel aglet reçoit alors un identifiant et commence à s'exécuter dès la fin de son initialisation.

Le clonage : le clonage d'un aglet est la création d'une copie de l'aglet initial dans le même contexte. Un nouvel identifiant permet de différencier l'aglet initial de son l'aglet clone. Le clonage d'un aglet donne naissance à un nouveau processus (thread) d'exécution pour l'aglet clone.

L'expédition (*dispatching*) : l'expédition d'un aglet consiste à l'extraire de son contexte d'exécution pour l'insérer dans un contexte destination où son exécution sera relancée. La mobilité des aglets repose sur cette opération de transfert d'un contexte à un autre.

Le retrait (*retractation*) : le retrait d'un aglet consiste à l'extraire de son contexte d'exécution distant afin de l'insérer et de le relancer dans le contexte qui a invoqué l'opération.

La désactivation : la désactivation d'un aglet est une interruption temporaire de son exécution qui entraîne son retrait du contexte et son stockage provisoire dans un espace de stockage secondaire (i.e sur le disque)

L'activation : l'activation est la restauration de l'aglet depuis la mémoire secondaire dans son contexte initial pour une reprise de son exécution. Cette réactivation de l'aglet peut se produire à l'expiration d'un délai ou à l'apparition d'un événement externe.

La destruction (*disposal*): la destruction est la dernière étape du cycle de vie d'un aglet. C'est l'interruption de son exécution et la suppression de son contexte d'exécution courant.

Le cycle de vie d'un aglet est basé sur la gestion d'événements et la gestion de messages. Nous expliquons dans les sections suivantes respectivement le modèle de gestion des événements, le modèle de communication et le protocole de transfert ATP (*Agent Transfer Protocol*).

D.1 Le modèle événementiel de l'aglet

La programmation des aglets est basée sur la gestion d'événements. L'aglet utilise des écouteurs (*listeners*) pour capturer les événements et exécute ensuite les méthodes associées aux événements capturés. Trois types d'événements sont définis pour notifier le clonage (*CloneEvent*), la mobilité (*MobilityEvent*) et la persistance (*PersistencyEvent*).

Par exemple, quand la méthode **clone()** est invoquée pour cloner un aglet, l'aglet qui fait l'objet du clonage reçoit alors les notifications suivantes :

Public void CloneAdapter.onCloning(CloneEvent event) annonce de clonage à l'aglet, appel de la méthode *onCloning*. Le clonage est ensuite réalisé et notifié à l'aglet original et à l'aglet *clone*.

Public void CloneAdapter.onCloned(CloneEvent event) méthode appelée par l'aglet original dès la réalisation du clonage.

Public void CloneAdapter.onClone(CloneEvent event) méthode appelée par l'aglet *clone* dès la réalisation du clonage.

La figure D.3 présente l'invocation des différentes méthodes de l'aglet original et de l'aglet *clone* au cours du processus de clonage. Nous donnons figure D.4, page 148, un exemple de programmation d'aglet avec la gestion des événements de clonage.

