N°d'ordre : 23/2012-M/Info

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE « HOUARI BOUMEDIENE »
FACULTE D'ELECTRONIQUE ET D'INFORMATIQUE

MÉMOIRE

Présenté pour l'obtention du diplôme MAGISTER

En: INFORMATIQUE

Spécialité : Informatique Mobile

Par: SAÏAH AMIN

Sujet:

Synchronisation de Temps Sécurisée dans les Réseaux de Capteurs Sans Fil

Soutenu Publiquement le 25/06/2012, devant le jury composé de :

Mr M. BENCHAIBA	**Maîtres de Conférences/A, à l'USTHB**	**Président**
Mr N. BADACHE	**Professeur, à l'USTHB**	**Directeur de mémoire**
Mr A. DERHAB	**Maîtres de Recherches/A, au CERIST**	**Examinateur**
Mr D. DJENOURI	**Maîtres de Recherches/A, au CERIST**	**Examinateur**
Mme C. BENZAID	**Maîtres de Conférences/B, à l'USTHB**	**Invitée**

Je dédie ce travail à tous ceux qui me sont chers...

A ma très adorable maman ;

A toutes ma famille ;

A tous mes amis ;

A toutes les personnes qui m'ont aidée de près ou de loin.

Amin.

Remerciements

Louange à notre Seigneur 'ALLAH' qui m'a doté de la merveilleuse faculté de raisonnement. Louange à notre créateur qui m'a incité à acquérir le savoir. C'est à lui que m'adresse toute ma gratitude en premier lieu.

En second lieu, je tiens à remercier Mr N.BADACHE mon promoteur et Mme C.BENZAID ma co-promotrice pour leur sympathie, leur disponibilité, leurs idées, leurs conseils et leurs encouragements qui m'ont permis de mener à bien ce mémoire.

Je remercie particulièrement les honorables membres de jury qui ont pris la peine de lire et d'évaluer ce mémoire.

Je tiens aussi à remercier l'ensemble des enseignants de l'USTHB, sans exception, ainsi que les employés qui ont rendu plus confortable notre formation au sein de l'université.

Un profond remerciement à Mr A.KHOUALDIA directeur des transports de la wilaya de chlef qui m'a aidé pour continuer mon magistère.

Un grand MERCI à tous les membres de ma famille et spécialement ma mère qui m'ont soutenu tout au long de mes études et qui ont fait en sorte, par leur amour, leur affection et leur soutien financier, que je puisse avoir les meilleures conditions possibles pour continuer mes études et aller de l'avant. Qu'ils trouvent ici ma gratitude et mon amour pour eux.

Résumé

La plupart des applications des réseaux de capteurs sans-fil (RCSF) exigent que les horloges des nœuds capteurs soient synchronisées. Trois approches de base de synchronisation de temps ont été proposées : la synchronisation *Emetteur-Récepteur* (SRS), la synchronisation *Récepteur-Récepteur* (RRS) et la synchronisation *Seulement-Récepteur* (ROS). L'approche ROS offre de meilleures performances (coût de communication et efficacité énergétique) par rapport aux approches SRS et RRS. Un attaquant peut certainement attaquer les protocoles de synchronisation de temps à cause de leur importance dans les applications RCSF. Ceci exige la sécurité pour la synchronisation de temps dans les réseaux de capteurs ; c'est-à-dire développer de nouveaux protocoles qui permettent de fournir une synchronisation de temps sécurisée dans les réseaux de capteurs. Récemment, plusieurs protocoles de synchronisation de temps sécurisée ont été proposés pour les RCSF dans les environnements hostiles. Ces solutions concentrent sur les deux approches SRS et RRS, et d'après nos études et connaissances, aucune approche de sécurité n'est fournie pour l'approche ROS.

Notre travail consiste en la conception d'une nouvelle méthode pour la synchronisation de temps sécurisée dans les RCSF permettant de limiter l'impact des informations de temps malicieuses et de permettre aux nœuds honnêtes d'exclure les nœuds sources de ces informations dans la synchronisation de temps. Notre approche consiste à fournir un mécanisme de sécurité pour l'approche ROS. Ainsi, nous obtenons un nouveau protocole de synchronisation de temps sécurisée, que nous avons appelé « SPBS » (*Secure Pairwise Broadcast time Synchronisation*).

Pour l'analyse et la réalisation, nous avons implémenté et analysé notre protocole dans un environnement réel en utilisant la plate-forme MICAz. Les résultats d'expérimentation obtenus ont permis de montrer l'efficacité de notre solution par rapport aux solutions existantes en faisant références aux paramètres de performances et de sécurité visés.

Mots clés: RCSF, Synchronisation de temps, ROS, SPBS, Sécurité.

Abstract

Most wireless sensor network (WSN) applications require that the clocks of sensor nodes are synchronized. Three basic approaches to time synchronization have been proposed: the sender-receiver synchronization (SRS), receiver-receiver synchronization (RRS) and receiver-only synchronization (ROS). The ROS approach offers better performance (communication cost and energy efficiency) compared to SRS and RRS approaches. An attacker can certainly attack the time synchronization protocols due to their importance in WSN applications. This requires security for time synchronization in sensor networks; that is to say, develop new protocols to provide secure time synchronization in sensor networks. Recently, several secure time synchronization protocols in WSNs have been proposed in hostile environments. These solutions focus on two approaches SRS and RRS, and to the best of our study and knowledge, no security approach is provided for ROS approach.

Our work consists on the design of a new method for secure time synchronization in wireless sensor networks (WSN) to limit the impact of incorrect time information and allow honest nodes to exclude bogus information source in time synchronization. Our approach is to provide a secure mechanism for a ROS approach. Thus, we obtain a new protocol for secure time synchronization, which we called « SPBS » (*Secure Pairwise Broadcast Synchronization time*).

For analysis and implementation, we implemented our protocol and analyzed it in a real environment using the platform MICAz. Experiment results gotten out made it possible to show the effectiveness of our solution compared to the existing solutions by referring to the parameters of performances and security requirements.

Key words: WSN, Time synchronization, ROS, SPBS, Security.

Publications

Nous avons communiqué notre proposition auprès de la communauté internationale en les publiant dans une conférence internationale :

C. BENZAID, A. SAIAH and N. BADACHE. "***Secure Pairwise Broadcast Synchronization in Wireless Sensor Networks***". The 7th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2011), 29 June 2011. Digital Object Identifier: 10.1109/DCOSS .2011.5982217.

Table des matières

Liste des Figures

Liste des Tableaux

Introduction Générale

L'essor des technologies des systèmes *micro-électro-mécaniques*[1], offre aujourd'hui de nouvelles perspectives dans plusieurs domaines d'applications. L'évolution dans ce domaine a donné naissance à des microcomposants, appelés *micro-capteurs* ayant plusieurs fonctionnalités telles que le captage, le traitement et la transmission / réception de données. Cette évolution conjuguée à celle des technologies de communications sans fil a fait du déploiement des réseaux de capteurs une vision pragmatique.

Les réseaux de capteurs sans fil ont reçu beaucoup d'attention récemment due à leurs applications étendues, telles que la poursuite des cibles, la surveillance des infrastructures critiques, et exploration scientifique dans les environnements dangereux. Les réseaux de capteurs sans fil se composent typiquement d'un grand nombre de petits nœuds capteurs qui capturent les modifications environnementales (par exemple la lumière, le son, la température, le mouvement, et la pression) et les rapportent aux nœuds de contrôle. Les nœuds de contrôle peuvent ultérieurement traiter les données rassemblées de nœuds capteurs, disséminer des commandes de contrôle aux nœuds capteurs, et connecter le réseau à un réseau câblé traditionnel. Les nœuds de contrôle peuvent avoir des processeurs de classe workstation / laptop, assez de mémoire, d'énergie, et de puissance de calcul.

Plusieurs applications réseaux de capteurs sans fil (exp : la fusion de données **[9]**, la poursuite des cibles **[10]**, et le temps de réveil **[35]**) exigent que les horloges des nœuds capteurs soient synchronisées (*une notion commune de temps*). Dans les applications de fusion de données, plusieurs algorithmes **[9]** ont traité l'ordre de lectures de capteurs par le temps d'occurrence (par exemple, le moment où un incendie de forêt a été capturé). Dans les applications de poursuite des cibles **[10]**, les nœuds capteurs ont besoin de l'emplacement et du moment où la cible est capturée afin de déterminer correctement la direction et la vitesse de la cible. Dans l'exemple de temps de réveil, plusieurs approches ont été proposées pour améliorer l'efficacité énergétique par la commutation des nœuds capteurs entre le mode active et le mode sommeil **[35]**. En outre, des protocoles de la couche MAC (par exemple, **[36]**) réalisent l'accès multiple au support de communication partagé en assignant des time-slots à un groupe de nœuds. Ainsi, les nœuds capteurs doivent avoir une horloge synchronisée pour accéder à leurs time-slots sans collision avec d'autres nœuds.

Les horloges des nœuds capteurs sont, généralement basées sur les oscillateurs à cristaux qui fournissent une heure locale pour chaque nœud capteur du réseau. L'horloge locale d'un nœud capteur peut être dérivée par rapport à un autre nœud capteur, et la dérive d'horloge est intolérable pour les applications réseaux de capteurs sans fil. Par conséquent, la synchronisation de temps est un composant important d'un réseau de capteurs sans fil pour fournir une ***horloge commune*** dans des nœuds capteurs.

[1] Systèmes comprenant un ou plusieurs éléments mécaniques, utilisant l'électricité comme source d'énergie, en vue de réaliser une fonction de capteur et / ou d'actionneur avec au moins une structure présentant des dimensions micrométriques. **[15]**

En raison des contraintes de ressource sur les nœuds capteurs, les protocoles de synchronisation de temps traditionnels (par exemple, NTP **[14]**) ne peuvent pas être directement appliqués dans les réseaux de capteurs. Récemment, plusieurs protocoles de synchronisation de temps (**[15, 16, 17, 18, 19, 20, 21, 22]**) ont été proposés pour les réseaux de capteurs sans fil dans les environnements bénis (non-hostile). Tous les protocoles proposés se focalisent sur l'un des approches suivantes : la synchronisation *Emetteur-Récepteur* (SRS, **[18]**), la synchronisation *Récepteur-Récepteur* (RRS, **[17]**) et la synchronisation *Seulement-Récepteur* (ROS, **[16]**). Dans SRS, le récepteur se synchronise suivant les estampilles reçues du nœud référence. Dans RRS, le nœud référence diffuse un message et les autres nœuds se synchronisent suivant le temps de réception de ce message. Dans ROS, les nœuds capteurs se synchronisent en écoutant seulement les messages de synchronisation échangés entre deux nœuds capteurs qui exécutent l'approche SRS. L'approche ROS offre de meilleures performances par rapport SRS et RRS car le coût de communication ne dépend pas le nombre de nœuds du réseau. Ceci réduit la consommation énergétique dans le réseau.

Dans les environnements hostiles, un attaquant peut certainement attaquer le protocole de synchronisation à cause de son importance. Donc, toutes les techniques de synchronisation de temps dans les réseaux de capteurs sans fil ne peuvent pas survivre aux attaques malveillantes dans les environnements hostiles. Noter que tous les protocoles de synchronisation de temps se basent sur des échanges de messages de temps. Pour tromper ces protocoles, l'attaquant peut lancer plusieurs attaques tel que : falsifier ou modifier les messages de synchronisation de temps, brouiller les canaux de communication, retarder les messages de synchronisation de temps, personnifier d'autres nœuds légitimes,…etc. Ceci exige de sécuriser les protocoles la synchronisation de temps dans les réseaux de capteurs ; c'est-à-dire développer de nouveaux protocoles qui permettent de fournir une synchronisation de temps sécurisée dans les réseaux de capteurs. Récemment, plusieurs protocoles de synchronisation de temps sécurisée (par exemple, **[43, 45, 46, 50, 52, 56, 60, 63]**) ont été proposés pour les réseaux de capteurs sans fil dans les environnements hostiles. Les protocoles de synchronisation de temps sécurisée proposés récemment sécurisent l'approche SRS ou RRS, et aucune approche de sécurité n'est fournie pour l'approche ROS.

Dans ce travail, nous nous intéressons à la synchronisation de temps sécurisée pour que les réseaux de capteurs sans fil fonctionnent dans les environnements hostiles. Notre proposition consiste à fournir un mécanisme de sécurité pour l'approche ROS qui offre de meilleures performances (coût de communication, consommation énergétique,…) parmi les approches existantes. Elle permet de limiter l'impact des informations de temps malicieuses et de permettre aux nœuds honnêtes d'exclure les nœuds sources de ces informations dans la synchronisation de temps.

Ce mémoire est organisé en six chapitres :

Le premier et le deuxième chapitre sont un état de l'art sur les réseaux de capteurs sans-fil et la synchronisation d'horloges. Ils se focalisent sur une étude sur les réseaux de capteurs sans-fil, le problème de la synchronisation, les protocoles de synchronisation dans les réseaux traditionnels, et finalement les protocoles de synchronisation dans les réseaux de capteurs et leur classification.

Le troisième et le quatrième chapitre sont un état de l'art sur la sécurité et la synchronisation de temps sécurisée dans les réseaux de capteurs. Ils se focalisent sur les mécanismes de sécurité

cryptographiques dans les réseaux de capteurs, les diverses attaques possibles contre la synchronisation de temps, les effets de ces attaques sur quelques applications de réseaux de capteurs, et finalement les solutions proposées récemment pour la synchronisation de temps sécurisée.

Le cinquième et le sixième chapitre sont consacrés à la description détaillée et une évaluation de performances de notre proposition. Pour l'implémentation, le système TinyOS est utilisé qui est le plus répondu des systèmes d'exploitation pour les réseaux de capteurs sans fil. Nous avons expérimenté notre protocole dans un environnement réel. Les résultats obtenus ne sont pas les mêmes avec la simulation, l'expérimentation nous a donné le vrai comportement de notre approche.

Enfin, nous terminons par une conclusion générale qui résume nos observations retenues de l'étude de l'état de l'art et de la discussion des résultats obtenus par notre approche, et présente les perspectives envisageables à notre travail.

Chapitre I : Les Réseaux de Capteurs Sans-fil (RCSF)

1 Introduction

Les récentes avancées dans le domaine des technologies sans-fil et électronique ont permis le développement à faible coût de minuscules capteurs consommant peu d''énergie (solution low-cost et low-power). Ces capteurs ont trois fonctions; la capture des données (de type son, vibration, optique, thermique, etc.), le calcul des informations à l'aide des valeurs collectées et l'envoi des résultats de calcul à travers un réseau de capteurs sans-fil. Un réseau de capteurs est souvent composé d'un nombre très important de nœuds qui sont, soit placés à un endroit précis ou bien dispersés aléatoirement (généralement déployés par voie aérienne à l'aide d'avion ou d'hélicoptère). Ce dispersement aléatoire des capteurs nécessite que le protocole utilisé pour les réseaux de capteurs possède des algorithmes d'auto-organisation. Afin de résister au déploiement, ces capteurs doivent être très solides et doivent survivre dans les conditions les plus extrêmes dictées par leur environnement d'utilisation (par exemple feu ou eau). Le faible coût de fabrication des capteurs, les fonctions qui peuvent être réalisées par ces derniers ainsi que leur résistance aux contraintes de déploiement, permettent aux réseaux de capteurs d'intéresser plusieurs domaines d'application.

Dans ce chapitre, nous décrirons l'architecture d'un nœud capteur, les réseaux de capteurs sans-fil et leurs applications.

2 Le nœud capteur

C'est un mini-composant, qui permet d'acquérir des données sur son environnement **[1]**, les traiter et les communiquer. Son intégration est une tâche difficile à réaliser en tenant compte de certaines contraintes : l'espace mémoire, la consommation énergétique,…etc. La **Figure I.1** représente l'architecture générale d'un nœud capteur.

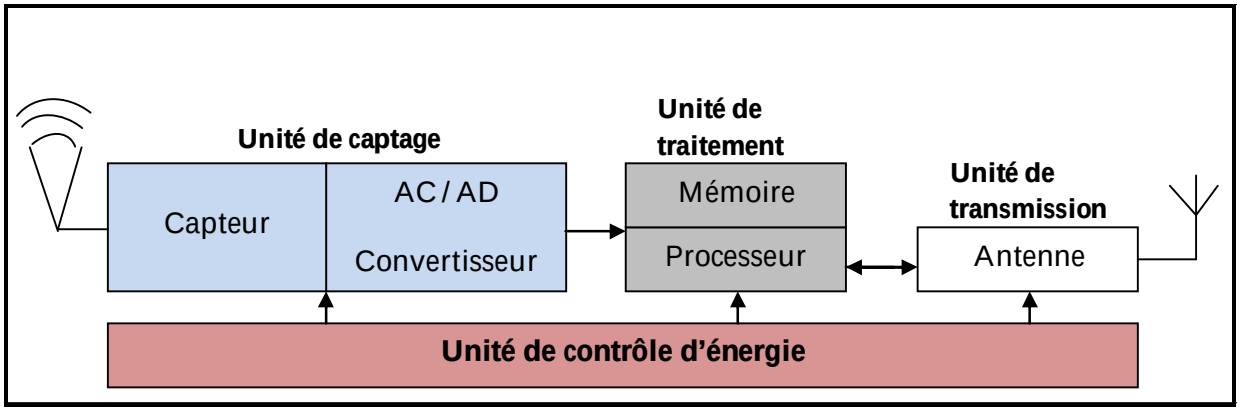


Figure I.1: Modèle d'un nœud capteur. [1]

- *Unité de captage* : Elle est composée de deux sous-unités : une *unité d'acquisition*, qui permet de détecter les mesures désirées par un capteur, et d'une *unité de traitement des signaux* qui transforme les mesures analogiques détectées en signaux numériques par un convertisseur analogique numérique, comme l'indique la figure **Figure I.1**.
- *Unité de traitement*: Elle est composée d'une mémoire (unité de stockage) et d'un processeur (unité de calcul) qui permet d'effectuer des calculs simples pour que ce nœud puisse collaborer avec les autres nœuds du réseau. De plus, elle possède deux interfaces: la première liée avec l'unité de captage par laquelle, elle reçoit les mesures détectées. La seconde liée avec l'unité de transmission par laquelle, elle communique les données qu'elle a traitées.
- *Unité de transmission* : Elle est responsable de toutes les émissions et réceptions de données qui représentent l'état actif du nœud. Par ailleurs, le nœud peut se mettre en veille ou écouter seulement le trafic. L'unité de transmission est l'unité qui consomme le plus d'énergie par rapport aux précédentes unités.
- *Unité de contrôle d'énergie* : L'énergie est la ressource la plus importante dans un réseau de capteurs, puisque elle influe directement sur la durée de vie des micro-capteurs et du réseau en entier. Elle est responsable de répartir l'énergie disponible aux autres modules et de réduire les dépenses en commutant les nœuds capteurs entre le mode active et le mode sommeil.

3 Modèles d'un nœud capteur

Les nœuds capteurs se déclinent en une multitude de modèles en relation avec l'application à laquelle ils sont destinés. Parmi les modèles les plus courants, on trouve les capteurs MICA développés par l'université de Berkeley et commercialisés par Crossbow, les capteurs Imote commercialisés par Crossbow ainsi que les capteurs TinyNode développés, pour des applications réelles liées à l'industrie, par Shockfish SA **[2]**.

Bien qu'ils soient différents, ces modèles ont en commun les mêmes composants de base. La **Figure I.1** illustre la description des composants les plus courants d'un nœud de capteurs. Les différents composants de chaque modèle ainsi que leurs caractéristiques sont donnés dans le tableau (**Tableau I.1**) ci-dessous.

	Modèle	Micro-Contrôleur	Type Radio	Radio (kbps)	RAM	Flash	EEPROM	Batterie
Crossbow Technology	MICA2	Atmega128L MPR400 (8-bit)	ChipCon CC1000	38	4 KB	128 KB	4 KB	2xAA
	MICAz	Atmega128L MPR2400 (8-bit)	ChipCon CC2420	250	4 KB	128 KB	4 KB	2xAA
	TelosB	TI MSP 430 (16-bit)	ChipCon CC2420	250	10 KB	48 KB	16 KB	2xAA
	Imote2	Intel PXA271 (32-bit)	ChipCon CC2420	250	32 MB	32 MB		3xAAA
Shockfish SA	TinyNode	TI MSP 430 (16-bit)	Semtech XE1205	153	10 KB	48 KB	16 KB	2/3xAA

Tableau I.1 : Les différentes technologies des nœuds capteurs.