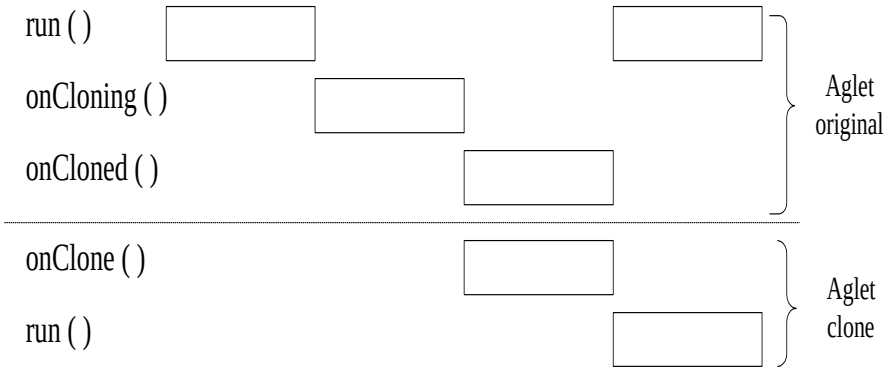


FIG. D.3 – *Diagramme de collaboration pour le clonage d'un Aglet*

```
public class Exemple_Clonage extends Aglet {
        boolean le_Clone = false;
        public void onCreation (Object o) {
                addCloneListener(
                        new CloneAdapter() {
                                public void onCloning(CloneEvent event) {
                                        // Exécuté juste avant le clonage
                                }
                                public void onClone(CloneEvent event) {
                                        // Exécuté après le clonage par l'aglet  clone
                                        le_Clone = true;
                                }
                                public void onCloned(CloneEvent event) {
                                        // Exécuté après le clonage par l'aglet original
                                }
                        }
                        );
                }

        public void run() {
                if (!le_clone) {
                // Exécution aglet original
                   try {
                     clone();
                   } catch (Exception e) {
                      System.out.println(e.getMessage());
                   }
                } else {
                //Exécution aglet clone
                }
        }
```

FIG. D.4 – *Gestion des événements de clonage d'un Aglet*

Les différents événements reçus par les méthodes des classes *CloneAdapter*, *MobilityAdapter* et *PersistyAdapter* sont présentés dans le tableau de synthèse D.1

Génération de l'événement	Événement	Écouteur	Méthode appelée
Avant le clonage	CloneEvent	CloneListener	onCloning
Par l'aglet clone,après le clonage	CloneEvent	CloneListener	onClone
Par l'aglet original, après le clonage	CloneEvent	CloneListener	onCloned
Avant le transfert	MobilityEvent	MobilityListener	onDispatching
Avant le retrait	MobilityEvent	MobilityListener	onReverting
Après l'arrivée à destination	MobilityEvent	MobilityListener	onArrival
Avant la désactivation	PersistencyEvent	PersistencyListener	onDeactivating
Après l'activation	PersistencyEvent	PersistencyListener	onActivating

TAB. D.1 – *Événements et méthodes de l'API Aglet*

D.2 Le modèle de communication des aglets

La communication entre les aglets, situés sur un même système ou des systèmes distants, est basée sur l'échange de messages. Le modèle de communication propose des échanges synchrones ou asynchrones indépendants de la localisation de l'aglet. La figure D.5 représente les deux principaux modèles d'échanges et les objets impliqués dans ces échanges. Un aglet adresse toujours son message au *proxy* de l'aglet destinataire. Ce *proxy*, accessible depuis le contexte initial de l'aglet, se charge de relayer les messages à son aglet quelque soit la localisation réelle de son aglet. Chaque aglet possède un gestionnaire de message (*handleMessage*) dont la fonction est de traiter les messages reçus selon leur catégorie et leur priorité. La figure D.5-A représente le modèle d'un échange synchrone. L'émetteur du message attend la réponse du destinataire pour poursuivre son exécution. Dans le modèle d'échange asynchrone, représenté sur la figure D.5-B, l'émetteur du message poursuit son exécution sans attendre la réponse du destinataire. Dans ce cas, l'émetteur dispose de méthodes pour tester l'arrivée de la réponse.

D.3 ATP le protocole de transfert de la plate-forme Aglet

La plate-forme Aglet peut se représenter par une architecture en 3 couches (Voir la figure D.7) : l'aglet, l'Aglet *runtime* et la couche de communication ATP (*Aglet Transfer Protocol*). Les deux dernières couches disposent d'interfaces (API) pour permettre l'accès à leurs fonctions. L'Aglet *runtime* a pour fonction de fournir les services nécessaires au management, à l'exécu-

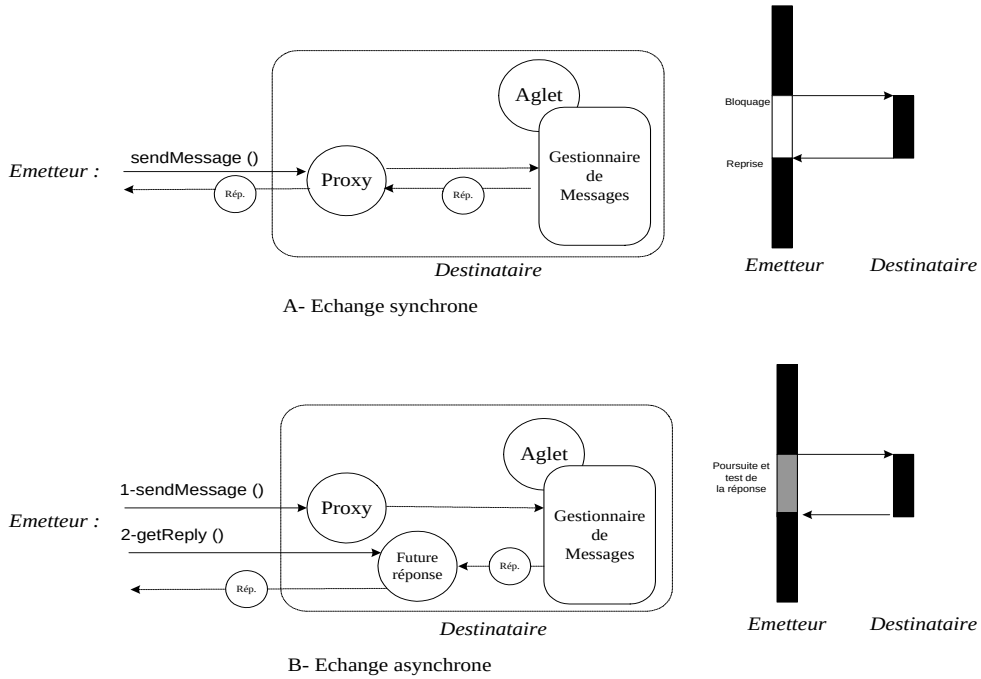


FIG. D.5 – *Communication par échange de messages*

tion et aux transferts des aglets. Il est formé de deux parties : le noyau et un ensemble extensible de composants comme, par exemple, le gestionnaire de sécurité (*SecurityManager*).

Les fonctions de bases (Sérialisation, Chargement des Classes,.), nécessaires à l'exécution de l'aglet et celles de l'API (création, transfert,.), sont implémentées dans le noyau. Les composants de management de la sécurité (*SecurityManager*), de la mémoire cache (*CacheManager*) et de la persistance (*PersistenceManager*) sont proposés par défaut et peuvent être optimisés pour adapter les performances et la sécurité du système. La couche *runtime* ne possède pas de mécanismes de communication propres pour supporter le transfert des aglets ou l'échange de messages. Les échanges entre plates-formes Aglet sont assurés par une couche de communication indépendante dont l'implémentation actuelle utilise le protocole ATP (*Aglet Transfer Protocol*. ATP est un protocole de niveau application, semblable au protocole HTTP dans son fonctionnement. Il n'impose aucun modèle de communication entre les systèmes mais s'appuie sur un modèle de communication de type demande/réponse avec 4 types de demandes pour effectuer le transfert d'agents ou l'échange de messages (Voir la figure D.6) :

Dispatch : cette méthode demande à l'entité destination de reconstruire

un agent avec le code contenu dans la commande et de commencer à l'exécuter. Si la demande est réalisée avec succès, l'expéditeur doit alors terminer l'exécution locale de l'agent et libérer les ressources locales nécessaires à son exécution.

Retract : cette méthode demande à l'entité destination d'expédier l'agent demandé dans la requête. A la réception, l'agent est reconstruit et son exécution relancée. Si l'opération est réalisée avec succès, le destinataire doit suspendre l'exécution de l'agent et libérer les ressources.

Fetch : cette méthode demande au récepteur d'envoyer l'information demandée à l'expéditeur. Cette méthode est utilisée pour transférer les classes nécessaires à l'exécution des agents sur les plates-formes distantes.

Message : cette méthode est utilisée pour transférer un message à un agent distant et envoyer sa réponse.

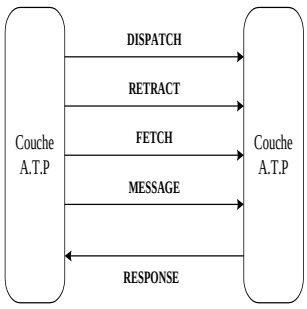


FIG. D.6 – *Echanges de la couche ATP (Agent Transfer Protocol)*

Le protocole ATP utilise les services des couches TCP/IP pour réaliser ses échanges. Pour permettre à la plate forme Aglet d'interopérer avec d'autres plates-formes suivant le standard MASIF (*Mobile Agent System Interoperability Facility*) de l'OMG, l'intégration d'une couche transport basée sur le standard MAF (Mobile Agent Facility) est en cours d'étude.