3.1 MICA2

Le capteur MICA2 **[3]** est un capteur de 3ème génération utilisé pour les réseaux de capteurs sans-fil et à faible consommation, voir le **Tableau I.1**. Ce type de capteurs a été développé par l'université de Berkeley et est utilisé dans les applications suivantes :

- Contrôles environnementaux
- Surveillance et sécurité
- Réseaux de capteurs de grande capacité (+1000 nœuds)

3.2 MICAz

Le capteur MICAz **[3]** est un capteur comme MICA2 utilisé pour les réseaux de capteurs sans-fil et à faible consommation. Il utilise la radio ChipCon CC2420 qui a un débit élevé par rapport à la ChipCon CC1000, voir le **Tableau I.1**. Plus des applications de MICA2, le capteur MICAz est aussi utilisé dans les applications suivantes :

- Acoustique, Vidéo, Vibration et d'autres captures de données avec une vitesse élevée

3.3 TelosB

La plate-forme TelosB **[3]** a été élaborée et publiée à la communauté scientifique par l'université de Berkeley. Cette plate-forme offre une faible consommation d'énergie permettant une longue autonomie de la batterie ainsi qu'un éveil rapide de l'état de veille. Le microcontrôleur TPR2420 utilisé dans TelosB, est compatible avec la distribution open-source de TinyOS. Ce type de nœud peut être utilisé dans les applications suivantes :

- Plate-forme à faible puissance pour le développement de la recherche
- Expérimentations des réseaux de capteurs sans-fil

3.4 Imote2

Le Imote2 **[3]** est une plate-forme avancée de nœuds capteurs sans-fil. Il est construit autour d'un processeur XScale PXA271 à faible puissance et intègre également une radio compatible 802.15.4. Ce type de nœud, développé par Crossbow, est utilisé dans les applications suivantes :

- Traitement des images numériques
- Contrôle et analyse industriels
- Surveillance des séismes et des vibrations

3.5 TinyNode

Le TinyNode **[2]** est un nœud capteur développé par Shockfish SA. Il est optimisé de telle sorte à supporter TinyOS. Ce type de nœud, on le retrouve dans les applications suivantes :

- Surveillance de l'environnement
- Agriculture de précision
- Gestion de stationnement

4 Architecture des RCSF

Un Réseau de Capteurs Sans-Fil (RCSF) est un ensemble de capteurs variant de quelques dizaines d'éléments à plusieurs centaines, parfois plus, utilisant des liens sans-fil pour la

communication. Chaque réseau de capteurs a la capacité de collecter des données à partir d'un champ de captage, qui définit la zone d'intérêt pour le phénomène capté. A l'aide d'une architecture multi-sauts, un RCSF transmet les données collectées à un nœud PUITS (plusieurs à un), voir la **Figure I.2**. Ce dernier est considéré comme un point de collecte et peut transférer les données collectées via internet ou satellite à un ordinateur central "gestionnaire de tâche" pour leur traitement. De plus, des requêtes précisant le type de données requises et le début/arrêt de captage peuvent être envoyées par le biais du nœud PUITS aux nœuds capteurs (un à plusieurs).

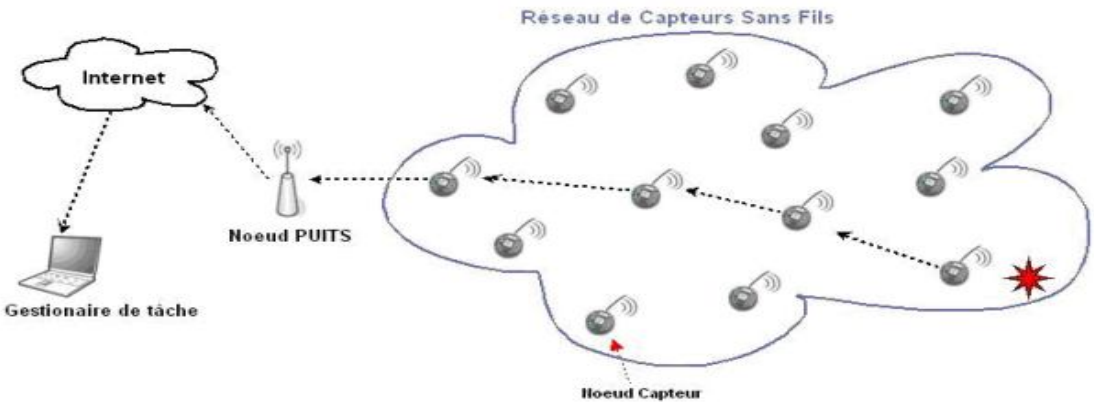


Figure I. 2: Architecture d'un réseau de capteurs sans-fil. [1]

Selon **[4]**, il existe deux types d'architectures pour les réseaux de capteurs sans-fil :

4.1 Les réseaux de capteurs sans-fil plats

Dans une topologie plate, tous les nœuds possèdent le même rôle. Les nœuds sont semblables en termes de ressources. Selon le service et le type de capteurs, une densité de capteurs élevée (plusieurs nœuds capteurs/m^2) ainsi qu'une communication multi-sauts peut être nécessaire pour l'architecture plate. En présence d'un très grand nombre de nœuds capteurs, le passage à l'échelle devient critique. Le routage et le contrôle d'accès au médium (MAC) doivent gérer et organiser les nœuds d'une manière très efficace en termes d'énergie.

4.2 Les réseaux de capteurs sans-fil hiérarchiques

Afin d'augmenter la scalabilité du système, les topologies hiérarchiques ont été introduites en divisant les nœuds en plusieurs niveaux de responsabilité. L'une des méthodes les plus employées est le clustering, où le réseau est partitionné en groupes appelés "clusters". Un cluster est constitué d'un chef (cluster-head) et de ses membres.

5 Caractéristiques d'un RCSF

Les réseaux de capteurs sont comme les réseaux Ad hoc caractérisés par l'absence de l'infrastructure fixe, une topologie dynamique, des capacités matérielles limitées, des liens de communication non fiables, des risques de sécurité ...etc. A toutes ces contraintes héritées des réseaux Ad-hoc viennent s'ajouter d'autres contraintes spécifiques aux réseaux de capteurs **[5]**:

- *Limitations de ressources physiques* : A cause de la miniaturisation des composants électroniques, les performances des nœuds capteurs (mémoire, CPU) sont limitées. Ils sont alimentés par des sources d'énergie non renouvelables.

- *Le nombre des nœuds capteurs* : Pour capter ou surveiller un phénomène, les nœuds capteurs sont employés en grand nombre. Ceci est aussi encouragé par leur prix qui est de plus en plus bas.
- *Modèle de communication* : Les nœuds dans les RCSFs communiquent selon un paradigme plusieurs-à-un (many to one). En effets, les nœuds capteurs collectent des informations à partir de leur environnement et les envoient toutes vers un seul nœud qui représente le centre de traitement.
- *Le déploiement aléatoire* : Afin de surveiller un phénomène, les nœuds capteurs sont distribués d'une façon très aléatoire dans la zone d'observation.
- *Le changement rapide de topologie* : La topologie d'un réseau de capteurs peut changer fréquemment notamment dans le cas où le déplacement des nœuds est influencé par les conditions de l'environnement (par exemple le vent).
- *La durée de vie :* constitue un facteur important dans l'utilisation des réseaux de capteurs. La maximisation de ce facteur est équivalente à celle du processus de surveillance que le réseau est censé mettre en œuvre. Cela fait de sa maximisation un objectif de priorité.
- *Absence d'adressage fixe des nœuds :* Les nœuds dans les réseaux sans fil classiques sont identifiés par des adresses IP. Cependant, cette notion n'existe pas dans les RCSFs. Ces derniers utilisent un adressage basé sur l'attribut du phénomène capté, on parle donc de l'adressage basé-attribut.
- *Sécurité* : En plus des problèmes de sécurité rencontrés dans les réseaux Ad Hoc en général, les RCSF rencontrent d'autres handicaps dus à leurs challenges ; à savoir l'autonomie et la miniaturisation des capteurs. Cela engendre l'inapplicabilité des mécanismes de défense utilisés dans les réseaux Ad Hoc tout en créant d'autres mécanismes de sécurité pour les RCSF. De plus, l'absence d'une sécurité physique dans l'environnement hostile où ils sont déployés expose les nœuds à un danger qui tend vers la falsification de l'information. En effet, les nœuds capteurs eux-mêmes sont des points de vulnérabilité du réseau car ils peuvent être modifiés, remplacés ou supprimés.

6 Application des RCSF

Il existe 3 grandes classes de réseaux de capteurs **[5]** : réseau de collection des données d'environnements, réseau de surveillance et sécurité et enfin les réseaux de poursuite.

6.1 Réseaux de collection des données d'environnement (Data Fusion)

Ces réseaux ont été conçus en général pour la collecte périodique des données environnementales puis leur transmission vers une station de base. Les nœuds du réseau peuvent avoir plusieurs fonctionnalités et différents types de capteurs. Ce type de réseau nécessite généralement un flux de données faible, une durée de vie importante et une topologie statique. Par contre, il ne présente pas de contraintes de types temps de chemin d'un paquet. Un exemple de tel réseau est un réseau dans lequel les nœuds mesurent la température, l'humidité et les radiations solaires au sein d'un champ agricole **[9]**.

6.2 Réseaux de surveillance et sécurité

Les réseaux de surveillance sont constitués de nœuds fixes qui contrôlent d'une façon continue la détection d'une anomalie dans le fonctionnement d'un nœud. La différence entre ce réseau et le premier réseau réside dans le fait que les nœuds ne sont pas en train de collecter des données. Ici

les nœuds transmettent seulement les rapports concernant une violation de la sécurité. Ce type de réseau exige un excellent temps de chemin d'un paquet pour propager les alarmes. Un exemple de tel réseau est la détection des incendies dans les forêts.

6.3 Réseaux de poursuite (Target Tracking)

Cet usage de réseau concerne généralement la poursuite d'un objet dans un lieu contrôlé par un réseau de capteurs. Ce type de réseau est caractérisé par une topologie mobile ce qui engendre une instabilité de la communication des nœuds. Ces réseaux sont généralement développés par l'armée comme le projet américain «SmartDust» dans la poursuite d'un tireur d'élite **[10]**.

7 Systèmes embarqués

Les systèmes embarqués sont des systèmes d'exploitation prévus pour fonctionner sur des machines de petite taille, telles que des nœuds capteurs. Les systèmes d'exploitation pour les nœuds de RCSF sont généralement moins complexes que les autres systèmes d'exploitation. Ceci à cause des exigences particulières des applications de réseau de capteurs et des contraintes de ressources des nœuds capteurs. Plusieurs systèmes d'exploitation sont conçus pour les nœuds de RCSF. Parmi ces systèmes nous citons TinyOS **[6]**, SOS **[7]**, etc. TinyOS est le plus répondu des systèmes d'exploitation pour les RCSFs. TinyOS est un système d'exploitation open source développé par l'université de Berkeley. Sa conception a été entièrement réalisée en NesC **[8]**, langage orienté composant syntaxiquement proche du C.

8 Conclusion

La flexibilité, la tolérance aux fautes, le prix réduit et les moyens rapides de déploiement des réseaux de capteurs offrent des possibilités énormes de développement pour différents domaines d'application et permettent de penser que les réseaux de capteurs feront bientôt partie intégrante de nos vies et satisferont sûrement les plus grands projets. Cependant, la mise en place d'une application de réseau de capteurs doit prendre en considération les contraintes qui caractérisent les nœuds capteurs et qui sont principalement : la consommation d'énergie, les contraintes physiques, le changement de la topologie et l'adaptation à l'environnement. Ces contraintes exigent que de nouvelles techniques de gestion de réseau sans-fil soient mises au point.

En outre, pour assurer la fiabilité du système, de nouveaux mécanismes et mesures de sécurité doivent être mis en place afin d'éliminer les vulnérabilités liées à la sécurité dans les RCSFs.

Dans les prochains chapitres, nous aborderons de façon plus complète l'aspect de sécurité de synchronisation de temps dans les RCSFs.

Chapitre II : Synchronisation de Temps

1 Introduction

La synchronisation d'horloges est le processus d'assurer que les processeurs physiquement distribués ont ***une notion commune de temps***. Dans les systèmes centralisés[1], on n'a pas besoin de synchroniser le temps parce qu'il n'y a aucune ambiguïté du temps. Un processus obtient le temps simplement en faisant un appel système au noyau. Quand un autre processus essaie de l'obtenir, il aura une valeur supérieure ou égale. Ainsi, l'ordre des événements et le temps auquel ils se sont produits est établi facilement. Dans les systèmes distribués, il n'y a pas d'horloge globale. Chaque processeur a sa propre horloge interne et sa propre notion du temps. Ceci exige le besoin de la synchronisation de temps entre les nœuds d'un système réparti. Dans la pratique, ces horloges peuvent facilement dériver de quelques secondes par jour **[11]**, accumulant des erreurs significatives dans le temps. En outre, elles ne peuvent pas rester tout le temps synchronisées bien qu'elles pourraient être synchronisées quand elles commencent parce que chaque horloge a sa propre fréquence. Ceci pose évidemment de sérieux problèmes aux applications qui dépendent de la notion de temps synchronisé. Pour la plupart des applications et algorithmes dans les systèmes distribués, on a besoin de connaître le temps dans un ou plusieurs des aspects suivants :

- L'heure de la journée à laquelle un événement s'est produit sur une machine spécifique dans le réseau.
- L'intervalle de temps entre deux événements qui se sont produits sur différentes machines du réseau.
- L'ordre relatif des événements qui se sont produits sur différentes machines du réseau. À moins que les horloges dans chaque machine aient une notion commune du temps, les requêtes basées sur le temps ne peuvent pas être satisfaites. Un exemple pratique qui souligne le besoin de synchronisation est énuméré ci-dessous :
  - Dans les systèmes de bases de données, l'ordre dans lequel les processus effectuent les mises à jour sur une base est important pour garder une vision cohérente et correcte de la base de données. Afin d'assurer le bon ordonnancement des événements, une notion de temps commun entre les processus devient impérative.

Il est assez courant que les applications distribuées et les protocoles réseaux utilisent les timers, et leurs performances dépendent de la façon dont les processeurs physiquement dispersés sont synchronisés dans le temps. La conception de ces applications est simplifiée lorsque les horloges sont synchronisées.

[1] Dans ce type d'architecture l'ensemble du système est entièrement présent sur la machine considérée. Les machines reliées sont vues comme des entités étrangères disposant elle aussi d'un système centralisé. **[15]**

La synchronisation d'horloges a un effet important sur de nombreux domaines comme les systèmes de sécurité, le diagnostic de pannes et de recouvrement, l'ordonnancement des événements, les systèmes de base de données, et les valeurs de l'horloge du monde réel.

Dans ce chapitre, nous citerons quelques définitions qui dépendent de la synchronisation de temps. Ensuite, nous étudierons la synchronisation de temps dans les réseaux traditionnels, et nous verrons que la synchronisation de temps dans les réseaux de capteurs ont de nouvelles contraintes, ceci exige de développer de nouveaux protocoles de synchronisation de temps pour les réseaux de capteurs. A la fin, nous décrirons les différentes familles de protocoles de synchronisation de temps déployés par les réseaux de capteurs et leur classification.

2 Définitions

2.1 Temps et Horloge

-Temps réel et Temps d'horloge : d'après **[45]**, le *temps réel* est le temps supposé newtonien qui ne peut pas être directement un objet observable, par contre le *temps d'horloge* est le temps qui peut être observé sur les horloges.

-Horloge : Les horloges des ordinateurs sont, généralement basées sur les oscillateurs à cristaux qui fournissent une heure locale pour chaque nœud du réseau. Le temps de chaque horloge est un compteur qui s'incrémente avec les oscillateurs à quartz et est dénommé *horloge logicielle*. Le gestionnaire d'interruption doit incrémenter l'horloge logicielle par 1 à chaque fois qu'une interruption se produit. La plupart des oscillateurs matériels ne sont pas si précis parce que la fréquence qui fait augmenter le temps n'est jamais parfaitement correcte. Même une déviation de fréquence de seulement 0,001% entraîne une erreur d'horloge d'environ une seconde par jour **[12]**.

Temps :	Le temps d'une horloge dans une machine p est donné par **la fonction** $\boldsymbol{C_p(t)=\alpha_p t+\beta_p}$ tel que α_p est la fréquence et β_p est l'offset, où $C_p(t)=t$ pour une horloge parfaite ($\alpha_p=1$ et $\beta_p=0$). La comparaison de deux horloges $C_A(t)$ et $C_B(t)$ est donnée par : $C_A(t)= \alpha_{AB} C_B(t)+ \beta_{AB}$.
Fréquence :	La fréquence est **le taux auquel progresse une horloge**, et désigne la vitesse de fonctionnement d'un processeur. La fréquence à l'instant t de l'horloge C_a est : $\boldsymbol{C'_a(t) = \partial C_a(t) / \partial t}$.
Offset :	L'offset d'horloge est **la différence entre le temps reporté par une horloge et le temps réel**. L'offset de l'horloge C_a est donnée par $[C_a(t)– t]$. L'offset de l'horloge C_a par rapport à C_b au temps $t \geq 0$ est donnée par : $\boldsymbol{C_a(t) - C_b(t)}$.
Skew :	Le skew d'une horloge est **la différence entre la fréquence de l'horloge et la fréquence de l'horloge parfaite**. Le skew de l'horloge C_a par rapport à l'horloge C_b à l'instant t est : $\boldsymbol{C'_{ab}(t) = [\partial C_a(t)/\partial t] - [\partial C_b(t)/\partial t]}$. Si le skew ne dépasse pas une valeur maximale σ, alors : $(1- \sigma) \leq (dC/dt) \leq (1+\sigma)$.

Tableau II.1: Terminologie. [15]

Considérant la synchronisation de l'horloge physique dans un système distribué à l'**UTC**[1], l'horloge de l'ordinateur affiche le temps $C(t)$, qui peut ou ne peut pas être identique à t, à un point quelconque du temps réel. Pour une horloge parfaite, la dérivée $[dC(t) / dt]$ devrait être égale à 1.

[1] Universal Time Coordinates : Temps se basant sur l'heure moyenne de Greenwich, ville anglaise par laquelle passe un méridien de référence pour les fuseaux horaires.

Cependant, en réalité la fréquence d'une horloge fluctue avec le temps dû aux changements de la température, de la pression, et de la tension. Ce terme est dénommé *clock skew*. Le tableau (**Tableau II.1)** donne une terminologie, qui est compatible avec les définitions précédentes [**12,13]**.

2.2 Erreur de synchronisation

Pour que les nœuds du réseau puissent être synchronisés, ils doivent posséder pendant une période de temps un canal de communication où le délai de transfert d'un message circulant entre les nœuds peut être mesuré de manière fiable. Cependant, l'ennemi de la synchronisation précise de réseau est le non-déterminisme du procédé d'évaluation du temps. L'estimation du temps de *latence* (ou chemin de temps critique) dans le canal est confondue par les événements aléatoires qui provoquent des retards asymétriques de livraison de messages d'Aller-Retour. Ceci provoque des erreurs de synchronisation. Le temps de latence dans le canal peut être décomposé en quatre éléments distincts **[17]**, comme indiqué dans la **Figure II.1**.

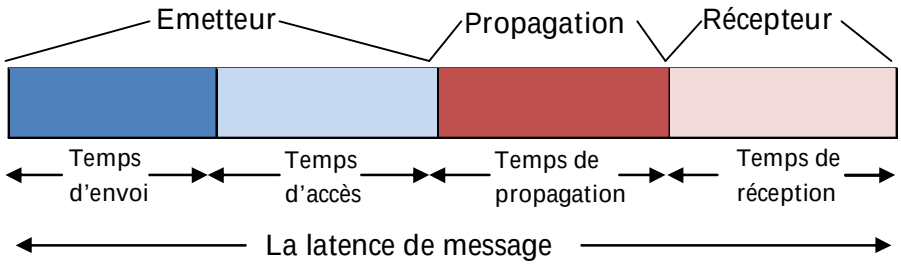


Figure II.1: Composants de retard d'un paquet.

2.2.1 Temps d'envoi

Il s'agit du temps utilisé par l'émetteur pour construire le message. Ce temps inclut les délais de traitement et le temps pour transférer le message à l'interface réseau.

2.2.2 Temps d'accès

Il s'agit du temps d'attente pour accéder au canal de transmission. Ce temps est spécifique au protocole MAC utilisé. La couche MAC du 802.11 utilise un algorithme distribué qui retarde l'accès au canal d'un temps aléatoire borné. De plus, les retransmissions au niveau MAC dues aux problèmes d'interférence ou de collisions font augmenter ce temps d'accès.

2.2.3 Temps de propagation

Il s'agit du temps nécessaire pour que le message transite de l'émetteur au(x) récepteur(s). Le temps de propagation varie, selon l'emplacement de l'émetteur et du récepteur.

2.2.4 Temps de réception

Il s'agit du temps de traitement nécessaire à l'interface réseau du récepteur pour la réception du message et la notification au système de son arrivée. Ceci est typiquement le temps nécessaire à une interface réseau pour générer un signal de réception d'un message.

-Le défi est non seulement l'existence du retard de paquets mais également la difficulté de prévision du temps nécessaire sur chaque composant de retard.

2.3 Méthodes de synchronisation

Il y a trois méthodes de synchronisation **[34]** :

- La première se fonde sur *l'ordre des messages et des événements* ; on le considère le plus simple.
- La deuxième méthode permet aux nœuds de *maintenir l'offset* et *le skew* en ce qui concerne leurs nœuds voisins (*pair à pair*). C'est la méthode de synchronisation la plus utilisée dans les applications de protocoles de synchronisation.
- La troisième est la *synchronisation globale* dans laquelle tous les nœuds sont synchronisés par rapport à un temps global à travers le réseau.

2.4 Précision de synchronisation

Chaque nœud du réseau dispose d'une horloge physique comportant des circuits matériels oscillateur. Malheureusement, la fréquence des horloges des machines varie d'un nœud à un autre. Ainsi, les horloges sur différents nœuds dans les réseaux sans fil fonctionnent à des vitesses différentes. Par conséquence, les valeurs d'horloge utilisées pour la synchronisation dans les réseaux sans fil ne sont pas les lectures physiques d'horloge. Au lieu de cela, les nœuds du réseau généralement emploient une notion logique des horloges et du temps. Les horloges logiques peuvent être modifiées par le logiciel (par exemple, pendant la synchronisation) et par le matériel (par exemple, par les horloges physiques). En conséquence, la précision de synchronisation peut être définie de deux manières.