D.4 Méthode RUN utilisée pour le modèle de simulation des agents mobiles

```
MAgent instproc run {c} {
    $self instvar context_ homenode_ init_ goals_ current_goal_list_
    $self instvar process_delay_
    global ns_ opt count_ latency opt context ANSWSIZE
```

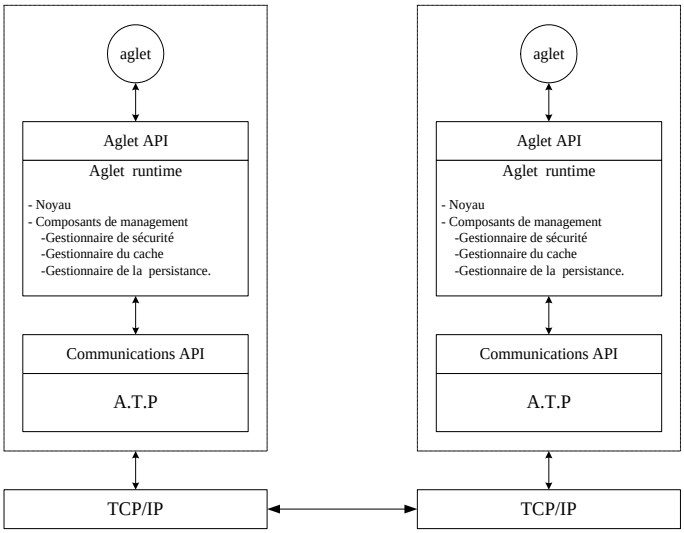


FIG. D.7 – *Architecture de la plate forme Aglet*

```
set node_ [$self node]
set context_ $c

record $self

if { $init_ == 0 } {
    # Initialisation
    # Set goals - for the moment, only dests...
    set goals_ [list [list \
            [list "move" [expr $opt(nn)-1]] \
            [list "move" [$homenode_ set address_]] \
            [list "nop"] \
        ]]
    set current_goal_list_ -1
    set init_ 1 ;# Initialised
}

# Delete the last successfully realized goal
if { [$self eval_goal] == 1 } {
    if { [$node_ set address_] == [expr $opt(nn)-1] } {
        $self set data_size_ $ANSWSIZE ;# Data size grows
    }
```

```
        MSG "Goal successfully realized"
        $self delete_goal
    }

    if { [llength $goals_] == 0 } {
        # No goals
        MSG "Work is done, bye!"
        $self dispose
        return
    }

    # Select a goal
    set nb_loops 0
    set i 0
    set err 1
    while { $err != 0 } {
        incr nb_loops  ;# Take in count the process time (approx.)
        set goal [$self select_goal $i]
        if { $goal == "" } {
            break
        }
        set err [$self realise_goal $goal]
        incr i
    }

    # Reschedule to take in count the process time
    $self resched 1
    return
%}
```

Annexe E

Configuration des nœuds de la plate-forme de tests

Configuration matérielle

- HP Vectra VLI8
- Architecture: i386
- Processeur: Pentium III 650 MHz

Carte Reseau Sans Fil

- Protocole: 802.11b
- Modèle: Cisco Aironet 340 PCI (reconnu en tant que PC4800)
- Qualité du signal - Valeurs du /proc/sys/wireless:
    - Link = 6
    - Level = 226
    - Noise = 0

Système d'exploitation et logiciels d'application

- Linux Suze 8.2
- Kernel: 2.4.20
- Logiciels:
    - Ethereal: 0.9.10
    - GTK+ 1.2.10
    - GLib 1.2.10

- libpcap 0.7
- libz 1.1.4
- ucd-SNMP 4.2.6
- java 1.4.1
- SNMP Java Package - ©Jonathan Sevy
- IBM Aglet Class Library 2.1.0 - ©IBM Corp.

Configuration des logiciels utilisés

Les nœuds sont configurés avec SNORT 1.9[1] associé au module IDMEF 1.1 Alpha pour SUSE 8.2 pro[2, 3, 4].