2.4.1 Précision absolue

C'est l'erreur maximale (c.-à-d., skew et offset) de l'horloge logique d'un nœud par rapport à une norme externe telle que l'UTC.

2.4.2 Précision relative

C'est la déviation maximale (c.-à-d., skew et offset) entre les lectures de l'horloge logique des nœuds appartenant à un réseau.

3 Mécanismes de synchronisation d'horloges pour les réseaux traditionnels

Le problème de la synchronisation a été étudié complètement dans l'Internet et les systèmes informatiques répartis. Il existe plusieurs mécanismes de synchronisation d'horloges pour les réseaux traditionnels (par exemple: GPS, NTP, SNTP,…).

Le protocole de synchronisation de temps le plus couramment utilisé dans les réseaux traditionnels est le NTP (Network Time Protocol) **[14]**. NTP a été conçu pour des réseaux à grande échelle avec une topologie statique (comme internet). Les nœuds sont synchronisés de manière externe avec un temps de référence global qui est injecté dans le réseau dans plusieurs endroits à travers un ensemble de nœuds maîtres (appelés serveurs stratum 1). Ces nœuds maîtres sont généralement synchronisés par le GPS (qui fournit un temps global avec une précision en dessous de 1 μs). Les nœuds dans NTP forment une hiérarchie : les nœuds feuilles sont appelés clients, les autres nœuds sont appelés serveurs stratum L, où L est le niveau du nœud dans la hiérarchie. Chaque nœud a connaissance de ses nœuds pères. Les nœuds échangent fréquemment des messages de synchronisation avec leurs pères et utilisent l'information obtenue pour ajuster leurs horloges en les incrémentant régulièrement.

4 Nouvelles contraintes pour la synchronisation de temps dans les RCFSs

Il serait plus facile si les protocoles (NTP, SNTP, CTP,…etc) peuvent être directement utilisés dans les RCSFs. Ceci est cependant impossible. La plupart des exigences sur lesquelles ces protocoles sont basés ne sont pas satisfaites dans les RCSFs **[45]**.

4.1 Problème d'énergie

Une des préoccupations les plus importantes dans les RCSFs est l'optimisation de la consommation de l'énergie. Pour les protocoles de synchronisation d'horloges dans les réseaux traditionnels filaires, l'utilisation de la puissance CPU pour écouter les messages de signalisation dans le réseau et supporter les transmissions occasionnelles a un impact négligeable sur l'opération d'un nœud de communication. Les protocoles traditionnels sont basés sur les hypothèses suivantes : NTP suppose que le CPU est toujours disponible et qu'il est toujours en écoute du réseau. Ces hypothèses ne sont pas satisfaites dans le cas de RCSFs. Les nœuds de RCSF ont souvent un CPU lent qui fonctionne le plus souvent en mode veille. Quoique les coûts de puissance de traitement diminuent avec le temps, il y a des limites qui les maintiennent relativement couteuses pour des nœuds dans un RCSF. Écouter, recevoir, envoyer dans le réseau est également très couteux en terme d'énergie. Cela pourrait être très difficilement supporté par des nœuds capteurs sans fil qui utilisent généralement des piles, qui ont une durée de vie d'énergie et d'électricité limitée et sont difficiles à remplacer.

4.2 Topologie de réseau Ad hoc

La plupart des protocoles sont basés sur une infrastructure de réseau hiérarchique. Les serveurs au plus haut niveau, souvent qualifiés de serveurs de niveau 1, appelés serveurs maîtres, se synchronisent les uns avec les autres en utilisant des technologies comme le GPS. Un nœud dans cette structure de réseau hiérarchique ne sera éloigné du serveur maître que de quelques sauts seulement. Cependant, dans les RCSFs l'infrastructure hiérarchique est généralement absente. Même si on peut créer une telle hiérarchie, la plupart des nœuds seront plus loin d'elle. Les nœuds qui sont loin du serveur de temps seront mal synchronisés. C'est une mauvaise situation dans les RCSFs, puisque les nœuds qui sont proches les uns des autres nécessitent souvent la plus haute précision de synchronisation.

4.3 Infrastructure dynamique

Les réseaux traditionnels sont basés sur des infrastructures statiques. Pour la synchronisation de temps, les nœuds sont manuellement configurés pour se connecter à certain serveurs de synchronisation de temps (serveurs maîtres). Bien qu'il soit possible d'utiliser des informations statistiques pour décider quel serveur utiliser, les nœuds dépendent toujours de leurs configurations initiales. Une configuration si statique n'est pas possible, en raison de la nature fortement dynamique de RCSFs. De plus, le besoin d'exécution inattendu fait d'une configuration manuelle de différents nœuds fortement indésirable.

4.4 Synchronisation à sens unique

La plupart des protocoles de synchronisation de temps utilisent des méthodes bidirectionnelles, ce qui signifie que les informations sont envoyées en deux directions. Mais, les méthodes bidirectionnelles ne sont pas appropriées aux RCSFs à cause de la mobilité et le déploiement

aléatoire des nœuds capteurs. Si le chemin entre le nœud et le serveur de synchronisation est réciproque (ou symétrique), le délai à sens unique peut être estimé à la moitié du temps d'aller-retour. Dans les RCSFs, ce chemin n'est souvent pas réciproque, ce qui le rend difficile d'estimer le temps. Les méthodes à sens unique pour la synchronisation de temps sont donc meilleurs candidats pour les RCSFs.

4.5 D'autres contraintes

Il existe d'autres défis tel que : *La latence du message, Mobilité, Capacités de calcul limitées, Mémoire limitée, Synchronisation sur des chemins multi-sauts, Bande passante limitée, Impact du choix du matériel, des OS et de la couche MAC*.

-Pendant la conception des algorithmes et des protocoles de réseaux de capteurs, il faut prendre en compte ces ***nouvelles contraintes*** afin de développer de nouvelles applications basées sur les réseaux de capteurs. Ceci exige de développer des protocoles de synchronisation d'horloges pour les réseaux de capteurs.

5 Mécanismes de synchronisation d'horloges pour RCSF

5.1 Classification

Dans cette section, nous classifions les mécanismes de synchronisation de temps dans RCSF :

5.1.1 Synchronisation interne versus Synchronisation externe

Dans **[15]**, les protocoles de synchronisation de temps sont classés comme suit :

-Synchronisation interne : Dans cette approche, une base de temps globale, appelée temps réel, n'est pas disponible au sein du système et l'objectif est de minimiser l'écart maximal entre les horloges locales des capteurs.

–Synchronisation externe : Dans la synchronisation externe, une source de temps standard (temps de référence), comme l'horloge universelle (UTC) est fournie. Les horloges locales des capteurs cherchent à s'ajuster à ce temps référence afin d'être synchronisées. Les protocoles comme NTP **[14]** synchronisent de cette façon, car la synchronisation externe est mieux adaptée aux réseaux à faible couplage, comme Internet. La plupart des protocoles dans les réseaux de capteurs, n'effectuent pas de synchronisation externe sauf si l'application l'exige, parce que l'efficacité énergétique est une préoccupation majeure et employer une source de temps externe induit généralement une haute consommation d'énergie.

-La synchronisation interne conduit généralement à un fonctionnement plus correct du système, tandis que la synchronisation externe est principalement utilisée pour donner aux utilisateurs des informations commodes de référence du temps.

5.1.2 Synchronisation Récepteur-Récepteur versus Synchronisation Emetteur-Récepteur versus Synchronisation Seulement-Récepteur

-Synchronisation Emetteur-Récepteur (SRS) **:** Cette approche traditionnelle se déroule généralement comme suit : dans la **Figure II.2**, le nœud *A* envoie périodiquement un message de synchronisation <sync> avec son temps local T1. Ceci est reçu par *P* au T2, et un paquet d'accusé de réception est envoyé par *P* au T3. Les valeurs T2 et T3 seront incluses dans le paquet d'accusé de réception <ack>. Ce paquet est reçu par *A* au T4. Sachant ces quatre valeurs de temps (T1, T2,

T3 et T4), le nœud *A* peut calculer son offset d'horloge δ relativement au nœud *P* aussi bien que le délai de propagation *d* comme suit **[18]** :

$$\mathbf{d} = \frac{(\mathbf{T2} - \mathbf{T1}) + (\mathbf{T4} - \mathbf{T3})}{\mathbf{2}} \qquad \text{et} \qquad \boldsymbol{\delta} = \frac{(\mathbf{T2} - \mathbf{T1}) - (\mathbf{T4} - \mathbf{T3})}{\mathbf{2}}$$

Où : (T2 = T1 + δ + d) et (T4 = T3 - δ + d).

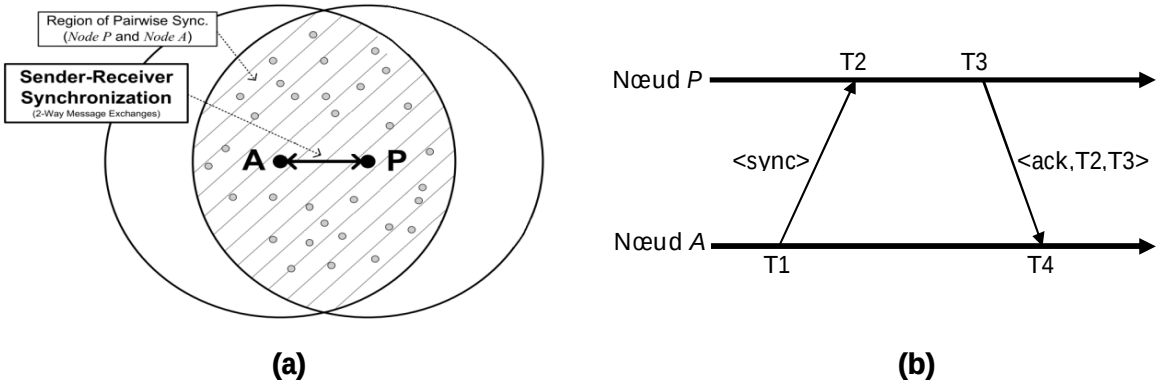


Figure II.2: Synchronisation Emetteur-Récepteur.

L'inconvénient de cette approche est la variation dans le temps de la latence de messages entre l'émetteur et le récepteur.

-Synchronisation Récepteur-Récepteur (RRS) : dans RRS, un message de référence est diffusé (par exemple dans la **Figure II.3**: un message de diffusion est envoyé par le nœud *P*). Si deux récepteurs (ex : *A* et *B*) reçoivent le même message dans une transmission à un-saut, ils le reçoivent à peu près au même moment. Les récepteurs s'échangent le temps dans lequel ils ont reçu le message et calculent leur décalage en se basant sur la différence des temps de réceptions (ex : dans RBS, la différence d'horloges est calculée par la fonction des moindres carrée et la régression linéaire). L'avantage évident est la réduction de la variance du délai de transfert de messages. L'erreur de synchronisation dans cette approche est seulement dépendante du temps de propagation et du temps de réception pour chaque récepteur.

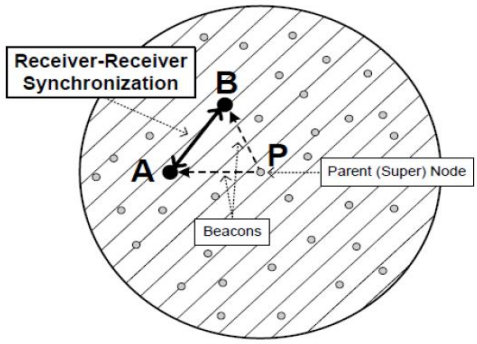


Figure II.3: Synchronisation Récepteur-Récepteur.

-Synchronisation Seulement-Récepteur (ROS) **:** Une nouvelle approche de synchronisation est développée dans **[16],** appelée synchronisation Seulement-Récepteur. Un groupe de nœuds capteurs peuvent être synchronisés seulement en écoutant les messages de synchronisation d'une paire de nœuds. Le schéma proposé dans la synchronisation de diffusion pair-à-pair (PBS **[16]**, voir

la **Figure II.4**) combine efficacement les approches *Emetteur-Récepteur* et *Seulement-Récepteur* pour réaliser la synchronisation du réseau avec un nombre de messages de synchronisation considérablement réduit. Cette nouvelle approche est plus efficace car elle réduit la communication (nombre de messages échangés dans le réseau) et par conséquent la consommation énergétique.

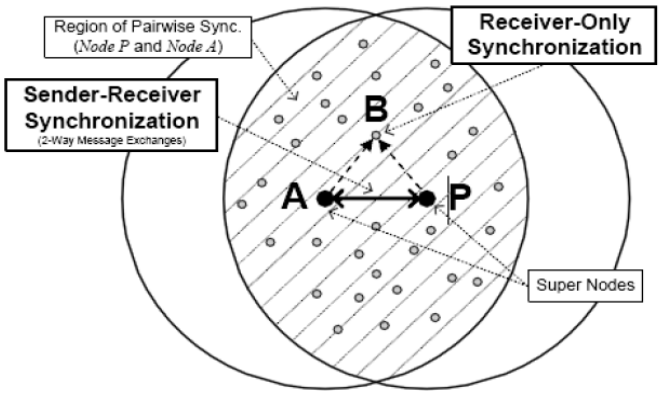


Figure II. 4: Synchronisation Seulement-Récepteur.

5.1.3 Synchronisation Un-Saut versus Synchronisation Multi-Saut

-Synchronisation Un-saut **:** Dans un réseau un-saut, un nœud capteur peut directement communiquer et échanger des messages avec tout autre capteur dans le réseau. Par conséquence, les nœuds capteurs d'un réseau à un-saut peuvent être synchronisés entre eux par trois approches : *synchronisation Emetteur-Récepteur (*ex : TPSN **[18]**), *synchronisation Récepteur-Récepteur (*ex : RBS **[17]**) et *synchronisation Seulement-Récepteur* (PBS **[16]**). Toutefois, de nombreuses applications de réseaux de capteurs sans fil couvrent plusieurs domaines ou voisinages. Le réseau est souvent trop grand, ce qui rend impossible, pour chaque nœud capteur d'échanger directement des messages avec chaque autre nœud.

-Synchronisation Multi-Saut **:** Dans un réseau multi-saut, un nœud capteur peut indirectement communiquer et échanger des messages avec tout autre capteur dans le réseau ; c'est-à-dire la communication peut également se produire comme une séquence de sauts à travers une chaîne de capteurs par paires adjacentes. Par conséquence, la synchronisation multi-saut se fait par plusieurs synchronisations un-saut (SRS, RRS et ROS), comme par exemple la synchronisation d'horloges multi-saut basé-diffusion (FTSP, **[19]**) et la synchronisation d'horloges multi-saut basé-arbre (TPSN, **[18]**).

5.2 Protocoles de synchronisation

Plusieurs protocoles de synchronisation de temps ont été proposés pour les RCSFs afin de réaliser la synchronisation d'horloges. Les protocoles majeurs les plus connus pour la synchronisation de temps dans RCSFs sont RBS **[17]**, TPSN **[18]** et FTSP **[19]**. Dans ce qui suit, nous présenterons en détail ces trois protocoles, ensuite nous citerons quelques autres protocoles.

5.2.1 RBS (Reference-Broadcast Synchronisation)

Un protocole de synchronisation pour les réseaux de capteurs dénommé *Reference-Broadcast Synchronisation* a été proposé par Elson et al. **[17]**. Il est basé sur la synchronisation *Récepteur-Récepteur*. La propriété fondamentale de RBS est qu'un message de diffusion est uniquement utilisé

pour synchroniser un ensemble de récepteurs entre eux, contrairement aux protocoles traditionnels qui synchronisent l'émetteur d'un message avec son récepteur. De cette manière, il supprime le temps d'envoi et le temps d'accès du chemin critique, comme le montre la **Figure II.5**. Où le temps d'envoi et le temps d'accès sont généralement les plus gros contributeurs au non-déterminisme dans la latence. Pour le cas multi-saut, la synchronisation de temps se fait en groupe (*clustering*).

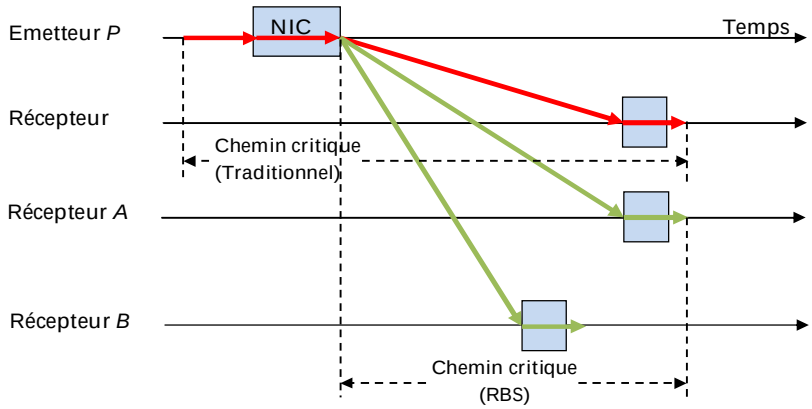


Figure II.5: Analyse du chemin de temps critique pour les protocoles de synchronisation de temps traditionnels et du protocole RBS.

L'avantage de RBS est qu'il élimine le non-déterminisme du côté d'émetteur. L'inconvénient de RBS est les messages échangé (le coût de communication) entre les nœuds récepteurs.

5.2.2 TPSN (Time-Sync Protocol for Sensor Networks)

Le protocole TPSN **[18]** a deux phases. La première phase est la *découverte du niveau* qui permettra de créer un arbre et consiste à assigner à chaque nœud, un niveau dans une hiérarchie, (son père et ses fils). La racine de l'arbre est habituellement une station de base et est assignée à un niveau 0. La deuxième phase est la *synchronisation* qui utilise des échanges de messages bidirectionnels entre les pères et les fils. Les nœuds fils sont synchronisés au père par l'approche *Emetteur-Récepteur*.

TPSN offre de meilleures performances par rapport à RBS par les messages estampillés à la couche MAC de la pile radio qui peut éliminer plusieurs erreurs de synchronisation ; c'est-à-dire éliminer le temps entre la couche application et la couche MAC. Il utilise également un échange de messages dans les deux sens au lieu d'un échange à sens unique comme dans RBS **[17]**. TPSN utilise seulement des liens symétriques pour la synchronisation pair-à-pair entre les nœuds quoique le réseau puisse également avoir des liens asymétriques qui peuvent être considérés comme un inconvénient de TPSN. Un autre inconvénient est la topologie mobile car TPSN se base sur une topologie hiérarchique qui doit être maintenus à chaque changement de la topologie.

5.2.3 FTSP (Flooding Time Synchronization Protocol)

Le but du protocole FTSP **[19]** est d'atteindre une synchronisation d'horloges locales des nœuds dans un réseau étendu en utilisant la *synchronisation Multi-saut* et en se basant sur l'approche *SRS*. Il utilise les messages estampillés à la couche MAC comme TPSN **[18]**. En outre, pour obtenir une haute précision, la compensation pour la différence d'horloge est nécessaire, FTSP utilise la méthode de régression linéaire qui est déjà proposée dans RBS **[17]** et utilise la synchronisation d'horloges multi-saut basé-diffusion ; ceci élimine la phase initiale d'établissement de l'arbre de

TPSN et le rend plus robuste contre les défaillances des nœuds, des liens et les changements dans la topologie du réseau.

5.2.4 D'autres protocoles

***-Synchronisation d'horloges multi-saut basé-diffusion* :** Les protocoles de synchronisation d'horloges basé-diffusion se basent sur le *flooding*[1] (ex : synchronisation d'horloges globale **[20]**) et ils utilisent la *synchronisation multi-saut*. Les nœuds réalisent la synchronisation par l'inondation de leurs voisins avec des informations sur la valeur de leur horloge locale. Après que chaque nœud ait reçu les valeurs d'horloges de tous ses voisins, le nœud peut utiliser ces valeurs d'horloges pour ajuster son horloge (ex : FTSP et RBS). Cette approche de synchronisation est appropriée pour les réseaux de capteurs dynamiques car elle n'utilise pas une topologie hiérarchique.

***-Synchronisation d'horloges multi-saut basé-arbre* :** Le protocole de synchronisation d'horloges basé-arbre se base sur une topologie hiérarchique (ex : TPSN **[18]**) et il utilise la *synchronisation multi-saut* pour synchroniser un réseau étendu. Cette approche de synchronisation est appropriée pour les réseaux de capteurs statiques.

***-Time Diffusion Synchronization Protocol (TDP)* :** TDP est un protocole de synchronisation des réseaux étendus proposé par Su et al. **[21]**. Basé sur l'approche *Récepteur-Récepteur*.

***-Lightweight Tree-Based Synchronization (LTS)* :** L'objectif principal du protocole LTS **[22]** est de réaliser une fiabilité raisonnable tout en utilisant des ressources de calcul modestes (en termes d'espace mémoire et de temps CPU). Basé sur l'approche *Emetteur-Récepteur*.

-Pairwise broadcast synchronization (PBS): PBS **[16]** est basé sur les approches de synchronisation SRS et ROS pour réaliser la synchronisation du réseau à un saut avec un nombre de messages de synchronisation considérablement réduit. Plusieurs approches **[69,70]** ont été proposées pour étendre PBS dans un réseau multi-saut. Cette approche est plus efficace car elle réduit la communication et par conséquent la consommation énergétique.