1. avec libpcap0.7.2.tar.gz, http://www.tcpdump.org
2. libxml2-2.5.1.tar.gz, ftp://xmlsoft.org
3. libidmef-0.7.2.tar.gz http://www.silicondefense.com/idwg/libidmef/index.htm
4. snort-1.9.0-idmef-1.1.tar.gz http://www.silicondefense.com/idwg/snort-idmef

Annexe F

RFC 1213 MIB II TCP Group

Nous donnons dans cette annexe les variables du groupe TCP de la MIB II utilisées pour la détection des chaînes de connexions *Telnet*.

```
SNMP Working Group

RFC 1213                          MIB-II            March 1991

-- the TCP group

-- Implementation of the TCP group is mandatory for all
-- systems that implement the TCP.

-- Note that instances of object types that represent
 -- information about a particular TCP connection are
 -- transient; they persist only as long as the connection
 -- in question.


    -- the TCP Connection table

    -- The TCP connection table contains information about this
    -- entity's existing TCP connections.

    tcpConnTable OBJECT-TYPE
        SYNTAX  SEQUENCE OF TcpConnEntry
        ACCESS  not-accessible
        STATUS  mandatory
```

```
        DESCRIPTION
                "A table containing TCP connection-specific
                information."
        ::= { tcp 13 }

    tcpConnEntry OBJECT-TYPE
        SYNTAX  TcpConnEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "Information about a particular current TCP
                connection.  An object of this type is transient,
                in that it ceases to exist when (or soon after)
                the connection makes the transition to the CLOSED
                state."
        INDEX   { tcpConnLocalAddress,
                tcpConnLocalPort,
                tcpConnRemAddress,
                tcpConnRemPort }
        ::= { tcpConnTable 1 }


    TcpConnEntry ::=
        SEQUENCE {
            tcpConnState
                INTEGER,
            tcpConnLocalAddress
                IpAddress,
            tcpConnLocalPort
                INTEGER (0..65535),
            tcpConnRemAddress
                IpAddress,
            tcpConnRemPort
                INTEGER (0..65535)
        }

    tcpConnState OBJECT-TYPE
        SYNTAX  INTEGER {
                    closed(1),
                    listen(2),
                    synSent(3),
```

```
                synReceived(4),
                established(5),
                finWait1(6),
                finWait2(7),
                closeWait(8),
                lastAck(9),
                closing(10),
                timeWait(11),
                deleteTCB(12)
            }
    ACCESS  read-write
    STATUS  mandatory
    DESCRIPTION
            "The state of this TCP connection.

            The only value which may be set by a management
            station is deleteTCB(12).  Accordingly, it is
            appropriate for an agent to return a 'badValue'
            response if a management station attempts to set
            this object to any other value.

            If a management station sets this object to the
            value deleteTCB(12), then this has the effect of
            deleting the TCB (as defined in RFC 793) of the
            corresponding connection on the managed node,
            resulting in immediate termination of the
            connection.

            As an implementation-specific option, a RST
            segment may be sent from the managed node to the
            other TCP endpoint (note however that RST segments
            are not sent reliably)."
    ::= { tcpConnEntry 1 }

tcpConnLocalAddress OBJECT-TYPE
    SYNTAX  IpAddress
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The local IP address for this TCP connection.  In
            the case of a connection in the listen state which
```

```
                is willing to accept connections for any IP
                interface associated with the node, the value
                0.0.0.0 is used."
        ::= { tcpConnEntry 2 }

    tcpConnLocalPort OBJECT-TYPE
        SYNTAX  INTEGER (0..65535)
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The local port number for this TCP connection."
        ::= { tcpConnEntry 3 }

    tcpConnRemAddress OBJECT-TYPE
        SYNTAX  IpAddress
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The remote IP address for this TCP connection."
        ::= { tcpConnEntry 4 }

    tcpConnRemPort OBJECT-TYPE
        SYNTAX  INTEGER (0..65535)
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The remote port number for this TCP connection."
        ::= { tcpConnEntry 5 }
```