***-Relative Referenceless Receiver/Receiver Time Synchronization (R⁴Sync)*:** R⁴Sync **[74]** est basé sur l'approche *Récepteur-Récepteur*. Il élimine le besoin d'un nœud référence fixé et la non-synchronisation du nœud référence. L'auteur propose un estimateur offset/skew dans plusieurs cycles. Dans un cycle, les nœuds diffusent un message séquentiellement. Un nœud reçoit *N*-1 messages tel que *N* est le nombre de nœuds où chaque nœud peut communiquer avec les autres nœuds. Chaque message diffusé contient toutes les estampilles de temps de réception des diffusions reçues précédemment, et par conséquent chaque nœud peut calculer la différence d'horloge en appliquant RRS. Donc, R⁴Sync élimine l'échange de temps de réception du message référence entre les nœuds récepteurs.

6 Discussion

Dans cette section, nous présentons une comparaison entre les approches de synchronisation de temps SRS, RRS et ROS. Le **Tableau II.2** présente quatre protocoles et chaque protocole basé sur une des approches de synchronisation ; à savoir : SRS, RRS et ROS. A partir de ce tableau, on remarque que l'approche ROS offre de meilleures performances par rapport aux approches SRS et

[1] Le flooding (technique d'inondation) est une technique classique qui peut être utilisée pour le routage dans les réseaux de capteurs. Dans cette approche, chaque nœud recevant une donnée ou un paquet de contrôle le diffuse à tous les nœuds voisins jusqu'à ce que le nombre maximum de sauts pour ce paquet soit atteint ou le paquet arrive à sa destination.

RRS. L'approche ROS réduit la complexité de messages (le nombre de messages ne dépend pas du nombre de nœuds du réseau), et par conséquent réduit la consommation énergétique dans le réseau.

Protocoles de synchronisation	**Précision de synchronisation**	**Coût de Communisation** **L : nombre de nœuds** **N : nombre de synchronisation**	**SRS vs RRS vs ROS**
RBS	29.13 µs (Estampillage à la couche application) 1.9 µs (Estampillage à la couche MAC)	N+L(L-1)	RRS
TPSN	16.9 µs	2N(L-1)	SRS
FTSP	1.4 µs	NL	SRS
PBS	Même que RBS	2N	ROS

Tableau II. 2: Analyse de performance des approches de synchronisation de temps.

7 Conclusion

Dans ce chapitre, nous avons vu que dans les systèmes distribués, contrairement aux systèmes centralisés, aucune horloge globale et les horloges n'ont ni la même fréquence ni la même vitesse, ce qui provoque une ambiguïté du temps d'où la nécessité d'une ***notion commune de temps*** (la synchronisation de temps). L'opposant de la synchronisation est le non-déterminisme du procédé d'évaluation du chemin critique. Ceci provoque des erreurs de synchronisation. Ensuite, nous avons vu que la synchronisation de temps dans les réseaux de capteurs ont de nouvelles contraintes par rapport aux réseaux traditionnels et par conséquence, les protocoles de synchronisation de temps des réseaux traditionnels ne peuvent pas être directement utilisés dans les réseaux sans fil. Finalement, nous avons étudié les différentes familles de protocoles de synchronisation de temps dans les réseaux de capteurs et les différents mécanismes qu'ils adoptent. Trois approches de base de synchronisation de temps ont été proposées : la synchronisation *Emetteur-Récepteur* (SRS), la synchronisation *Récepteur-Récepteur* (RRS) et la synchronisation *Seulement-Récepteur* (ROS). L'approche ROS offre de meilleures performances (coût de communication et efficacité énergétique) par rapport aux approches SRS et RRS.

Plusieurs applications réseaux de capteurs sans fil exigent que les horloges des nœuds capteurs soient synchronisées. Dans les environnements hostiles, un attaquant peut certainement attaquer le protocole de synchronisation à cause de son importance. Dans les chapitres suivants, on étudiera en détail la synchronisation de temps sécurisée dans les réseaux de capteurs sans-fil.

Chapitre III : Sécurité dans les Réseaux de Capteurs Sans-Fil (RCSF)

1 Introduction

Les réseaux de capteurs sans-fil (RCSF) connaissent actuellement une grande extension et une large utilisation dans différents types d'applications, exigeant une grande sécurité. Nous avons abordé dans le premier chapitre les principales caractéristiques des nœuds capteurs des RCSFs ; une capacité limitée de calcul et de stockage, une communication sans-fil et une ressource énergétique limitée. Ces contraintes font que l'application des mesures classiques de sécurité est restreinte. Ce chapitre traite les problèmes de la sécurité dans les RCSFs qui diffèrent d'autres réseaux en ce qu'ils offrent des restrictions plus sévères en termes de contrainte de RCSF. Nous commencerons donc à étudier les menaces contre les RCSFs. Par la suite, nous étudierons les services de base de la sécurité à respecter pour éviter ces menaces et décrirons les différents mécanismes permettant d'assurer ces services. A la fin, nous présenterons une comparaison entre la cryptographie symétrique et la cryptographie asymétrique dans les RCSFs.

2 Les menaces contre les RCSFs

Une menace étant définie comme l'arrivée potentielle d'événements qui peuvent causer des pertes. Les menaces qui peuvent affecter la sécurité dans les WSNs sont divisées en deux catégories : les mauvais comportements et les attaques.**[75]**

2.1 Les mauvais comportements

On définit un mauvais comportement comme un comportement non autorisé d'un nœud interne (appartenant au réseau) qui peut entraîner involontairement des dommages à d'autres nœuds. C'est-à-dire, ce nœud a d'autres objectifs que de lancer une attaque **[26]**. Par exemple, un nœud refuse de transférer les paquets vers les autres nœuds pour préserver ses ressources.

2.2 Classification des attaques

Les attaques connaissent plusieurs classifications envisageables dont les plus utilisées sont groupées selon les catégories ci-dessous **[23]** :

Selon l'origine (Interne ou Externe)

- *Attaque externe* : elle est déclenchée par un nœud qui n'appartient pas au réseau, ou qui n'a pas la permission d'accès. Par conséquence, l'attaquant externe peut envoyer de fausses informations dans le réseau pour désynchroniser les horloges de nœuds capteurs.
- *Attaque interne* : elle est déclenchée par un nœud interne malicieux. Les attaques internes sont celles dans lesquelles les attaquants qui annoncent de fausses informations à leurs nœuds voisins.

Selon la nature (Passive ou Active)

- *Attaque passive* : elle est déclenchée lorsqu'un nœud non-autorisé obtient un accès à une ressource sans modifier les données ou perturber le fonctionnement du réseau. Une fois l'attaquant ayant acquis suffisamment d'informations, il peut produire un attentat contre le réseau, ce qui transforme l'attaque passive en une attaque active.
- *Attaque active* : elle est déclenchée lorsqu'un nœud non-autorisé obtient un accès à une ressource en apportant des modifications aux données ou en perturbant le bon fonctionnement du réseau.

2.3 Les attaques

Une attaque est un ensemble de techniques informatiques, visant à causer des dommages à un réseau. En effet, les conséquences liées à ces attaques peuvent varier d'une simple écoute du trafic jusqu'à l'arrêt total du réseau selon les capacités des attaquants. Pour les combattre, il est nécessaire de connaître les types d'attaques afin de mettre en œuvre des solutions optimales.

Ecoute (eavesdropping). Elle permet à l'attaquant d'écouter facilement les transmissions pour récupérer le contenu des messages circulant dans le réseau. **[24]**

Déni de service (DoS). Cette attaque vise à saturer le réseau en entier. Pour se faire, elle consiste à provoquer une saturation ou un état instable des nœuds victimes en leur envoyant des données ou des paquets de contrôle de constitution inhabituelle. Néanmoins, aucun protocole existant ne peut survivre à de telles attaques de DoS extrêmes. Une attaque de DoS ne peut pas être surmontée par des techniques cryptographiques. **[24]**

Brouillage (jamming). C'est une attaque de type DoS qui vise les médias de communication utilisés dans les RCSFs. L'attaquant peut émettre un signal d'une fréquence proche de celle utilisée dans le réseau afin de brouiller la communication (l'indisponibilité des canaux de transmission sans fil). Cela empêche les nœuds d'échanger les messages. Aucune défense à une attaque de brouillage (puisque c'est une attaque physique). **[24]**

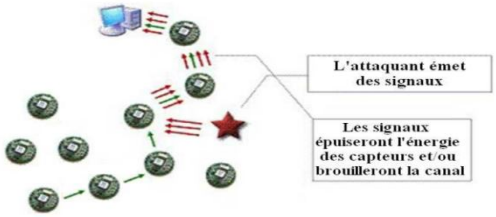


Figure III. 1: Attaque Brouillage.

Rejeu (replay). Dans ce cas, un nœud malicieux surveille et intercepte les messages actuels d'autres nœuds, les enregistre dans la mémoire tampon et les rejoue plus tard. Ce type d'attaque peut habituellement être détecté par l'utilisation du jeton de fraîcheur tel qu'un *numéro de séquence* **[24]**.

Sybil. Un nœud malicieux peut essayer de créer (falsifier) des identités multiples par des attaques Sybil **[25]**. Par exemple, le nœud malicieux peut fabriquer des identités fausses (*identités fabriquées*), ou personnifier d'autres nœuds légitimes dans le réseau (*les identités volées*). L'attaque Sybil peut augmenter le nombre de nœuds malicieux dans le réseau.

Wormhole. Dans une attaque wormhole, un attaquant reçoit des paquets dans un point du réseau, puis les encapsule vers un autre attaquant pour les réintroduire dans le réseau. L'encapsulation permet de cacher les nœuds se trouvant entre les deux attaquants. Donc, les chemins passant par le nœud malicieux apparaissent plus courts et les messages passant à travers ces chemins parcourent une longue distance. Cette technique peut être employée contre les protocoles qui se basent sur la latence des routes. **[24]**

Attaque physique. C'est une attaque qui permet de reprogrammer ou détruire un nœud légitime en accédant au logiciel ou aux matériels qu'il utilise **[24]**.

Pour combattre les différentes menaces, la sécurité informatique se base sur un certain nombre de services qui permettent de mettre en place une réponse appropriée à chaque menace. Ainsi, les principaux services de sécurité sont définis dans la section suivante.

3 Objectifs et services de base de la sécurité

Un réseau de capteurs est un type particulier de réseau. Donc, les exigences de sécurité ne sont pas différentes de celles dans les autres réseaux classiques **[24]**. Mais pose également une seule exigence. Cette exigence est les nouvelles contraintes comme nous l'avons discuté dans le chapitre I.

Les objectifs de la sécurité dans les RCSF ne sont pas différents de ceux dans les autres réseaux classiques. En effet, elle vise à assurer que l'information soit correcte, qu'elle n'ait pas été altérée et émane effectivement de source légitime. La sécurité vise donc à assurer les services de base suivants **[24]**:

La confidentialité des données

La confidentialité des données est la question la plus importante dans la sécurité des réseaux. Chaque réseau avec n'importe quelle focalisation de sécurité abordera typiquement ce problème d'abord. Ce service désigne la garantie que l'information n'a pas été divulguée (propagée) et que les données ne sont compréhensibles que par les entités qui partagent un même secret.

L'intégrité de données

Avec la mise en place de la confidentialité, un adversaire ne peut pas voler l'information. Cependant, ceci ne signifie pas que les données sont sûres. L'adversaire peut changer les données, afin d'envoyer le réseau de capteurs dans le désordre. Ce service permet de vérifier que les données ne subissent aucune altération ou destruction volontaire ou accidentelle, et conservent un format permettant leur utilisation lors de leurs traitements, de leurs conservations ou de leurs transmissions.

La fraîcheur de données

Même si la confidentialité et l'intégrité des données sont assurées, on doit également assurer la fraîcheur de chaque message. Ce service permet de garantir que les données échangées sont actuelles et ne sont pas une réinjection de précédents échanges interceptés par un attaquant.

L'authentification

C'est la propriété qui permet de vérifier que la source de données est bien l'identité prétendue. Un adversaire peut non seulement modifier les paquets de données, mais aussi peut

changer un flux de paquets en injectant des paquets fabriqués. Il est donc essentiel pour un récepteur d'avoir un mécanisme pour vérifier que les paquets reçus sont bien venus à partir du nœud expéditeur réel.

La non-répudiation

Ce service génère, maintient, rend disponible et valide un élément de preuve concernant un événement ou une action revendiquée de façon à résoudre des litiges sur la réalisation ou non de l'événement ou de l'action. C'est donc un mécanisme prévu pour assurer l'impossibilité que la source ou la destination puisse nier avoir émis ou reçu un message, respectivement.

Le contrôle d'accès

Ce service consiste à empêcher des éléments externes d'accéder au réseau, et cela en attribuant aux participants légitimes des droits d'accès afin de discerner les messages provenant des sources internes du réseau de celles externes.

La localisation sécurisée

Souvent, l'utilité d'un réseau de capteurs se fondera sur sa capacité de localiser exactement et automatiquement chaque capteur dans le réseau. Un réseau de capteurs conçu pour détecter des anomalies aura besoin d'informations précises sur l'emplacement afin d'indiquer exactement l'emplacement d'une panne. Malheureusement, un attaquant peut facilement manipuler les informations sur la localisation non-sécurisée.

La disponibilité

Ce service désigne la capacité du réseau à assurer ses services pour maintenir son bon fonctionnement en garantissant aux parties communicantes la présence et l'utilisation de l'information au moment souhaité. Comme les nœuds peuvent jouer le rôle de serveurs, la disponibilité reste difficile à assurer. En effet, un nœud peut ne pas servir des informations pour ne pas épuiser ses ressources d'énergie, de mémoire et de calcul en provoquant ainsi un mauvais comportement.

4 Mécanismes de sécurité pour les RCSFs

Dans cette section, le mécanisme de défense pour combattre les différents types d'attaque sera discuté. Les techniques cryptographiques à clé symétrique et à clé publique sont présentées.

4.1 Cryptographie dans les RCSFs

Plusieurs mécanismes, basés généralement sur la notion de cryptographie, sont mis en place afin de répondre à la question de la sécurité. Le mot « cryptographie » est composé des mots grecques : «crypto» qui signifie caché, «graphy» qui signifie écrire. C'est donc l'art de l'écriture secrète. La cryptographie est l'étude des techniques mathématiques qui permettent d'assurer certains services de sécurité. Elle est définie comme étant une science permettant de convertir des informations "en clair" en informations cryptées (codées) ; c'est à dire non compréhensibles, et puis, à partir de ces informations cryptées, de restituer les informations originales **[27]**.

La cryptographie est réalisée selon certains outils cryptographiques. Avant de les aborder, il est commode de définir la notion de *clé* qui sera utilisée tout au long de cette partie. ***Une clé*** est une

information secrète utilisée en entrée d'une opération cryptographique et qui doit être utilisée avec les algorithmes pour produire le message crypté.

4.2 Les outils cryptographique

Dans cette section, nous définissons les techniques cryptographiques à clé symétrique et à clé publique :

4.2.1 Cryptographie à clé symétrique

Dans la cryptographie à clé symétrique une même clé est utilisée entre deux nœuds communicants pour chiffrer et déchiffrer les données en utilisant un algorithme de chiffrement symétrique. Les algorithmes de chiffrement symétriques sont décomposés en deux catégories:

- Le chiffrement en chaîne (à flot) est fait bit à bit. L'algorithme le plus connu est : RC4 (Rivest Cipher 4), **[28]**.
- Le chiffrement par bloc consiste à fractionner les données en blocs de taille fixe (64 bits, 128 bits). Chaque bloc sera ensuite chiffré une fois qu'il atteint la taille envisagée. Les algorithmes les plus utilisés sont: DES (Data Encryption Standard), AES (Advanced Encryption Standard), **[29]**.

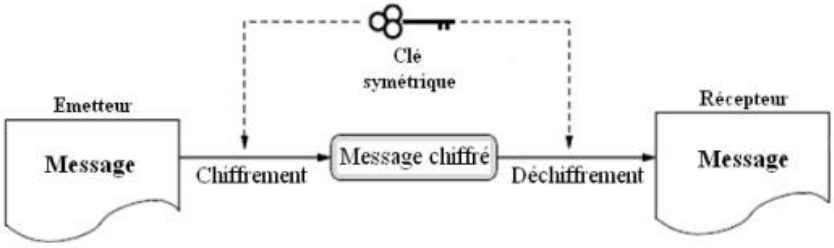


Figure III. 2: Le chiffrement symétrique.

4.2.2 Cryptographie à clé publique

Dans la cryptographie à clé publique, aussi appelée cryptographie asymétrique, deux clés différentes sont générées par un nœud: une *clé publique* diffusée à tous les autres nœuds et une *clé privée* maintenue secrète. L'algorithme de chiffrement asymétrique le plus connu est RSA (Rivest Shamir Adleman) **[29]**.

Figure III. 3: Le chiffrement asymétrique.

4.2.3 La signature digitale

La signature digitale est un système cryptographique assurant la non-répudiation de la source et l'intégrité. Elle repose sur les clés asymétriques. Un émetteur signe les données à transmettre avec sa clé privée en produisant une signature digitale. Cette dernière est par la suite envoyée avec les données. Si elle peut être déchiffrée avec la clé publique par le récepteur et si son résultat est

identique aux données reçues alors la signature est valide, c'est-à-dire, les données proviennent bien de leur émetteur légitime qui ne pourra pas nier l'émission de ces données dans le futur.

4.2.4 Les fonctions de hachage

C'est le mécanisme qui assure l'intégrité de données. Cette fonction calcule une courte empreinte de taille fixe à partir d'une donnée de taille arbitraire. Etant donnée une fonction de hachage *f*, et un message à transmettre *m*. La fonction *f* doit remplir ces conditions:

- Il est facile de calculer *f*(*m*), c'est-à-dire, de calculer l'empreinte à partir du contenu du message.
- Il est difficile de calculer *m* tel que *f*(*m*) = *f*, c'est-à-dire, de trouver le contenu du message à partir de l'empreinte. C'est pourquoi la fonction *f* est dite « à sens unique ».
- Il est difficile de trouver un autre message *m2* tel que *f*(*m*) = *f*(*m2*), c'est-à-dire, il est difficile de trouver deux messages aléatoires qui donnent la même empreinte et cela mène à la résistance aux collisions.

Quelques exemples des fonctions de hachage les plus courantes sont: MD5 (Message Digest 5), SHA-1 (Secure Hash Algorithm), MMH (Multilinear Modular Hash), Poly32. **[24]**

4.2.5 Le code d'authentification de messages (MAC)

Le code d'authentification de message MAC **[24]** (Message Authentification Code) fait partie des fonctions de hachage à clé symétrique assurant l'intégrité de données (code inclus dans un message et permettant de vérifier qu'il n'a pas été modifié pendant son transfert) comme toute autre fonction de hachage, en plus, l'authenticité de la source de données. Dans la pratique, HMAC (Keyed-Hash Message Authentification Code) est utilisé, par exemple : HMAC-SHA-1, HMAC-MMH, HMAC-Ploly32.

4.3 Gestion de clés

La gestion des clés est l'un des aspects les plus difficiles de la configuration d'un système cryptographique de sécurité. Pour qu'un tel système fonctionne et soit sécurisé, chacun des utilisateurs doit disposer d'un ensemble de clés secrètes (dans un système à clés secrètes) ou de paire de clés publiques/privés (dans un système à clés publiques). Cela implique de générer les clés et de les distribuer de manière sécurisée aux utilisateurs ou d'offrir à l'utilisateur le moyen de les générer. Il doit aussi pouvoir enregistrer et gérer ses clés publiques et privées de manière sûre. Dans les systèmes à clés publiques, la gestion des clés comprend la capacité à vérifier et à gérer les clés publiques des autres utilisateurs qui sont signées sous formes de certificats numériques.

Figure III. 4: Fonction de gestion de clés.

5 Cryptographie symétrique vs Cryptographie asymétrique

La cryptographie à clé publique, est une solution très attirante qui fournit des mécanismes plus sûrs et fiables pour l'authentification et la distribution des clés. Traditionnellement, la cryptographie asymétrique exige un espace mémoire assez grand et de haute capacité de calcul, ce qui la rend inappropriée pour les RCSFs. Cependant, des recherches récentes **[30, 31]** ont montré qu'il est possible d'appliquer la cryptographie à clé publique aux réseaux de capteurs en choisissant les bons algorithmes et les paramètres appropriés. Quelques autres travaux **[32, 33]** se concentraient sur la mise en œuvre de l'ECC dans les RCSFs.

Dans les systèmes à clés publiques, l'échange de clé est fortement simplifié. Chaque partie communicante publie sa clé publique. Les clés publiques sont habituellement distribuées en utilisant des certificats numériques, utilisés par le destinataire pour authentifier la clé publique reçue ; toutes les communications avec cette partie seront alors cryptées avec cette clé. L'avantage principal d'utiliser des algorithmes à clés publiques est la facilité de gestion des clés et leur fiabilité. Les inconvénients de cette approche incluent la consommation d'énergie due au calcul des algorithmes à clé publiques et la consommation d'énergie due à la transmission des certificats.

Bien que la cryptographie à clé publique comporte des avantages certains par rapport à la cryptographie à clé symétrique et malgré les recherches qui visent à les appliquer aux RCSFs, la cryptographie à clé symétrique possède ses propres qualités qui la rend toujours la plus préférée pour les RCSF. Pour cette raison la plupart des schémas de gestion de clés proposés pour les RCSFs sont basés sur la cryptographie symétrique. Cependant, l'échange de clés dans les systèmes à clés symétriques est beaucoup plus compliqué. La solution commune est d'utiliser une méthode de pré-distribution, dans laquelle les clés sont chargées dans les nœuds capteurs avant le déploiement. En outre, Le problème majeur avec la cryptographie symétrique est de pouvoir trouver une méthode qui facilite l'établissement des clés entre les nœuds.

6 Conclusion

Dans ce chapitre, nous avons examiné les questions les plus importantes de la sécurité dans les RCSF ; à savoir : les attaques, les services de base, ainsi que les mécanismes de prévention les plus utilisés qui incitent de développer un grand nombre de travaux de recherche. Nous avons vu que la cryptographie à clé publique offre une sécurité plus robuste et fiable par rapport la cryptographie à clé symétrique. Malgré les recherches qui visent à appliquer la cryptographie symétrique aux RCSFs, la cryptographie symétrique possède ses propres qualités qui la rend toujours la plus préférée pour les RCSFs.

Nous avons vu dans le chapitre II que la synchronisation de temps dans les RCSFs est un point très important. Donc, Elle exige un mécanisme de sécurité que nous détaillons dans le chapitre suivant.

Chapitre IV : Sécurité de Synchronisation de Temps dans les RCSF

1 Introduction

La synchronisation de temps est essentielle pour une exécution correcte d'un réseau de capteurs. Plusieurs protocoles de synchronisation de temps ont été proposés pour les réseaux de capteurs sans fil dans les environnements bénis (l'absence des attaques dans l'environnement). Cependant, Toutes les techniques de synchronisation de temps dans les réseaux de capteurs ne peuvent pas survivre aux attaques malveillantes dans les environnements hostiles (la présence des attaques dans l'environnement). En outre, tous les protocoles de synchronisation de temps se basent sur des échanges de messages de temps et par conséquence, les attaquants peuvent tromper ces protocoles par la modification, le retardement, et la falsification des messages de synchronisation de temps. Récemment, plusieurs approches de sécurité ont été proposées pour la synchronisation de temps dans les réseaux de capteurs sans-fil.

Dans ce chapitre, nous présenterons les différentes attaques possibles contre la synchronisation de temps, ensuite les attaques possibles dans quelques protocoles de synchronisation (RBS et TPSN). Après, nous présenterons les effets d'attaquer la synchronisation de temps dans quelques applications de réseaux de capteurs. A la fin, nous décrirons les différentes approches proposées récemment pour sécuriser la synchronisation de temps dans les réseaux de capteurs.

2 Définitions

2.1 Nœud malicieux

D'après **[42]**, le but d'un nœud malicieux (attaquant) est d'injecter une fausse information de synchronisation de temps dans le réseau, sans être détectée par les nœuds *honnêtes*. L'objectif de l'information incorrecte est de calculer une différence de temps incorrecte qui mène à des horloges de nœuds capteurs non-synchronisées dans le réseau.

2.2 Nœud compromis

D'après **[42]**, un nœud compromis est un nœud qui a été pris (occupé) par l'attaquant ; c'est-à-dire, un nœud normal qui met à jour son horloge par une information de temps incorrecte d'un nœud malicieux. Il peut tromper ses voisins avec son information de synchronisation incorrecte.

2.3 Nœud honnête

D'après **[45]**, un nœud capteur est honnête (ou normal) s'il exécute correctement un algorithme donné de la synchronisation ; sinon, il est considère comme un nœud *malicieux*.

3 Les attaques contre la synchronisation de temps dans RCSFs

D'après **[45]**, les types d'attaques qui peuvent toucher la synchronisation de temps sont :

3.1 Attaque pulse-delay

Un nœud malicieux peut déformer la synchronisation de temps. Il brouille le signal entre deux nœuds normaux et puis rejoue le signal pour introduire l'erreur de synchronisation **[42]**. Même un attaquant externe peut lancer cette attaque. Les techniques cryptographiques ne peuvent pas adresser cette attaque.

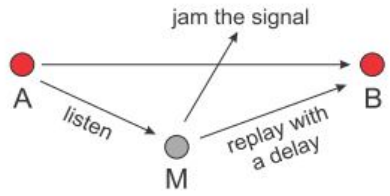


Figure IV. 1: Attaque *pulse-delay.*

La **Figure IV.1** présente l'attaque *pulse-delay*. L'attaquant (M) brouille et rejoue des messages à un temps postérieur, de ce fait retardant leur réception par le récepteur. Par exemple dans l'approche SRS, les équations changeront en : **T2* =T1 + δ + d +**Δ et **T4 = T3 - δ + d**, où **T2*=T2 +** Δ, et Δ est l'erreur introduite par l'attaquant. L'offset δ et le délai de propagation d deviennent alors :

$$\boldsymbol{\delta} = \frac{(\mathbf{T2} - \mathbf{T1}) - (\mathbf{T4} - \mathbf{T3}) + \Delta}{\mathbf{2}} \qquad \text{et} \qquad \mathbf{d} = \frac{(\mathbf{T2} - \mathbf{T1}) + (\mathbf{T4} - \mathbf{T3}) + \Delta}{\mathbf{2}}$$

3.2 Attaque fallacieuse

Dans cette attaque, un nœud référence malicieux peut fournir une fausse information de temps aux nœuds voisins pour calculer une fausse différence d'horloges. Par exemple, dans la synchronisation *RRS* (ex : RBS **[17]**) un nœud compromis peut fournir une fausse information de temps aux nœuds voisins au sujet de la réception du paquet référence. En outre, dans la synchronisation *SRS* (par exemple, TPSN **[18]**), le nœud père peut envoyer une fausse valeur de T2 aux nœuds fils.

3.3 Attaque manipulation de messages

Dans cette attaque, un attaquant peut modifier et falsifier les messages de synchronisation et les envoyer aux nœuds voisins **[50]**.

3.4 Attaque rejeu (replay)

Dans ce cas, un nœud malicieux surveille et intercepte les messages de synchronisation actuels d'autres nœuds, les enregistre dans la mémoire tampon et les rejoue plus tard dans un autre tour de synchronisation.

3.5 Attaque Sybil

Nous avons vu dans le chapitre précédent que l'attaque Sybil peut personnifier les nœuds du réseau. Donc, un attaquant peut personnifier le nœud référence pour désynchroniser les nœuds du réseau.

3.6 Attaque wormhole

Cette attaque peut étendre le chemin d'un message échangé entre deux nœuds, ceci retarde la réception du message de synchronisation. Ce retard permet de calculer une fausse différence d'horloges comme on a vu pour l'attaque delay.

3.7 Attaque silencieuse

Plusieurs protocoles de synchronisation (par exemple, **[18, 22]**) ont besoin d'établir *une hiérarchie de niveau* ou *un arbre* à l'avance pour réaliser la synchronisation de temps dans un réseau étendu. Un nœud malicieux peut avoir un comportement normal dans la phase de découverte de niveau, mais reste silencieux dans la phase de synchronisation. Donc, les nœuds qui se synchronisent au nœud source par l'intermédiaire du nœud malicieux, ne peuvent pas recevoir les messages de synchronisation et ainsi ne peuvent pas mettre à jour leurs horloges, et ils restent non synchronisés.

4 Attaques possibles sur les protocoles de synchronisation de temps

Dans cette section, nous décrirons les différentes attaques possibles sur le protocole de synchronisation de temps RBS qui utilise l'approche *Récepteur-Récepteur* (RRS), et le protocole TSPN qui utilise l'approche *Emetteur-Récepteur* (SRS).

4.1 Attaques possibles sur RBS

Dans RBS **[17]**, un nœud récepteurs *i* peut exécuter *l'attaque fallacieuse* en fournissant une fausse information de temps $ti_{(err)}=ti+\Delta$ aux nœuds voisins au sujet de la réception du paquet référence et par conséquent les nœuds voisins calculent un *offset* incorrect $\delta_{err}=\delta+\Delta$ **[34]**.

4.2 Attaques possibles sur TPSN

Dans TPSN [18], les nœuds fils se synchronisent avec leur père par l'approche SRS, le fils envoie un message de synchronisation <sync> (*t1* : le temps d'envoi du message sync et *t2*: le temps de réception du message sync) et le père envoie un message d'accusé de réception <ack> (*t3* : le temps d'envoi du message ack et *t4*: le temps de réception du message ack). Le père peut envoyer une fausse valeur de *t2* aux nœuds fils. En outre, un attaquant externe peut retarder la réception du message de synchronisation $t2_{(err)}=t2+\Delta$ ou la réception du message d'accusé de réception $t4_{(err)}=t4+\Delta$ pour introduire une erreur de synchronisation Δ. Par conséquent le fils calcule un fau offset $\delta_{err}=\delta+\Delta$. Cette erreur Δ sera alors propagée dans les niveaux inferieurs de l'arbre. Un nœud père peut déconnecter un certain nombre de nœuds inclus dans l'arbre en exécutant l'attaque *silencieuse* **[34]**. La façon pour qu'un attaquant puisse compromettre un plus grand nombre de nœuds est de se positionner dans un emplacement plus élevé sur l'arbre.

5 Effets d'attaquer la synchronisation de temps sur des applications RCSFs

Pour motiver cette discussion des attaques de synchronisation, nous décrirons en détail dans cette section les effets d'attaquer la synchronisation de temps sur un ensemble d'applications et services du réseau de capteurs qui dépendent de la synchronisation de temps.

Dans ce qui suit, nous discuterons trois applications des RCSFs qui se basent sur le principe de synchronisation de temps. La première est temps de réveil (*wake-up time*), la deuxième est partage de canal basé-TDMA (*TDMA-based channel sharing*) et la dernière est *détection des ondes acoustiques*.

5.1 Temps de réveil (wake-up time)

Cet exemple concerne le temps de réveil des capteurs. Les capteurs ont des ressources d'énergie limitées. Pour étendre la vie d'un réseau déployé, les nœuds capteurs sont sélectivement mis dans l'état sommeil. Plusieurs approches ont été proposées pour améliorer l'efficacité énergétique par la commutation des nœuds capteurs entre le mode active et le mode sommeil (puissance-économie) **[35]**. L'idée de la méthode est que l'ensemble des nœuds actifs actuel à n'importe quel instant donné forment un réseau connecté (c'est-à-dire les nœuds actifs actuels couvrent toute la zone). Si un attaquant peut modifier l'horloge interne de certains nœuds capteurs, tels que ces nœuds, par exemple, ne se réveillent pas à temps. Par conséquent, certaines zones ne pourraient pas être surveillées par les capteurs pour une certaine durée de temps, donc un intrus ne peut pas être détecté pendant cette durée **[24]**.

5.2 Partage de canal basé-TDMA (TDMA-based channel sharing)

Dans les protocoles qui utilisent le partage de canal basé-TDMA (Time Division Multiple Access), le temps est divisé en intervalles (time-slots) et chaque nœud est assigné un slot pour émettre et recevoir des messages. Les petites différences de temps sur les horloges de différents nœuds peuvent faire une transmission interférée sur des time-slots adjacents (c'est-à-dire deux nœuds échangent les messages sur une même partie d'un time-solt). Par conséquent, ces transmissions entraînent une collision. Les collisions répétées peuvent significativement perturber le réseau **[36]**. Parmi les protocoles qui utilisent l'approche partage de canal basé-TDMA, le protocole FPS (Flexible Power Scheduling) **[37]** et le protocole PEDAMACS **[38]**.

5.3 Détection des ondes acoustiques

La **Figure IV.2** illustre une application **[39]** qui exige la précision de synchronisation. Des ondes acoustiques se produisent par une source sonore distante. Après, les nœuds capteurs doivent estimés l'angle φ comme montre la **Figure IV.2**. Chaque nœud capteur connait sa propre position exacte, et enregistre l'heure d'arrivée de l'événement du son. L'angle φ peut être déterminé lorsque la longueur l et y sont connues, en utilisant le rapport trigonométrique $\varphi = arcsin(l/y)$. La distance l peut être dérivée par des positions connues des capteurs. De même, la distance y peut être dérivée par la relation $y = v * \partial_t$, tel que v=330 m/s (la vitesse du son), et ∂_t est la différence de temps entre deux arrivée de l'événement du son dans deux capteurs adjacents. Si la précision de synchronisation est seulement 500 µs, l'angle φ peut varier entre 0,166 et 0,518. Donc, une petite erreur dans la synchronisation de temps peut mener à des erreurs importantes dans les évaluations.

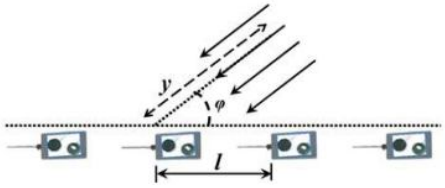


Figure IV. 2: Détection des ondes acoustiques.

6 Solutions proposées récemment pour sécuriser la synchronisation de temps

Dans cette section, nous décrirons les différentes solutions proposées récemment qui permettent de fournir des protocoles de synchronisation de temps sécurisée dans RCSFs. L'objectif de base de ces protocoles est la *sécurité*, *résilience* et *tolérance* aux différentes attaques possibles sur la synchronisation de temps.

- ***Approche de Manzo et al.*[40]**

Manzo et al. ont proposé des contremesures contre les attaques de synchronisation de temps. Ils considèrent un cas spécifique d'un réseau à un-saut où il y a une station de base qui peut couvrir tous les nœuds du réseau. La sécurité consiste à empêcher un nœud malicieux de compromettre la station de base et d'injecter au réseau des informations de synchronisation incorrectes. Pour garantir l'authentification et l'intégrité de messages de synchronisation de temps, ils ont utilisé un système de diffusion authentifié tel que *µTESLA* **[41]** ou des clés secrètes partagées entre l'émetteur et les nœuds récepteurs. Dans le cas où la station de base tombe en panne les nœuds peuvent élire un nœud racine pour que les autres nœuds synchronisent leurs horloges avec ce nouveau nœud racine.

Pour obtenir une haute précision de synchronisation, ils proposent l'utilisation de la *régression linéaire* et *des moindres carrées* qui permettent à calculer une différence d'horloges à partir plusieurs valeurs de différences d'horloges calculées entre les voisins. Dans cette approche, un nœud nécessite plusieurs nœuds voisins pour calculer la différence d'horloge.

- ***Approche de Ganeriwal et al.* [42,43]**

Ganeriwal et al. ont proposé une série de protocoles de synchronisation de temps sécurisée. Ces protocoles sont adaptés à la synchronisation pair-à-pair à un-saut (*Synchronisation Emetteur-Récepteur*). L'idée proposée est présentée par le protocole de synchronisation pair-à-pair sécurisée (SPS) suivant :

Secure Pairwise Synchronization (SPS)

1 $A(T1) \rightarrow (T2)B: \ A, B, N_A, sync$

2 $B(T3) \rightarrow (T4)A: \ B, A, N_A, T2, T3, ack, MAC_{K_{AB}}[B, A, N_A, T2, T3, ack]$

3 A calculates delay $d = \frac{(T2-T1)+(T4-T3)}{2}$

If $d \leq d^*$ then $\delta = \frac{(T2-T1)-(T4-T3)}{2}$

else abort

Ce protocole peut également détecter l'existence d'une attaque pulse-delay en calculant le *délai de propagation (end-to-end delay) d* du message. Si le délai est plus grand qu'un *seuil prédéterminé d**, alors la synchronisation est annulée. Noter que le calcul du délai de propagation *d* vient comme avantage auxiliaire du protocole. Le protocole SPS n'ajoute aucun extra coût de communication sur la synchronisation Emetteur-Récepteur.

Effets d'attaques

L'attaque *Externe*. Les attaques externes peuvent affecter tous les types de protocoles. La défense primaire contre une attaque externe est les *techniques cryptographiques*. Dans ce protocole, l'intégrité de données et l'authentification de messages sont assurées par l'utilisation de code MAC, et d'une clé K_{AB} partagée entre les nœuds A et B, qu'ils peuvent l'utiliser pour signer les messages. Ceci empêche les attaquants externes de modifier avec succès toutes les valeurs dans le paquet de d'accusé de réception (*ack*). En outre, l'attaquant ne peut pas personnifier une identité du nœud B

car il ne tient pas la clé secrète K_{AB}. Pour empêcher un attaquant (attaque rejeu) de rejouer un message d'un round de synchronisation précédent, l'émetteur (A) envoie un nombre aléatoire (nonce) *Na* dans le message *sync*. Le nœud B détecte l'attaque rejeu en comparant le nonce reçu dans le message d'accusé de réception avec le nonce envoyé dans le message de synchronisation (*sync*).

L'attaque *Pulse-delay* et l'attaque *wormhole*. Les attaques potentiellement plus nocives sont l'attaque pulse-delay et l'attaque wormhole. Dans ce protocole, ces attaques sont détectées par une comparaison entre le délai de propagation calculé *d* du message, avec le délai de propagation maximal prévu *d** **[42]**.

L'idée de l'approche du protocole SPS est adaptée à la synchronisation de temps multi-sauts. Ils ont proposé plusieurs protocoles de synchronisation de temps multi-sauts ; à savoir : la synchronisation pair-à-pair à multi-saut sécurisée (Secure Opportunistic Multi-hop synchronization SOM, Secure Direct Multi-hop Synchronisation SDM, Secure Transitive Multi-hop Synchronization STM), la synchronisation de groupe sécurisée (Lightweight Secure Group Synchronization L-SGS, Secure Group Synchronization SGS) et la synchronisation sécurisée dans les réseaux de capteurs étendus **[43]**.

Sun et al.**[46]** montrent que le protocole SPS fonctionne pour les radios de capteurs de bas débit (ex : le capteur MICA2 motes avec un débit de 38,4 Kbps); et il ne peut pas être fonctionné avec les récentes radios (IEEE 802.15.4[1] **[44]** dans les capteurs MICAz et TelosB, avec un débit de 250 Kbps). C'est-à-dire, il n'y a pas assez de temps de produire et d'insérer le code MAC avant la transmission des octets de code MAC dus au délai introduit par le calcul de code MAC. Dans **[46]**, les auteurs proposent une solution pour que SPS fonctionne avec les récentes radios.

- ***Approche de Sun et al.*[45]**

Sun et al. ont proposé une technique statistique pour la synchronisation d'horloges sécurisée et résiliente en présence des nœuds compromis. La technique est appliquée pour la *synchronisation d'horloge basée-niveau* et la *synchronisation d'horloge basée-diffusion* qui réalisent la synchronisation globale dans un réseau entier. Tous les nœuds capteurs synchronisent leurs horloges à une source commune *S*, qui est à son tour synchronisée à l'horloge externe, par exemple, par un récepteur GPS. Un nœud du réseau calcule une différence d'horloge source avec chaque nœud voisin. Donc, il obtient plusieurs différences d'horloges sources. L'idée proposée pour la tolérance aux attaques est : « un nœud calcule correctement la différence d'horloge s'il obtient 2*t*+1 différences d'horloges sources tel que *t* est le nombre maximal des nœuds malicieux parmi les voisins, ensuite il corrige sa propre horloge avec la différence d'horloge qui est la *médiane* des différences d'horloges source».

Cette approche exige qu'un nœud ait un nombre suffisant de voisins honnêtes, chose qui n'est pas facilement garantie en pratique **[34]**.

Effets d'attaques

L'attaque *Silencieuse*. Dans la synchronisation d'horloge basée-niveau, un nœud honnête peut tolérer cette attaque, s'il obtient 3*t*+1 nœuds père, de sorte que même si jusqu'à *t* nœuds malicieux

[1] Une norme de Zigbee : une technologie de connexion par micro-ondes destinée aux périphériques qui ont une source d'énergie faible (exp : un capteur).

restent silencieux, le nœud honnête peut encore recevoir 2*t*+1 différences d'horloges source. Cette approche échoue quand un attaquant peut lancer l'attaque de *brouillage*, puisque les nœuds honnêtes ne peuvent recevoir aucun message de synchronisation.

L'attaque *Wormhole*. Les attaques wormhole peuvent être détectées par leurs *localisations* et / ou les *délais de transmission* de messages **[26 ,49]**.

L'attaque *Sybil*. Cette approche exige une seule authentification basée sur des clés pair-à-pair pour empêcher l'attaque Sybil de personnifier les nœuds honnêtes et envoyer une fausse information de temps aux nœuds voisins. Ces nœuds malicieux peuvent être détectés et les éliminés par la technique proposée dans **[47]**.

- ***Approche de Sun et al.*[46]**

Sun et al. ont proposé un protocole appelé *TinySeRSync* qui permet de fournir une synchronisation de temps sécurisée et résiliente aux attaques dans les réseaux de capteurs. Ils supposent qu'il y a un nœud source *S* qui est synchronisé à l'horloge externe (ex : un récepteur GPS) et tous les nœuds du réseau synchronisent leurs horloges avec le nœud source *S*. *TinySeRSync* réalise la synchronisation de temps globale dans un réseau de capteurs sur deux phases *asynchrones* : *Phase-I synchronisation pair-à-pair à un-saut sécurisée*, et *Phase-II synchronisation globale sécurisée et résiliente*. Dans la première phase, la synchronisation pair-à-pair à un-saut est exécutée. Ils fournissent une solution pour le problème du protocole SPS **[42]**, en proposant une approche *basée-prévision* où l'estampillage de temps authentifié inclut une prévision du temps requis pour calculer le code MAC (spécifiquement destiné à IEEE 802.15.4 avec la radio ChipCon CC2420 **[48]**). Donc, les nœuds réalisent une bonne synchronisation pair-à-pair à un-saut, qui est exploitée dans la deuxième phase pour synchroniser le réseau avec un temps globale *Cs*.

Pour tolérer aux attaques dans la synchronisation globale, les auteurs utilisent l'approche proposée dans **[45]** pour tolérer jusqu'à *t* nœuds voisins malicieux. TinySeRSync utilise *la diffusion locale authentifiée* pour propager des messages de synchronisation, alors que l'approche **[45]** utilise l'unicast authentifié qui nécessite un coût de communication important suivant les indications de l'étude de performance dans **[45]**.