Annexe G

ASN1 OLSR Experimental MIB Specification

Nous donnons dans cette annexe l'extension de la MIB II et la définition des variables utilisées pour la détection de l'attaque N_HOP sur le protocole de routage OLSR.

```
RAHMS-OLSR-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE
    experimental, IpAddress        FROM SNMPv2-SMI ;

rahmsOlsrMIB MODULE-IDENTITY
    LAST-UPDATED "0207051145Z"
    ORGANIZATION "ESEO"
    CONTACT-INFO "rahms@eseo.fr"
    DESCRIPTION  "The MIB module for RAHMS networks"
    ::= \{ experimental 6060 \}

rahms   OBJECT IDENTIFIER ::= { experimental 6262 }

-- the olsr group
olsr      OBJECT IDENTIFIER ::= { rahms 1 }

-- OLSR Neighbor Table
-- The OLSR Neighbor Table contains information concerning this entity's
-- existing neighbors and the status of the link between this host and
-- each of its neighbors
```

```
olsrNeighborTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF olsrNeighborEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION "A table containing OLSR neighbor information."
    ::= { olsr 1 }

olsrNeighborEntry OBJECT-TYPE
    SYNTAX      olsrNeighborEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION "A conceptual row of the olsrNeighborTable containing
                 Information about the connection towards a particular
                 OLSR neighbor. Each row of this table is transient, in
                 that it ceases to exist when (or soon after) the
                 connection with a neighbor is lost."
    INDEX   { olsrNeighborAddress}
    ::= { olsrNeighborTable 1 }

olsrNeighborEntry ::= SEQUENCE {
        olsrNeighborState     INTEGER,
        olsrPreviousNeighborState INTEGER,
        olsrNeighborAddress   IpAddress
    }
olsrNeighborState OBJECT-TYPE
    SYNTAX      INTEGER { ASYM(1),SYM(2),MPR(3),LOST(4) }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION "The state of this OLSR neighbor connection."
    ::= { olsrNeighborEntry 1 }
olsrPreviousNeighborState OBJECT-TYPE
    SYNTAX      INTEGER { ASYM(1),SYM(2), MPR(3),LOST(4) }
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION "The previous state of this OLSR neighbor connection."
    ::= { olsrNeighborEntry 2 }
olsrNeighborAddress OBJECT-TYPE
    SYNTAX      IpAddress
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION "This neighbor IP address"
```

```
    ::= { olsrNeighborEntry 3 }
END
```

Annexe H

Mécanismes de contrôle des flux de données du protocole TCP

Les mécanismes mis en œuvre par TCP pour assurer la fiabilité et le contrôle de flux sont présentés ci-dessous.

Le mode connecté : avant de transférer ses données, l'émetteur s'assure que le destinataire est prêt à les recevoir. Pour cela, il transmet au destinataire un segment de demande d'ouverture de connexion, identifiable par le bit SYN de son en-tête positionné à 1, en indiquant :

- dans le champs *numéro de séquence*, un numéro de séquence initial (ISN, *Initial Sequence Number*), celui-ci permettra d'identifier chaque octet dans le flux,
- dans le champ *fenêtre*, la taille de la fenêtre de réception. Elle indique au destinataire du segment le nombre maximal d'octets transmissibles sans attendre d'acquittement,
- éventuellement des options, comme la taille maximale des segments (MSS, *Maximum Size Segment*) .

Si le récepteur de ce premier segment accepte la connexion, il répond à l'initiateur de la connexion par l'émission d'un segment avec le bit ACK de son en-tête positionné à 1 pour indiquer que le champ ACK est valide. Celui-ci est positionné à la valeur ISN+1 et indique le numéro du premier et prochain octet dans le flux attendu par le récepteur. Dans ce segment, le récepteur positionne le bit SYN à 1 et indique dans le champ numéro de séquence, un ISN relatif aux octets qui seront émis. La phase d'établissement de la connexion se termine lorsque l'initiateur de la connexion acquitte le segment de synchronisation transmis par le récepteur. Cette phase d'ouverture est représentée sur la figure H.1-a. Si l'état du réseau ne permet pas la réception de l'acquittement (ACK)

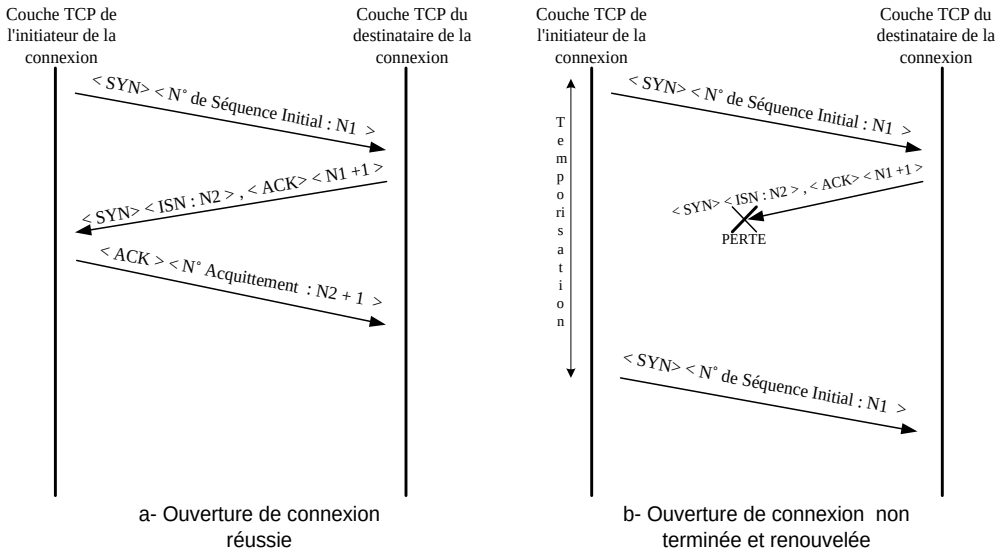


FIG. H.1 – *Ouvertures d'une connexion TCP*

de la demande de synchronisation (figure H.1-b), la demande de connexion est alors renouvelée N fois en doublant le temps d'attente (Temporisation) entre chaque réémission. Si la connexion ne peut être établie à l'issue de ces N tentatives d'ouvertures, l'opération est annulée et TCP en informe l'application.
Le nombre de réémissions est une option de la commande d'ouverture de connexion *OPEN* [66] transmise par l'application à TCP. La valeur de la temporisation et le nombre de retransmissions autorisées dépendent de l'implémentation du protocole TCP.

La figure 3.4 de la page 62 représente les résultats des mesures effectuées sur les stations de notre plate-forme de tests avec les systèmes LINUX Suse 8.0 et MS-Windows 2000 SP1.

Le contrôle des erreurs et l'acquittement des octets transmis: pour détecter les erreurs de transmission, chaque segment possède dans son en-tête une somme de contrôle (*Checksum*) calculée sur l'en-tête et les données avant l'émission du segment. La valeur de cette somme est calculée à l'arrivée du segment. Si celle-ci diffère de la valeur présente dans l'en-tête, le segment n'est pas acquitté.
Après un certain temps, appelé RTO (*Retransmission Time Out*) , le segment qui n'a pas été acquitté est retransmis. Pour une efficacité

maximum, la valeur du temporisateur RTO doit être proche du temps écoulé entre la transmission d'un segment et la réception de son acquittement. Ce temps appelé temps d'aller-retour ou RTT (*Round Trip Time*) varie selon l'état du réseau. Le RTO est calculé en fonction d'un RTT estimé pour anticiper au mieux les évolutions du réseau. Le mécanisme de retransmission rapide permet de réexpédier un segment avant la fin du RTO lorsque plusieurs segments hors séquences sont reçus. Ce mécanisme nécessite qu'à la réception d'un segment hors séquence, le récepteur émette immédiatement un acquittement des segments correctement reçus.
Après plusieurs réémissions d'un segment, si celui-ci ne peut être acquitté, la connexion est libérée à l'initiative de l'application ou de TCP. Les résultats des tests effectués sur notre plate forme de tests avec différentes applications et les systèmes Linux et Windows 2000 sont présentés sur la figure H.2, page 168.

Le contrôle de flux : celui-ci est effectué par le mécanisme de la fenêtre glissante. A l'établissement de la connexion et dans chaque segment d'acquittement, le récepteur indique la taille de sa fenêtre de réception, appelée *rwnd* (*Reception Window*). Celle-ci correspond à la capacité de stockage du récepteur et indique à l'émetteur le nombre d'octets qu'il peut transmettre sans attendre la réception d'un acquittement. Quand le nombre d'octets transmis atteint la valeur cette fenêtre, l'émetteur bloque ses émissions. La réception par l'émetteur de l'acquittement des octets précédemment transmis lui permet de déplacer d'autant sa fenêtre de gestion des émissions et ainsi de reprendre la transmission.

Le contrôle de congestion : développé pour Internet, TCP doit aussi gérer les *congestions* résultant de la saturation des nœuds du réseau et principalement des routeurs. Pour les équipements d'extrémité, une congestion se traduit par une augmentation des délais de transmission et par conséquence, par une réémission des données. Si ce phénomène n'est pas maîtrisé, il y a un risque d'effondrement du réseau. Pour contrôler les congestions, TCP utilise une fenêtre de congestion appelée *cwnd* (*Congestion Window*). Afin de limiter le flux, la taille de la fenêtre d'émission est ajustée à la valeur suivante :

$$\text{Fenêtre d'émission} = \min(rwnd, cwnd).$$

La fenêtre de congestion, *cwnd*, est initialisée à la valeur du MSS (*Maximum Size Segment*) et double sa valeur à chaque RTT.