Pour cela, le protocole d'authentification de diffusion *µTESLA* a été utilisé. Dans ce protocole, l'émetteur devise le temps en plusieurs périodes et utilise une chaine de clé unidirectionnel, une clé pour chaque période. La clé utilisée par l'émetteur pour la période actuelle sera divulguée aux récepteurs dans la prochaine période. Les messages reçus doivent être enregistrés par le récepteur et seront authentifiés dans la période suivante. Une application directe du $\mu TESLA$ pour authentifier les messages de synchronisation de diffusion locale conduit à un risque. $\mu TESLA$ est sujet à l'attaque DoS ; un attaquant peut utiliser la clé révélée dans le message pour falsifier les messages de synchronisation. Par conséquence, le récepteur met en mémoire tous les messages falsifiés et donc il n'a pas d'espace pour les autres messages de synchronisation. Pour réduire les menaces de DoS, ils proposent une version modifiée de *µTESLA*, en utilisant un intervalle de temps très court *r* pour limiter la durée vulnérable aux attaques DoS. Mais, l'émetteur doit produire une chaîne de clé assez longue due aux intervalles de temps très courts, et la plupart des clés seront gaspillées. Après, ils proposent deux intervalles différents : un intervalle court *r* et un intervalle long *R,* et les messages de synchronisation sont diffusés seulement dans l'intervalle *r* (voir la **Figure IV.3**).

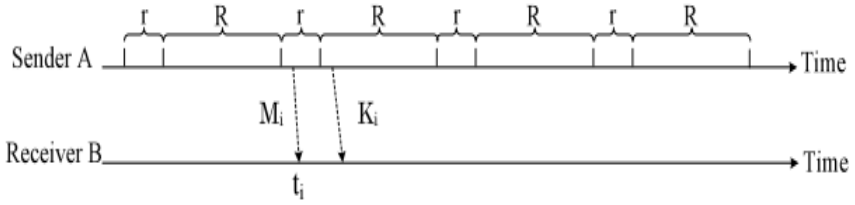


Figure IV. 3: Court délai de *µTESLA*.

D'après la **Figure IV.3**, le récepteur B reçoit un message de synchronisation M_i à t_i, le récepteur B peut calculer $i= |\frac{ti-To}{R+r}|$ et vérifier la condition de sécurité suivante : $\{t_i-T_o+\delta_{AB}+\Delta_{max} < i*(r+R)+r\}$ tel que : Δ_{max} est l'erreur de synchronisation maximale, δ_{AB} est la différence d'horloges pair-à-pair entre A et B, et T_o est le temps de démarrage de $\mu TESLA$ dans A. Si la condition est vérifiée alors B stocke M_i sinon le message est supprimé. Après que B obtienne la clé révélée K_i, il vérifie la validité de la clé, si elle est vrai, B utilise K_i pour vérifier le code MAC inclus dans M_i.

Effets d'attaques

TinySeRSync peut avec succès détecte toutes les attaques externes (*non-DOS*) contre la synchronisation de temps, et est résilient aux nœuds compromis, car il utilise l'approche **[45]** pour l'attaque silencieuse et l'approche **[42]** pour les autres attaques.

- ***Approche de Du et al.*[50]**

Du et al. ont proposé un protocole de synchronisation de temps sécurisée (HBS), efficace pour les réseaux de capteurs hétérogènes (HSN) et se base sur un protocole de gestion de clés publiques **[51]**. HSN se compose d'un petit nombre de capteurs à puissance élevée (H-sensors , exp : PDA) formant des chefs de groupes (*clustrer-head*), et un grand nombre de capteurs à puissance basse (L-sensors, exp : Motes) distribués sur les groupes. La synchronisation se fait en deux étapes : la première est que les capteurs H-sensors sont synchronisés par l'approche *Emetteur-Récepteur* avec une station de base qui est à son tour synchronisée à une horloge source externe (exp: GPS). Pour la sécurité, l'émetteur utilise un *numéro de séquence* du message, et un mécanisme cryptographique à clé publique (il peut exécuter RSA). La seconde est que les capteurs L-sensors synchronisent leurs horloges avec le chef de groupe (H-sensors) par un message de synchronisation diffusé par le chef de groupe. Chaque L-sensor a une clé partagée unique et H-sensor connait toutes les clés partagées. Les clés utilisées pour calculer le code MAC sont différentes pour chaque message de diffusion différent, et ainsi l'attaque rejeu peut être détectée. Le délai de propagation *d* est estimé par une information de localisation, et la différence d'horloge (offset) est calculée par δ=T2-T1-*d* tel que T1 et T2 sont les estampilles d'émission et de réception, respectivement. Les résultats d'expérimentation montrent que HBS a une meilleure précision par rapport à TPSN (HBS : $10{,}6\ \mu s$ et TPSN : 16,9 μs). Puisque le message de diffusion d'un chef de groupe atteint chaque L-sensor dans le groupe à un saut, un attaquant ne pourrait pas lancer l'attaque de manipulation de message. Mais HBS a une limitation pour l'attaque DoS, et les nœuds nécessitent de pré-chargé un grand nombre de clés pour une communication secrète, qui exigent un espace de stockage et un coût de communication importants.

- ***Approche de Song et al.*[52]**

Song et al. ont proposé deux approches de synchronisation de temps résiliente aux attaques dans les RCSFs, et ils utilisent les méthodes cryptographiques pour l'attaques Sybil et un numéro

de séquence de message pour l'attaque rejeu. Ces approches se focalisent sur l'attaque delay qui est détectée par des méthodes statistiques.

La première approche utilise une méthode statistique GESD **[53]** pour détecter les nœuds compromis qui retardent les messages de synchronisation. On suppose qu'un nœud obtient un ensemble de *n* différences de temps $\Gamma=\{\delta_1,\delta_2,\ldots,\delta_n\}$ par le biais d'un protocole de synchronisation de temps, par exemple RBS **[17]** (*Récepteur-Récepteur*), la moyenne de Γ est dénotée δ_{avg} et la déviation standard est dénotée σ. T_i est une valeur correspondante à chaque δ_i tel que $T_i=|\delta_i - \delta_{avg}|/\sigma$ où $i=1,2,\ldots,n$. La valeur de différence δ_j est une différence d'horloges d'un nœud compromis, si sa valeur T_j dépasse une valeur λ (un seuil prédéfinie par l'algorithme GESD, voir l'algorithme détaillé dans **[52]**). Après que l'ensemble Ω des différences de temps des nœuds compromis soit trouvé, la différence de temps est calculée comme suit :

$$Offset = \sum_{i=1}^{n-k}\left(\frac{x_i}{n-k}\right) \quad \text{où } x_i \in (\Gamma - \Omega) \text{ et } k \text{ est la taille de } \Omega.$$

La deuxième approche se base sur un *seuil de différence d'horloge* δ^* (une limite supérieure de différence de temps entre deux nœuds voisins). Supposant qu'il existe une limite inférieure de temps T_{min} qui est nécessaire pour qu'un attaquant compromise un nœud capteur. Le premier round de synchronisation sera exécuté et terminé dans l'intervalle T_{min} ; c'est-à-dire, il n'y a aucune attaque pendant le calcul du seuil δ^* (détail de calcul se trouve dans **[52]**). Si la différence d'horloge est plus grande que ce seuil δ^*, alors la différence d'horloge est supprimée. L'offset est calculé par les différences d'horloges inferieures à δ^*, comme montre la fonction d'offset de l'approche précédente (GESD).

L'approche basée sur un seuil ne nécessite pas de plusieurs nœuds références et le seuil est calculé lors le premier round de synchronisation. Par contre, GESD s'exécute à chaque round de synchronisation et nécessite plusieurs nœuds références.

- ***Approche de Yang et al.*[54]**

Yang et al. ont proposé un mécanisme de sécurité pour le protocole TPSN **[18]** basé sur la défense contre les attaques internes possibles contre ce protocole. Ils supposent que tous les messages de synchronisation sont encryptés et authentifiés pour qu'un nœud malicieux ne puisse pas modifier, lire ou personnifier d'autres nœuds honnêtes. En outre, DAS (Detection Analysis Self-healing) est un mécanisme de défense qui se focalise sur les attaques wormhole et fallacieuse. La structure de mécanisme de défense DAS est montrée sur la **Figure IV.4**.

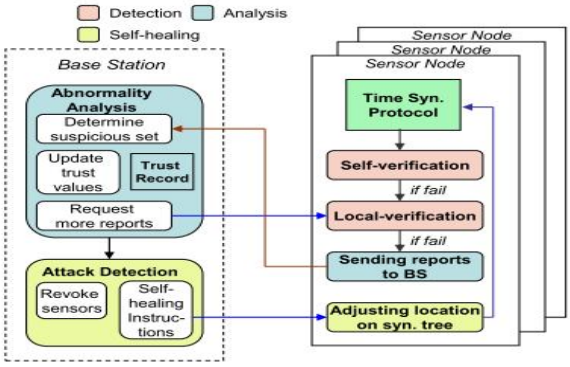


Figure IV. 4: La structure de mécanisme de défense DAS.

La défense peut être divisée en quatre étapes :

-Dans la *première*, après exécution de TPSN, chaque nœud envoie les identités de son père et ses fils vers la station de base, pour que la station de base aura une vue complète de l'arbre de synchronisation.

-Dans la *deuxième*, les nœuds capteurs détectent l'anomalie en deux procédures. La procédure *self-verification* basée sur le délai *d* et la différence δ par la définition de sécurité suivante : $R=1-(1-P_d)(1-P_\delta)(1-P_{skew})$ où P_x est la probabilité, $P_x = 1$ si $x \geq x^*$ sinon $P_x = 0$ où x^* est un seuil prédéterminé. La procédure *local-verification* s'exécute par un nœud *i* comme suit, il diffuse un message de demande de synchronisation à ses voisins. Après, il reçoit plusieurs messages de réponses de synchronisation des nœuds $j_1,\ldots,j_b$, tel que *b* est le nombre de réponses et j_y $(y=1\ldots b)$ est un nœud non-père et non-fils de *i*. Ensuite, *i* calcule $\Gamma=\{\delta ij_1,\delta ij_2,\ldots,\delta ij_b\}$ et vérifié : si $\max(\delta ij_y)>\delta^*$ alors *i* envoie un rapport d'alarme vers la station de base contient $\{j_1,\delta ij_1,j_2,\delta ij_2,\ldots,j_b,\delta ij_b\}$. Ce procédure se déclenche si une des trois conditions est activée : 1) un nœud détecte un risque *R*=1, 2) la procédure *self-verification* toujours généré bas risque et 3) dans le premier round de synchronisation.

-Dans la *troisième*, la station de base analyse les rapports d'alarme et exécute la détection d'un nœud malicieux. A l'itération *k* (*k*^{éme} tour de synchronisation), pour chaque *i* et *j* tel que $\delta ij \in \Gamma$, calculer une valeur de confiance $r_{i,k}$ utilise la technique proposée dans **[55]**, $r_{i,k}=\alpha_{i,k-1}/(\alpha_{i,k-1}+\beta_{i,k-1})$ tel que *i* est bien-comporté (α-1) et mal-comporté (β-1) de temps dans le passé, et $\beta_{i,k}=\beta_{i,k-1}-Neg_i$ (Initialement $\alpha_{i,1}=1$ et $\beta_{i,1}=1$ donc $r_{i,1}=1/2$, $r_{i,k}$ peut être visualisée comme évaluation de la probabilité avec laquelle *i* se comportera bien à l'avenir). Neg_i est la marque négative de *i* calculé par $Neg_i=(-1/T$, tel que T est la taille de l'ensemble responsable contient tous les nœuds des chemins $i \rightarrow$racine et $j \rightarrow$racine.

-La *quatrième* étape basée sur l'analyse dans l'étape 3, la station de base isole les capteurs avec $r_{i,k}$ très basses de confiance telles qu'ils ne peuvent pas propager dans le futur. Cependant, la station de base demande les autres nœuds de changer (ajuster) leur emplacement dans l'arbre de synchronisation.

Ce protocole a deux attaques contre sa défense : 1)- un nœud malicieux peut envoyer un fau rapport et essaye d'isoler les nœuds honnêtes, 2)- L'attaquant peut déclenche plusieurs procédures slocal-verification dans l'ordre pour gaspiller les ressources du réseau.

- ***Approche de Manzo et al.*****[56]**

Manzo et al. ont proposé un protocole de synchronisation de temps tolérant aux attaques appelé ATSP. La première phase est que les nœuds se synchronisent par la diffusion de messages comme FTSP. Donc, un nœud *i* diffuse un message de synchronisation et rassemble des messages de synchronisation de tous ses voisins. Ensuite, *i* calcule un profile de comportement c_{ij} pour chaque voisin *j*. Ce profile se base sur la fréquence relative de nœuds *i* et *j* (calcul détaillé se trouve dans **[56]**). Après *k* itérations, le nœud *i* obtient *k* profile $\{c_{ij}(n)\}^k_{n=1}$ pour *j*. Ensuite, le nœud *i* peut estimer un profile prévu de *j* par l'algorithme RLS[1] **[57]**. Donc, *i* a un comportement prévu pour chaque nœud voisin *j*. A l'itération *k*+1, *i* vérifie si le nouveau profile calculé pour *j* est consistant avec le profile prévu de *j* dans *i*. S'il est consistant, alors *j* est bien-comporté et *i* calcule la différence d'horloges δij. Sinon *j* est mal-comporté, *i* ajoute *j* à la liste-noir, détecte que *j* est un

[1] Recursive Least Square : est un algorithme récursive se base sur la fonction des moindres carrée pour calculé une valeur à partir plusieurs valeurs calculées précédemment.

nœud compromis et le supprime pour le prochain round de synchronisation. Ensuite, *i* mettre à jour son horloge : $C_i(t)=C_i(t)+(\sum \delta ij) / n)$ tel que *j*=1,…,n et *j* n'appartient pas à la liste-noir.

Effets d'attaques

Dans ATSP, chaque nœud dans le réseau peut détecter les attaques qu'ils peuvent modifier l'estampille d'émission ou de réception de messages de synchronisation.

L'attaque Sybil peut augmenter le nombre de nœuds compromis dans le réseau, mais l'hypothèse fondamentale de l'ATSP est que la majorité de nœuds dans le réseau sont bien-comporté car les nœuds ajustent leurs horloges en se basant sur le nombre de voisins. Plusieurs solutions ont été proposées contre l'attaque Sybil tel que : méthode basée-ressource **[58]**, méthode basée-localisation et l'utilisation des clés partagées avec les voisins **[59]**. Ces méthodes peuvent être utilisées dans ce protocole.

- ***Approche de Jadliwala et al.*[60]**

Jadliwala et al. ont proposé une théorie pour la synchronisation de temps sécurisée. Cette théorie formule le problème de synchronisation de temps sécurisée comme un problème de satisfaction de contraintes (CSP) **[61]**, basé-graphe appelé *graphe de différence de temps redondant*. Par conséquent, la solution du problème de synchronisation de temps sécurisée existe si seulement le graphe correspondant est *consistant* (la preuve dans **[60]**). $G(V,E,\delta)$ est un graphe de différence de temps tel que: V est l'ensemble de nœuds, E est l'ensemble des arcs entre les nœuds, δ est la fonction de différence de temps qui donne δ_{ij} de chaque arc (v_i,v_j). Le graphe G est consistant si tous les cycles dans G satisfont la loi de triangle (la somme de différence $\delta_{SUM}=0$, la preuve dans **[62]**). Pour déterminer la consistance, G ne doit pas contenir des boucles acycliques. Si un attaquant fournit une information de temps incorrecte, la différence de temps calculée est incorrecte (utilisé par exemple **[18]**, pour calculé la différence), et le graphe devient inconsistant. Cette inconsistance sera détectée car l'association de l'attaquant avec ses voisins échouera lors de vérification de la loi de triangle.

- ***Approche de Rahman et al.*[63]**

Rahman et al. ont proposé un protocole de synchronisation de temps sécurisée basé sur la cryptographie basée-appariement (PBC) et le chiffrement basé-identité (IBE) dans les deux types de réseaux de capteurs : réseau homogène (WSN ou RCSF) et réseau hétérogène (HSN **[50]**). L'objectif principal de ce protocole après la synchronisation de temps sécurisée est de réduire le *coût de communication* et le *coût de stockage* requis pour chaque nœud. Les nœuds synchronisent leurs horloges en se basant sur un message de synchronisation diffusé par la station de base (*SB*). La communication entre la station de base et les nœuds du réseau est authentifiée. Les nœuds dans le réseau génèrent leur propre clé secrète *K* sur le vol (on-the-fly) en partageant leurs identités *ID* avec les nœuds voisins. L'intégrité de messages est assurée par le code MAC. Chaque message de synchronisation est défini par *un numéro de séquence* différent.

Effets d'attaque

Dans ce protocole, un attaquant ne peut pas calculer les clés secrètes, car la génération de clés est une opération de fonction d'appariement. Même si le nœud malicieux obtient une copie du code MAC précédent, il ne peut pas modifier le message ; comme le code MAC est mettre à jour

(par *sequence#*) chaque fois le message est diffusé. Ainsi, tous les messages de synchronisation sont différents.

La synchronisation à multi-saut consiste à regrouper le réseau multi-saut en groupe (*clustering*, **[65]**) et le synchroniser. Le même processus de synchronisation à un-saut entre la station de base et les nœuds est utilisé pour la synchronisation entre le cluster-head et ses membres. En outre, les nœuds ne sont pas exigés pour être statiques, plutôt ils peuvent être dynamiques car ils produisent dynamiquement les clés secrètes partagées. Finalement, n'importe quel nœud peut se déplacer d'un groupe à l'autre sans coût de communication.

7 Discussion

Dans cette section, nous discutons une comparaison entre les différentes approches de sécurité pour la synchronisation de temps vu précédemment. Ces approches utilisent des méthodes cryptographiques et des méthodes statistiques pour faire face aux différentes attaques. Elles fournissent la sécurité pour les deux approches de synchronisation SRS et RRS, et aucune approche de sécurité n'est fournie pour l'approche ROS.

La plupart des méthodes cryptographiques utilisées sont la cryptographie symétrique. Avec la cryptographie symétrique, un attaquant peut facilement trouver la clé secrète partagée entre les nœuds capteurs. Tandis que dans la cryptographie asymétrique, il est difficile de trouver la clé privée à partir la clé publique. Donc, la cryptographie asymétrique est plus robuste par rapport à la cryptographie symétrique. Les mécanismes cryptographiques permettent d'assurer l'authentification et l'intégrité de données (faire face aux attaques Sybil et manipulation de messages). Cependant, ces mécanismes échouent dans la détection des attaques qui retardent les messages de synchronisation (delay, wormhole et fallacieuse). Pour cela, d'autres méthodes sont utilisées qui permettent d'assurer le service de non-retardement de messages. Ces méthodes sont des méthodes statistiques qui permettent de calculer un seuil de différence d'horloge **[52]**, un seuil de délai de propagation **[43]** ou identifier les fausses valeurs d'offset **[45]** pour tolérer aux attaques qui retardent les messages de synchronisation.

Pour assurer la fraîcheur de données, les solutions proposées utilisent un numéro de séquence ou un nonce pour faire face à l'attaque Rejeu.

Pour les performances, les approches qui utilisent les clés secrètes partagées nécessitent un coût de stockage élevé (ex : HBS) et même un coût de communication. Cependant, il existe d'autres approches pour réduire le coût tel que la diffusion authentifié **[46]** et les méthodes cryptographiques qui calculent les clés secrètes sur-le-vol **[63]**. En outre, les approches de synchronisation de temps sécurisée réussissent à conserver les mêmes performances des approches de synchronisation de temps (ex : SPS a la même précision de TPSN), ou améliorer la précision (ex : HBS a une meilleure précision que TPSN). Le protocole SPS n'ajoute aucun extra coût de communication à l'approche SRS. Pour cela, on constate que lors d'une conception d'un protocole de synchronisation de temps sécurisée, on doit conserver les mêmes performances ou les améliorer.

Le **Tableau IV.1** ci-dessous présente une analyse de sécurité pour les différentes solutions illustrées précédemment.

X : signifie que l'attaque est détectée.

Méthodes de synchronisation de temps sécurisée		Les attaques possibles contre la synchronisation de temps								utilisation des méthodes cryptographiques	utilisation des méthodes statistiques
		Dos	Rejeu	Sybil	Wormhole	delay	Manipulation de Message	Fallacieuse	Silencieuse		
Manzo et al.[40]				X			X			Oui	Oui
Ganeriwal et al. [42,43] (SPS)			X	X	X	X	X	X		Oui	Oui
Sun et al.[45] (2t+1)					X	X		X	X	Non	Oui
Sun et al.[46] (TinySeRSync)			X	X	X	X	X	X	X	Oui	Oui
Du et al.[50] (Protocole HBS)			X	X			X			Oui	Non
Song et al.[52]	Basé-GESD					X		X		Non	Oui
	Basé-seuil δ^*					X		X		Non	Oui
Yang et al.[54] (TPSN sécurisé)					X	X		X		Non	Oui
Monzo et al.[56] (Protocole ATSP)						X		X		Non	Oui
Rahman et al.[63] (Protocole basé PBC et IBE)			X	X	X		X	X		Oui	Non

Tableau IV. 1: Analyse de sécurité.

8 Conclusion

Dans ce chapitre, nous avons vu que les protocoles de synchronisation de temps dans les RCSFs sont une cible commode pour les attaques. Ensuite, nous avons présenté les différentes attaques possibles (pulse-delay, rejeu,…) contre un protocole de synchronisation de temps. Ces attaques conduit à une exécution incorrecte des applications RCSFs, ceci exige de développer de nouveaux protocoles pour sécuriser la synchronisation de temps dans les réseaux de capteurs. En outre, nous avons présenté les différentes approches et techniques proposées récemment pour fournir une synchronisation de temps sécurisée et tolérante aux attaques. Ces approches réalisent la synchronisation de temps sécurisée entre les nœuds du réseau. Ensuite, nous avons comparé toutes les méthodes de synchronisation de temps sécurisée selon leurs performances et leur robustesse contre les différentes attaques possibles sur la synchronisation de temps dans RCSFs.

Les solutions proposées pour sécuriser la synchronisation de temps utilisent des méthodes cryptographiques et des méthodes statistiques pour faire face aux différentes attaques. Dans la cryptographie, la cryptographie asymétrique est plus robuste par rapport à la cryptographie symétrique. En outre, les approches proposées pour la synchronisation de temps sécurisée se focalisent sur les deux approches de synchronisation SRS et RRS, et aucune approche de sécurité n'est fournie pour l'approche ROS.

Dans le chapitre suivant, nous verrons en détail notre proposition qui sécurise l'approche ROS.

Chapitre V : Approche Proposée pour la Synchronisation de Temps Sécurisée dans RCSF

1 Introduction

Ce chapitre décrit notre proposition pour sécuriser la synchronisation de temps dans les réseaux de capteurs sans fils. Nous avons vu dans le chapitre I que les réseaux de capteurs sans fils ont des contraintes (batterie faible, puissance limitée,.etc.) pour leur conception. Dans le chapitre II, nous avons vu que l'approche ROS offre de meilleures performances parmi les approches de synchronisation existantes (SRS et RRS). Pour la sécurité, il existe deux mécanismes cryptographiques : la cryptographie à clé symétrique et la cryptographie à clé publique. La cryptographie à clé publique est plus robuste que la cryptographie à clé symétrique comme on a vu dans le chapitre III. Cependant, la majorité des approches de sécurité proposées pour la synchronisation de temps utilisent la cryptographie à clé symétrique (voir le chapitre IV), et elles concentrent sur les approches SRS et RRS. Tandis que pour l'approche ROS, aucune approche ne sécurise elle. En outre, les méthodes cryptographiques ne peuvent pas adresser l'attaque delay, donc il existe d'autres méthodes qui sont les méthodes statistiques.

Dans les sections suivantes, nous verrons en détail notre approche de synchronisation de temps sécurisée dans les RCSF. Pour cela, nous définissons notre modèle du système de synchronisation et les objectifs visés par la sécurisation. Ensuite, nous montrons les mécanismes de sécurité utilisés et à la fin une description détaillée de notre protocole est présentée.

2 Modèle du système

Dans cette section, nous définissons notre modèle du système que nous appelons *PBS basé-offset*. Dans le modèle skew / offset du PBS **[16]**, la différence d'horloges est calculée dans plusieurs rounds de synchronisation. Tandis que pour notre modèle, la différence d'horloges est calculée dans un seul round de synchronisation. Le principe (voir la **Figure V.1**) est quand deux nœuds (le *nœud A* et le *nœud P*) échangent des messages de synchronisation par l'approche de synchronisation SRS, d'autres nœuds (le *nœud B*) dans la région hachurée peuvent être synchronisés en écoutant les messages de synchronisation entre *A* et *P* (l'approche ROS). Ici, $T1^{(A)}$ et $T4^{(A)}$ représentent les temps mesurés dans l'horloge locale du nœud *A*, $T2^{(P)}$ et $T3^{(P)}$ représentent les temps mesurés dans l'horloge locale du nœud *P*. De même, $T2^{(B)}$ représente le temps mesuré dans l'horloge locale du nœud *B*.

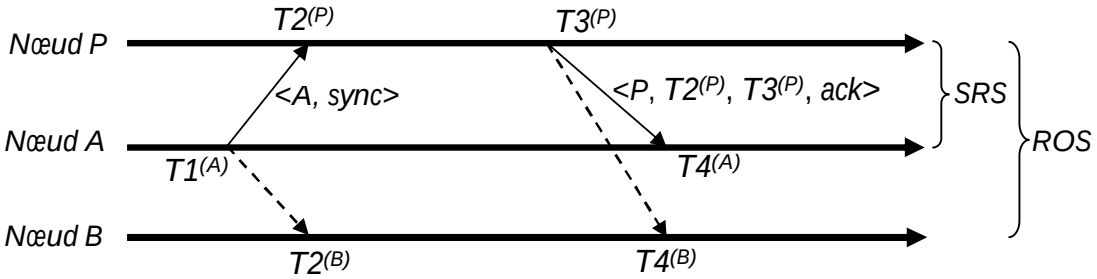


Figure V.1 : PBS basé-offset.

Estampillage à la couche MAC

L'estampillage à la couche MAC peut éliminer plusieurs erreurs de synchronisation de temps comme observé dans **[18,19]**. Dans notre synchronisation de temps, toutes les estampilles de temps sont lues à la couche MAC pour obtenir une meilleure précision de synchronisation. Théoriquement, notre PBS basé-offset donne une haute précision par rapport au PBS **[16]**. Nous verrons dans le chapitre suivant que pratiquement PBS basé-offset a une précision élevée que PBS.

Détail du PBS basé-offset

A l'instant $T1^{(A)}$, le nœud *A* envoie un message de synchronisation « *sync* » vers le nœud *P*. Le nœud *P* et le nœud *B* reçoivent le message de synchronisation aux instants $T2^{(P)}$ et $T2^{(B)}$, respectivement. Les valeurs $T2^{(P)}$ et $T2^{(B)}$ peuvent être représentées par :

$$T2^{(P)} = T1^{(A)} + \delta_{AP} + d_{AP} \quad \text{.....} \quad (1)$$

$$T2^{(B)} = T1^{(A)} + \delta_{AB} + d_{AB} \quad \text{.....} \quad (2)$$

Où δ_{AP} et δ_{AB} sont les différences d'horloges entre le nœud *A* et le nœud *P*, et entre le nœud *A* et le nœud *B*, respectivement. d_{AP} et d_{AB} sont les délais de propagation entre le nœud *A* et le nœud *P*, et le nœud *A* et le nœud *B*, respectivement. A l'instant, $T3^{(P)}$, le nœud *P* envoie un message d'accusé de réception « *ack* » qui contient les valeurs $T2^{(P)}$ et $T3^{(P)}$. Le nœud *A* et le nœud *B* reçoivent le message d'accusé de réception aux instants $T4^{(A)}$ et $T4^{(B)}$. Les valeurs $T4^{(A)}$ et $T4^{(B)}$ peuvent être représentées par :

$$T4^{(A)} = T3^{(P)} + \delta_{PA} + d_{PA} \quad \text{.....} \quad (3)$$

$$T4^{(B)} = T3^{(P)} + \delta_{PB} + d_{PB} \quad \text{.....} \quad (4)$$

Où δ_{PA} et δ_{PB} sont les différences d'horloges entre le nœud *P* et le nœud *A*, et entre le nœud *P* et le nœud *B*, respectivement. d_{PA} et d_{AB} sont les délais de propagation entre le nœud *P* et le nœud *A*, et entre le nœud *P* et le nœud *B*, respectivement. De (1) et (3), on trouve:

$$d_{AP} = T2^{(P)} - T1^{(A)} - \delta_{AP} \quad \text{.....} \quad (5)$$

$$d_{PA} = T4^{(A)} - T3^{(P)} - \delta_{PA} \quad \text{.....} \quad (6)$$

Où $\delta_{AP} = -\delta_{PA}$. Supposant $d = d_{AP} = d_{PA}$ (c'est-à-dire que les délais de propagation d_{AP} et d_{PA} sont symétriques), le nœud *A* peut maintenant calculer la différence d'horloge δ_{AP} et le délai de propagation *d* par la soustraction de (5) et (6) :

$$d = \frac{(T2^{(P)} - T1^{(A)}) + (T4^{(A)} - T3^{(P)})}{2} \quad \text{..... (7)}$$

$$\delta_{AP} = \frac{(T2^{(P)} - T1^{(A)}) - (T4^{(A)} - T3^{(P)})}{2} \quad \text{..... (8)}$$

Par conséquent, le nœud *A* synchronise sa propre horloge avec l'horloge du nœud *P* :

$C_A = C_A + \delta_{AP}$ / C_A est l'horloge locale du nœud *A*

Ainsi, le nœud *B* peut être synchronisé avec le nœud *P* en appliquant la méthode similaire de RRS avec aucun coût de communication. La soustraction des équations (1) et (2) donne :

$$T2^{(P)} - T2^{(B)} = \delta_{BP} + d_{AP} - d_{AB} \quad \text{..... (9)}$$

Supposant d_{AP} et d_{AB} sont symétriques, l'équation (9) permet de calculer la différence d'horloge δ_{BP}:

$$\delta_{BP} = T2^{(P)} - T2^{(B)} \quad \text{..... (10)}$$

Par conséquent, le nœud *B* synchronise sa propre horloge avec l'horloge du nœud *P* :

$C_B = C_B + \delta_{AB}$ / C_B est l'horloge locale du nœud *B*

3 Modèle d'attaques

Dans les environnements hostiles (en présence d'attaque), un attaquant peut perturber le processus de synchronisation de temps, soit par des *attaques externes* comme l'attaque Sybil, l'attaque manipulation de message, l'attaque rejeu et l'attaque pulse-delay, ou des *attaques internes*, comme l'attaque wormhole et l'attaque fallacieuse. Dans cette section, nous démontrons comment ces attaques peuvent interrompre notre modèle du système, c'est-à-dire décrire comment chaque attaque peut perturber la synchronisation du modèle PBS basé-offset (voir la **Figure V.17**).

L'attaque *Sybil*. Elle permet à l'attaquant de personnifier les nœuds honnêtes pour envoyer des fausses informations de temps. Un attaquant *E* peut personnifier le nœud *P* et envoyer une fausse valeur de $T2^{(P)}$. Ceci permet de calculer une différence d'horloge incorrecte entre le nœud *P* et le nœud *B*, et entre le nœud *P* et le nœud *A*.

L'attaque *Manipulation de messages*. Dans cette attaque, un attaquant *E* peut supprimer, modifier ou falsifier le message d'accusé de réception pour tromper la synchronisation entre le nœud *P* et le nœud *B*, et entre le nœud *P* et le nœud *A*. Par exemple, modifier la valeur $T2^{(P)}$ ou $T3^{(P)}$ du message d'accusé de réception.

L'attaque *Rejeu*. Un attaquant *E* peut surveiller et intercepter un message *sync* ou *ack*, l'enregistre dans la mémoire et le rejoue plus tard dans un autre round de synchronisation. Un attaquant *E* peut rejouer un message *ack* qui a une valeur $T2^{(P)}$ ancienne. Ceci trompe les nœuds *A* et *B* pour être synchronisés avec une fausse valeur d'horloge du nœud *P*.

L'attaque *Pulse-Delay et l'attaque wormhole*. Ces attaques peuvent retarder un message *sync* pour introduire une erreur de synchronisation Δ ; $T2^{(P)*} = (T2^{(P)} + \Delta)$ ou $T2^{(B)*} = (T2^{(B)} + \Delta)$. Le nœud *B* calcule une différence d'horloge incorrecte:

$$\delta_{BP(err)} = T2^{(P)*} - T2^{(B)} = (T2^{(P)} + \Delta) - T2^{(B)} = (T2^{(P)} - T2^{(B)}) + \Delta = \delta + \Delta$$

$$\delta_{BP(err)} = T2^{(P)} - T2^{(B)*} = T2^{(P)} - (T2^{(B)} + \Delta) = (T2^{(P)} - T2^{(B)}) + \Delta = \delta - \Delta$$

L'attaque *Fallacieuse*. Le nœud *P* peut envoyer une fausse valeur de $T2^{(P)*}$:

$$\delta_{BP(err)} = T2^{(P)*} - T2^{(B)} = (T2^{(P)} + \Delta) - T2^{(B)} = (T2^{(P)} - T2^{(B)}) + \Delta = \delta + \Delta$$
$$\delta_{BP(err)} = T2^{(P)*} - T2^{(B)} = (T2^{(P)} - \Delta) - T2^{(B)} = (T2^{(P)} - T2^{(B)}) - \Delta = \delta - \Delta$$

4 Objectifs visés par la sécurisation

L'objectif principal est la sécurisation de notre modèle PBS basé-offset contre les attaques citées dans la section précédente, en prenant en compte les contraintes des RCSF.

La sécurisation doit être en mesure de garantir les services suivants :

- *Offrir un niveau de sécurité acceptable*: Prendre en charge les services de sécurité requis par la synchronisation de temps de notre modèle du système.
- *Faire face aux attaques* : Pallier aux principales attaques étudiées dans le modèle d'attaques.
- *Maintenir les performances du réseau* : Ne pas dégrader les performances du réseau après la sécurisation (coût de communication, précision de synchronisation, etc.).

5 Services de sécurité

Le concept de sécurité des RCSF permet d'assurer les services de base de la sécurité telles que : l'intégrité de données, la fraîcheur de données, etc.

Avant de préparer la conception d'un protocole sécurisé, il faut déterminer l'ensemble de mesures à prendre, et cela dans le but d'atteindre les objectifs visés. Afin de faire face aux attaques décrites précédemment, la synchronisation de temps du PBS basé-offset a besoin de garantir l'intégrité de données, leur fraîcheur et leur non-retardement. Ainsi, nous obtenons un nouveau protocole de synchronisation de temps de diffusion pair-à-pair sécurisé, que nous appelons « SPBS » (*Secure Pairwise Broadcast time Synchronisation*).

5.1 Authentification de sources de messages

Le service d'authentification de la source de messages doit être assuré dans tous les paquets émis sur tous les liens de communication reliant les différentes entités du réseau (*A-P*, *A-B*, *P-A*, *P-B*). Elle est garantie grâce aux clés secrètes et publiques qui seront vues plus tard dans les mécanismes nécessaires pour l'intégrité de données. Par conséquent, une assurance d'authentification de sources de messages permet de faire face à l'attaque « Sybil ».

5.2 Intégrité de messages échangés

Il est primordial d'assurer que les informations de synchronisation de temps transmises n'ont pas été modifiées. Par conséquent, une garantie d'intégrité de données permet de faire face à l'attaque « Manipulation de message ».

5.3 Fraîcheur de données

Notre modèle du système doit garantir que les données présentes échangées sont récentes et ne sont pas une réinjection de précédents échanges interceptés par un attaquant. Dans le cas où un nœud émetteur *P* (nœud référence) envoie une information de temps vers un nœud récepteur *B*, un attaquant *E* peut l'intercepter et la réinjecter. Par conséquent, garantir la fraîcheur de données permet de faire face contre l'attaque « Rejeu ».

5.4 Non-retardement de messages

La synchronisation de temps se base sur les estampilles des messages de synchronisation de temps échangées entre les nœuds capteurs. Ces messages peuvent être retardés et ce retard a un effet important pour la synchronisation de temps. Donc, on a un nouveau service de sécurité spécialement pour la synchronisation de temps qui est le non-retardement de messages. PBS basé-offset doit assurer que les messages de synchronisation de temps ne sont pas retardés par un attaquant. Par conséquent, ce service permet de faire face aux attaques « Delay », « Fallacieuse » et « Wormhole ».

5.5 Confidentialité

Dans plusieurs domaines d'applications des RCSF, les données captées sont sensibles et peuvent être menacées par des évènements extérieurs ou intérieurs, d'où le besoin d'assurer leur confidentialité. En ce qui concerne notre modèle du système, il ne manipule aucune donnée secrète. Par conséquent, on n'a pas besoin d'assurer ce service au niveau de la synchronisation de temps.

6 Mécanismes de sécurité utilisés

Afin de garantir les objectifs de sécurité que nous avons fixés, nous devons avoir recours à un certain nombre de mécanismes qui nous permettent d'implémenter les services de sécurité désirés. Dans cette section, nous définissons les mécanismes de sécurité utilisés pour sécuriser notre modèle PBS basé-offset.

6.1 Mécanismes nécessaires pour l'authentification et l'intégrité de données

Le code d'authentification de message et les signatures digitales, tel que décrit dans le chapitre III, nous permettent d'assurer l'intégrité de données. Pour que ce service soit mis en place, nous devons disposer d'un ensemble de clés entre les différentes entités du réseau, ceci permet d'assurer l'authentification de source de messages. Dans notre approche de sécurité, nous utilisons en premier lieu la cryptographie à clé symétrique. Le problème de la cryptographie symétrique comme on a vu dans le chapitre III, est qu'elle est moins robuste aux attaques. Donc, dans second lieu, nous utilisons pour une autre version de SPBS, la cryptographie à clé publique qui est plus robuste.

Les messages authentifiés sont le message de synchronisation et un message que nous ajoutons au PBS basé-offset. Nous appelons ce message ajouté *message d'estampillage*. Il est envoyé après le message d'accusé de réception. Les valeurs $T2^{(P)}$ et $T3^{(P)}$ ne sont pas envoyées dans le message d'accusé de réception, elles sont envoyées dans le message d'estampillage. Nous avons ajouté un troisième message qui est le message d'estampillage parce qu'il n'y a

pas assez de temps à la couche MAC pour calculer le code d'authentification ou la signature digitale avant la transmission dû au délai introduit par les calculs pour les nouvelles radios (ex : 802.15.4) **[46]**.

6.2 Mécanismes nécessaires pour la fraîcheur de données

Pour assurer la fraîcheur de données dans notre approche, nous utilisons un *nonce* qui est un nombre aléatoire. Les nœuds détectent l'attaque « rejeu » en se basant sur ce nombre. Par la suite, nous définissons les nonces utilisés dans notre approche.

6.3 Mécanismes nécessaires pour le non-retardement

Les messages de synchronisation de temps peuvent être retardés et ce retard a un effet important pour calculer la différence d'horloge. Par exemple, si un attaquant *E* retarde le message de synchronisation « sync » dans PBS basé-offset, alors la valeur de réception du message de synchronisation par le nœud *B* est incorrecte et elle est calculée comme suit :

$$T2^{(B)}_{err} = T2^{(B)} + \Delta = T1^{(A)} + \delta_{AB} + d_{AB} + \Delta = T1^{(A)} + \delta_{AB} + (d_{AB} + \Delta) = T1^{(A)} + \delta_{AB} + d_{AB}^{(err)}$$

Dans **[42]**, les auteurs ont utilisé un seuil de délai de propagation pour faire face contre l'attaque delay. Dans notre approche, nous utilisons deux seuils (un seuil maximal d^*_{max} et un seuil minimal d^*_{min}) pour vérifier si le message de synchronisation « sync » ou le message d'accusé de réception « ack » est retardé ou falsifié. Nous détaillerons l'utilisation de ces seuils dans la section suivante.

7 Description détaillée

Dans cette section, nous donnons le déroulement détaillé du schéma de sécurisation du protocole SPBS. Afin de ne pas s'encombrer de définitions répétées, nous citons les principales notations qui seront utilisées tout au long de la description de ce schéma.

7.1 Notation

Notation	Description
ID_P et ID_A	Identité du nœud *P* et du nœud *A*, respectivement. Cela pour identifier les nœuds *A* et *P*.
FraicheurBol	Variable booléenne, vrai si le message n'est pas réinjecté, faux sinon.
IntégritéBol	Variable booléenne, vrai si l'intégrité est assurée, faux sinon.
Ti	La valeur d'horloge du nœud i lue à la couche MAC.
*	Adresse de diffusion. Cette adresse est utilisée pour envoyer un message vers tous les nœuds voisins.
Générer_Nonce ()	Fonction qui retourne un nonce (nombre aléatoire) pour assurer la fraîcheur des messages échangés entre les nœuds *P*, *A* et *B*.
NP	NP est la valeur du nonce générer par la fonction Générer_Nonce () dans le nœud *P*. Ce nonce est utilisé pour garantir la fraîcheur de message d'estampillage.
NA	NA est la valeur du nonce générer par la fonction Générer_Nonce ()

	dans le nœud *A*. Ce nonce est utilisé pour garantir la fraîcheur de message d'estampillage.
Fraîcheur(x,xx,y,yy)	Fonction qui retourne vrai si le nonce x est égal à xx, et le nonce y est à égal yy. C'est-à-dire : vérifier si la fraîcheur est assurée.
Intégrité(msg)	Le mécanisme cryptographique appliqué sur le message msg pour assurer l'intégrité et l'authenticité des messages.
IntégritéRes	Le paramètre qui assure l'intégrité. Il se génère par Intégrité(msg).
msg_sync	Structure du message de synchronisation avec les champs suivants : id (l'identité), nonceA (nonce) et intégrité qui représente le paramètre de l'intégrité du message de synchronisation. Elle est représentée comme suit : <id, nonceA, intégrité>.
msg_ack	Structure du message d'accusé de réception avec les champs : id (l'identité) et nonceP (nonce). Elle est représentée comme suit : <id, nonceP>.
msg_ts	Structure du message d'estampillage avec les champs : id (l'identité) et nonceP (nonce), les valeurs T2P et T3P estampillées dans *P*, et intégrité qui représente le paramètre de l'intégrité du message d'estampillage. Elle est représentée comme suit: <ID_P, NA, NP, T2P, T3P, intégrité>.
msg.x	x est le champ du message de type msg.
Envoyer(*,msg)	Diffuser un message de type *msg* vers tous les nœuds voisins.
Recevoir(msg)	Recevoir un message de type *msg*.
d1	Délai de propagation entre le nœud *A* et le nœud *P*.
d2	Délai de propagation entre le nœud *B* et le nœud *P*.
CA et CB	L'horloge locale du nœud *A* et du nœud *B*, respectivement.