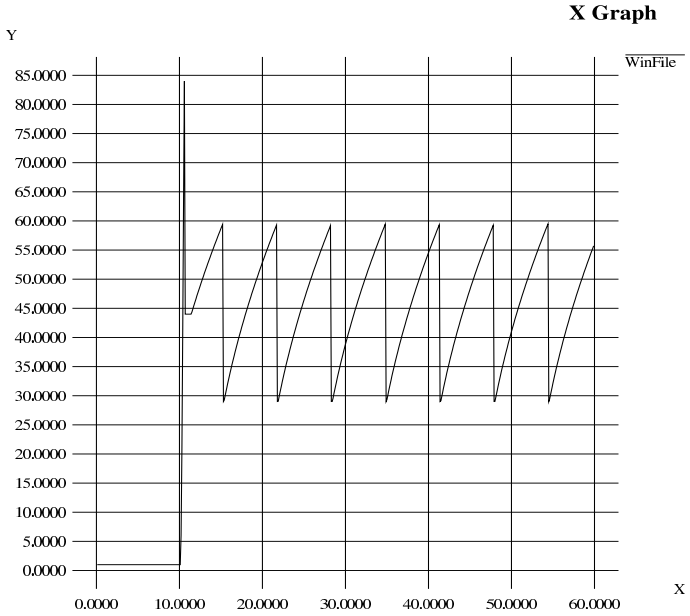


FIG. H.2 – *Évolution de la fenêtre de congestion. Y: cwnd (Segment) - X: t (Seconde)*

Nous relevons et nous représentons, sur la Fig H.2, l'évolution des variations de la fenêtre de congestion, *cwnd*, lors d'un transfert point à point entre deux nœuds fixes et d'un flux continu de données.
Ce processus d'augmentation de la fenêtre de congestion est appelé le démarrage lent (*slow start*) et se poursuit jusqu'à la détection de la perte d'un segment ou tant que *cwnd* possède une valeur inférieure au seuil de démarrage lent, *ssthresh* (*slow start threshold*). Au delà de ce seuil, TCP entre dans la phase d'évitement de congestion (*congestion avoidance*) et ralentit le rythme d'augmentation de la fenêtre *cwnd*. Pendant cette phase d'évitement de congestion, la fenêtre de congestion est augmentée de la valeur d'un MSS par RTT.
La valeur du seuil *ssthresh* est fixée à une fois et demie la valeur initiale de *cwnd*. Différentes implémentations du protocole TCP ([8], [28]) permettent d'aboutir à des recouvrements rapides en fixant la valeur de seuil à la moitié de la fenêtre d'émission.

Glossaire

Glossaire

AODV : Ad hoc On demand Distance Vector

DSDV : Destination Sequence Distance Vector
DSSS : Direct Sequence Spread Spectrum

FSD : Finite State Diagram
FTP : File Transfer Protocol

IBSS : Independent Basic Service Set
IDMEF : Intrusion Detection Message Exchange Format
IHM : Interface Homme Machine
IPSec: Internet Protocol Security

LAN : Local Area Network
LIDS : Local Intrusion Detection System
LLC : Logical Link Control

MAC : Medium Access Control
MAN : Metropolitan Area Network
MANET : Mobile Ad hoc NETworks
MIB : Management Information Base
MPR : MultiPoint Relay
MS : MPR Selector
MSS : Maximum Size Segment

NIDS : Network Intrusion Detection System

OLSR : Optimized Link State Routing Protocol

PAN : Personal Area Network

RMI : Remote Method Invocation
RTO : Retransmission Time Out
RTT : Round Trip Time

SNMP : Simple Network Management Protocol
SSL: Secure Socket Layer

TC : Topology Control
TCP : Transmission Control Protocol

UDP : User Datagram Protocol

WAN : Wide Area Network
WEP : Wired Equivalent Privacy
WI-FI : Wireless-Fidelity

XML : eXtensible Markup Langage