7.2 PBS basé-offset sécurisé (SPBS)

Après avoir défini les mécanismes nécessaires pour la sécurité et les appliqués au PBS basé-offset, nous obtenons le protocole SPBS présenté dans la **Figure V.2**.

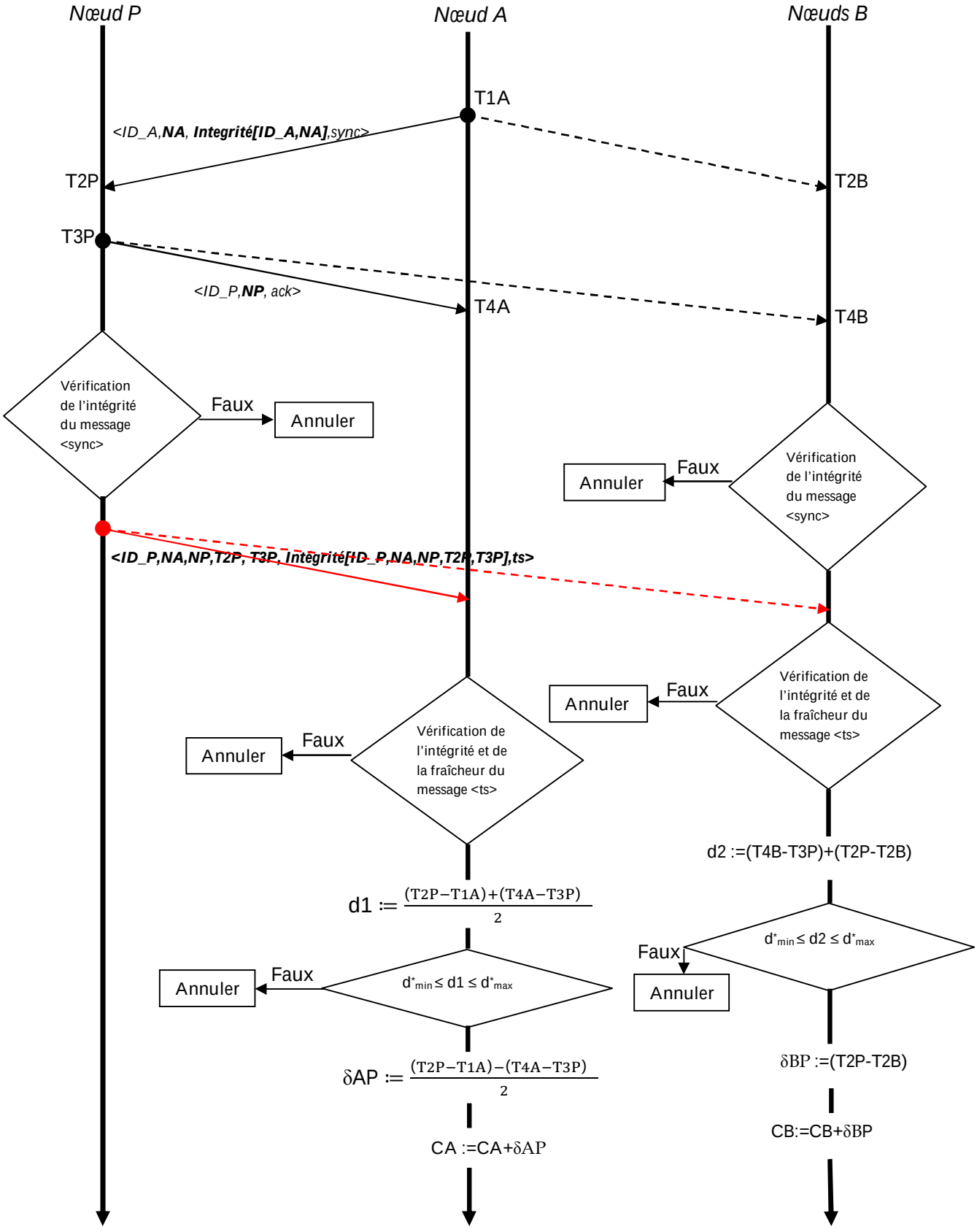


Figure V. 2: Le déroulement du PBS basé-offset sécurisé (SPBS).

7.3 Détail du protocole

L'algorithme suivant montre le déroulement détaillé de notre protocole SPBS :

1. Le nœud A prépare et diffuse un message de synchronisation <sync> vers le nœud *P* et les nœuds *B*:

```
début
  msg_sync.id := ID_A;
  NA := Générer_Nonce() ;
  msg_sync.nonceA := NA ;
  msg_sync.intégrité := Integrité(ID_A, NA);
  Envoyer (*, msg_sync);
  T1A := temps d'émission;
fin
```

2. Réception du message de synchronisation par le nœud *P* et les nœuds *B*, ensuite le nœud *P* diffuse un message d'accusé de réception <ack> et après un message d'estampillage:

```
Recevoir  Msg(msg_sync)
début
  NA := msg_sync.nonceA ;
  IntegritéRes := Integrité(msg_sync.id, msg_sync.nonceA) ;
  si  (ID_P)
      alors
           T2P := temps de réception;
           NP := Générer_Nonce();
           msg_ack.id := ID_P;
           msg_ack.nonceP := NP;
           Envoyer(*, msg_ack);
           T3P := temps d'émission;
          si (msg_sync.integrité = IntegritéRes) alors
              msg_ts.id := ID_P ;
              msg_ts.nonceA := NA ;
              msg_ts.nonceP := NP ;
              msg_ts.estampille2 := T2P ;
              msg_ts.estampille3 := T3P ;
              msg_ts.integrité := Integrité(ID_P,NA,NP,T2P,T3P);
              Envoyer (*, msg_ts);
```

```
            finsi ;
        sinon
              si (msg_sync.integrité = IntégritéRes)
                 alors
                       T2B := temps de réception;
                 finsi ;
        finsi ;
  fin
```

3. **Réception du message d'accusé de réception par le nœud *A* et les nœuds *B* :**

```
  Recevoir  Msg(msg_ack)
  début
       si (ID_A)
          alors
                  T4A := temps de réception;
                  NP := msg_ack.nonceP ;
          sinon
                  T4B := temps de réception;
                  NP := msg_ack.nonceP ;
       finsi ;
   fin
```

4. **Réception du message d'estampillage par le nœud *A* et les nœuds *B* :**

```
  Recevoir  Msg(msg_ts)
  début
    IntégritéBol := FAUX ;
    T2P := msg_ts.estampille2 ;
    T3P := msg_ts.estampille3 ;
    NA_reçu := msg_ts.nonceA ;
    NP_reçu := msg_ts.nonceP ;
    FraicheurBol := Fraîcheur(NA,NA_reçu,NP,NP_reçu) ;
    IntégritéRes := Intégrité(msg_ts.id,NA,NP,T2P,T3P);
    si (IntégritéRes = msg_ts.intégrité)
        alors
```

```
                IntégritéBol := VRAI ;
          finsi ;
     si  (ID_A)
       alors
            d1 := (T2P−T1A)+(T4A−T3P) / 2 ;
          si (FraicheurBol=VRAI et IntégritéBol=VRAI et d*min≤d1≤d*max)
             alors
                        δAP = (T2P−T1A)−(T4A−T3P) / 2 ;
                        CA := CA + δAP ;
             finsi ;
       sinon
             d2 := (T4B – T3P) + (T2P – T2B) ;
          si (FraicheurBol=VRAI et IntégritéBol=VRAI et d*min≤d2≤d*max)
             alors
                       δBP := (T2P – T2B) ;
                       CB := CB + δBP;
             finsi ;
       finsi ;
    fin
```

7.4 Analyse de sécurité

Notre protocole est sécurisé contre les différentes attaques définies précédemment (modèle d'attaque). Donc, nous décrivons comment le protocole SPBS est robuste contre ces attaques, c'est-à-dire comment assurer les services de sécurité suivants (définis précédemment) :

- La fraîcheur de données. Elle est assurée par deux nonces *NA* et *NP*. Lorsque le nœud *A* ou le nœud *B* reçoit le message d'estampillage (*msg_ts*), il vérifie si le nonce (*NA*) envoyé dans le message de synchronisation (*msg_sync*) est égale à nonce reçu correspondant, de même vérifie si le nonce envoyé dans le message d'estampillage est égale à nonce reçu correspondant. Par conséquent, si un attaquant (attaque rejeu) rejoue le message *msg_ts*, les nœuds de synchronisation peuvent le détecter.

- L'intégrité de données et l'authentification. Ces services sont garantis par l'utilisation de la fonction *Intégrité(msg)*. Cette fonction peut être définie par la cryptographie à clé symétrique ou la cryptographie à clé publique. Cela empêche une attaque externe de modifier les valeurs d'estampillage dans les messages. En plus, un attaquant ne peut pas personnifier le nœud *P* ou le nœud *A* parce qu'il ne connaisse pas la clé privée.

Dans la cryptographie à clé symétrique, tous les nœuds (nœud *P*, nœud *A* et les nœuds *B*) partagent la même clé secrète *k* qui est pré-chargée lors du déploiement dans tous les nœuds.

Les messages de synchronisation de temps échangés sont authentifiés par la clé *k*, et chaque nœud qui reçoit ou émet un message calcule le code MAC pour garantir l'intégrité de données. Donc, la fonction *Intégrité*(*msg*) pour la cryptographie à clé symétrique est $MAC_K(msg)$ qui génère un code d'authentification de messages *mac*.

Dans la cryptographie à clé publique, le nœud *A* génèrent lors du déploiement une clé privée SK_A et une clé publique PK_A. De même, le nœud *P* génèrent lors du déploiement une clé privée SK_A et une clé publique PK_A. Ensuite, le nœud *A* et le *P* envoient leur clé publique vers tous les autres nœuds en utilisant une clé secrète *k* pré-chargé dans tous les nœuds lors du déploiement pour authentifier les clés publiques des nœuds *A* et *P*. Tous les messages échangés sont authentifiés par les clés privée et publique, et chaque nœud qui reçoit ou émet un message génère une signature digitale pour garantir l'intégrité de données. Donc, la fonction *Intégrité*(*msg*) pour la cryptographie à clé publique est $ECDSA_{SK}(msg)$ pour l'émetteur et pour le récepteur $ECDSA_{PK}(msg)$, tel que ECDSA est un algorithme de signature digitale à courbe elliptique qui génère une signature digitale *s*.

Nous verrons dans le chapitre suivant les différents résultats obtenus par la cryptographie à clé symétrique et la cryptographie à clé publique.

- Le non-retardement de messages. Pour assurer le non-retardement, nous utilisons deux seuils de délai de propagation, un seuil de délai minimal d^*_{min} et un seuil de délai maximal d^*_{max}. Le seuil minimal vérifie si la valeur d'estampillage est décrémentée ou non par le nœud *P*, cela empêche le nœud *P* de falsifier les nœuds *A* et *B*. Le seuil maximal empêche l'attaque delay et l'attaque wormhole d'introduire un retard dans la synchronisation de temps.

Le nœud *A* utilise le délai de propagation *d1* pour assurer le non-retardement, *d1* est le délai de propagation entre le nœud *A* et le *P*. De même, le nœud *B* utilise le délai de propagation *d2* pour assurer le non-retardement, *d2* est le délai de propagation du message d'accusé de réception envoyé par le nœud *P* et reçu par le nœud *B*.

Estimation du délai de propagation : pour calculer les seuils, nous supposons que le délai de propagation suit une distribution gaussienne $d \sim N(d_{moy}, \sigma)$ **[42]**, ainsi avec une confidence de 99,97%, le délai correcte tombera dans l'intervalle $[d_{moy}-3\sigma, d_{moy}+3\sigma]$. Donc, nous pouvons calculer les seuils comme suit: $d^*_{max} = d_{moy}+3\sigma$ et $d^*_{min} = d_{moy}-3\sigma$. L'estimation de ces seuils dans un environnement réel sera illustrée dans le chapitre suivant.

7.5 Impact maximal de l'attaque

Comme montre la section précédente, avec une probabilité élevée le délai de propagation est entre d^*_{min} et d^*_{max}. Le plus mauvais cas est lorsque le délai de propagation est égal à d^*_{min} ou d^*_{max}. Dans ce cas, un attaquant peut introduire une erreur maximale $\Delta=(d^*_{max}-d^*_{min})=6\sigma$ pour le nœud *B* et une erreur maximale $\Delta=2*6\sigma=12\sigma$ (délai bidirectionnel) pour le nœud *A*.

Le nœud *A* calcule le délai de propagation comme suit :

$$d1_{err}=d1+(12\sigma/2)= d^*_{min}+(12\sigma/2)= d_{moy}-3\sigma+(12\sigma/2)= d_{moy}+3\sigma = d^*_{max}$$

Le nœud *B* calcule le délai de propagation comme suit :

$$d2_{err}=d2+6\sigma = d^*_{min}+6\sigma = d_{moy}-3\sigma+6\sigma = d_{moy}+3\sigma = d^*_{max}$$

8 Conclusion

Dans ce chapitre, nous avons présenté les différents mécanismes qui permettent à l'approche ROS de faire face à la plupart des attaques. Cela nous a permis de concevoir un protocole de synchronisation de temps de diffusion pair-à-pair sécurisé que nous avons appelé SPBS (Secure Pairwise Broadcast Synchronisation). En effet, le but de sécuriser la synchronisation est de proposer un protocole qui assure l'authentification de la source de messages, l'intégrité de données, la fraîcheur de données et le non-retardement de messages. Pour l'intégrité de données, nous avons présenté deux versions de SPBS, une version qui utilise la cryptographie à clé symétrique et l'autre utilise la cryptographie à clé publique. La différence entre eux pour la synchronisation sera montrée dans le chapitre suivant.

Nous allons passer à l'implémentation de toutes les étapes de notre étude et donner des résultats démonstratifs qui les justifient. Le prochain chapitre sera consacré à l'analyse et à la réalisation de notre protocole.

Chapitre VI : Analyse et Réalisation

1 Introduction

Tel qu'on l'a montré au cours de l'étape de conception, l'objectif principal de notre travail est la mise en œuvre d'une solution qui se charge de sécuriser l'approche ROS. Notre premier but est d'atteindre un niveau de sécurité acceptable sans dégrader les performances du réseau. De ce fait, nous avons établi un nouveau protocole SPBS qui est en mesure de faire face aux importantes attaques visant l'approche ROS. L'objectif de ce chapitre est donc de démontrer l'efficacité du protocole SPBS par rapport au protocole de synchronisation de temps (FTSP qui donne une haute précision) et de synchronisation de temps sécurisée (SPS qui est le plus utilisé et plus robuste) en termes de sécurité ainsi que d'autres métriques de performances via l'implémentation, l'expérimentation et la simulation.

Pour cela, nous commencerons par définir les outils nécessaires pour la réalisation des protocoles PBS basé-offset, SPBS basé sur la cryptographie symétrique et SPBS basé sur la cryptographie asymétrique. Ensuite, nous décrirons la mise en œuvre de toutes les structures de données et processus décrits lors de la conception. Nous terminerons ce chapitre par une présentation des résultats relevés lors des tests de performances.

2 Environnement de réalisation

Dans cette section, nous présentons les outils utilisés pour la mise en œuvre de notre protocole. Nous commençons tout d'abord par TinyOS, le système d'exploitation conçu pour les RCSF. Nous parlons ensuite du langage de programmation NesC avec lequel nous avons programmé le code du protocole. Nous terminons cette section par les techniques d'évaluation de performances.

2.1 TinyOS

TinyOS est un système d'exploitation intégré, modulaire, destiné aux réseaux de capteurs miniatures. Cette plate-forme logicielle ouverte est une série d'outils développés par l'Université de Berkeley et enrichie par une multitude d'utilisateurs. En effet, TinyOS est le plus répandu des OS pour les réseaux de capteurs sans-fil. Cet OS est capable d'intégrer très rapidement les innovations en relation avec l'avancement des applications et des réseaux eux même tout en minimisant la taille du code source en raison des problèmes inhérents de mémoire dans les réseaux de capteurs. La librairie TinyOS comprend les protocoles réseaux, les services de distribution, les drivers pour capteurs et les outils d'acquisition de données. La conception de TinyOS a été entièrement réalisée en NesC **[8]**, langage orienté composant syntaxiquement proche du C.

2.1.1 Pourquoi TinyOS ?

Les OS classiques sont généralement conçus pour un usage générique. Ils sont ainsi conçus en supposant une disponibilité sans limite des ressources. Leur objectif est la facilité d'usage, la

rapidité et efficacité. Parmi leurs caractéristiques sont : architecture multi-thread (mémoire importante), modèle E/S, séparation entre espace noyau et utilisateur, pas de contraintes d'énergie, ressources disponibles. Cependant, un mote (nœud capteur) a de nouvelles contraintes (voir le premier chapitre). Plusieurs systèmes d'exploitation ont été proposés pour les RCSF parmi lesquels on trouve SOS **[7]**, Contiki **[66]**, MANTIS **[67]**. TinyOS reste néanmoins le plus répandu pour les RCSF car il répond aux exigences particulières des applications des RCSF. Il convient alors de mentionner les propriétés qui rendent TinyOS aussi populaire pour ce genre de réseaux:

- Une taille de mémoire réduite.
- Une basse consommation d'énergie.
- Applications orientées composants: TinyOS fournit une réserve de composants systèmes utilisables au besoin.
- Programmation orienté évènement : Généralement sur TinyOS, un programme s'exécute suivant le déclenchement des événements. Sinon, les capteurs restent en veille ce qui maximise la durée de vie du réseau.

2.1.2 Caractéristiques de TinyOS

Les caractéristiques de TinyOS sont:

- Concurrence : utilise une architecture orientée événement
- Modularité :
  - Application composée de composants
  - OS + Application compilés en un seul exécutable
- Communication : utilise un modèle event / command
- Pas de séparation noyau / utilisateur

2.1.3 Notions principales

TinyOS est construit autour des différents concepts décrits ci-dessous:

- Les composants : constitués de :
  - Frame : est un espace mémoire de taille fixe permettant au composant de stocker les variables globales et les données qu'il utilise. Il n'en existe qu'un seul par composant.
  - Tâches : contiennent l'implémentation des fonctions. Elles sont décomposées en deux catégories : les commandes et les évènements.
- Les interfaces : représentent le descriptif des fonctions définies dans les tâches.

Deux principales versions de TinyOS sont disponibles: la version Tinyos-1.x et la version Tinyos-2.x. TinyOS peut être installé sur Windows (2000 et XP), GNU / Linux, Mac OS ou sur un capteur.

Dans ce travail, nous avons procédé à l'installation de la version de Tinyos-2.1.1 sur Linux (Ubuntu 10.04). La procédure d'installation se trouve dans **[72]**.

2.2 NesC : le langage de programmation de TinyOS

NesC est un langage de programmation orienté composants syntaxiquement proche du langage C. Il est conçu pour la réalisation des systèmes embarqués distribués, en particulier, les RCSF.

Une application NesC est un ou plusieurs composants reliés ensemble pour former un exécutable. Un composant est un élément de base pour former une application NesC. Il existe deux types de composants: modules et configurations **[8]**.

Une *configuration* définit les composants et/ou les interfaces utilisés par l'application déployée sur le capteur. Elle définit aussi la description des liaisons entre eux.

Un *module* constitue la brique élémentaire du code et implémente une ou plusieurs interfaces.

Une *interface* définit d'une manière abstraite les interactions entre deux composants. Elle définit un fichier décrivant les commandes et les évènements proposés par le composant qui les implémente. Une commande doit être implémentée par le fournisseur de l'interface et un évènement doit être implémenté par l'utilisateur de l'interface.

2.3 Les techniques d'évaluation de performances

2.3.1 Expérimentations

Il s'agit de faire des mesures et de les analyser directement sur un système réel. Cette technique permet de comprendre le vrai comportement du système.

Par la suite, nous présenterons les différents résultats obtenus par l'expérimentation. Le nœud capteur utilisé est le mote MICAz qui supporte un processeur de 8 Mhz (l'horloge principale). Nous avons utilisé 10 nœuds d'une plate-forme qui est accessible par internet dans **[73]**.

2.3.2 Modélisation

Il s'agit de réduire le système en un modèle mathématique (Les automates, Les réseaux de pétri, Les approches probabilistes, Les approches déterministes, etc…) et de l'analyser numériquement. Généralement, des hypothèses sont posées pour simplifier l'étape de modélisation du système et rendent l'évaluation numérique faisable. Ces hypothèses simplificatrices peuvent toucher à la fidélité de la représentation du système.

Dans ce travail, nous n'avons pas utilisé la modélisation pour évaluer notre protocole.

2.3.3 Simulation

La simulation est l'un des outils permettant de simuler (faire paraître comme réel ce qui ne l'est pas) des phénomènes réels et désigne un procédé selon lequel on exécute un programme informatique sur un ordinateur. La simulation n'est pas une méthode exacte, et nécessite de prêter une attention particulière aux interprétations des résultats. Plusieurs simulateurs **[68]** sont proposés pour les réseaux de capteurs tels que : TOSSIM, Avrora, NS-2, EmStar, OMNeT++, J-Sim, ATEMU.

Pour le cas de la simulation, nous avons utilisé le simulator Avrora pour estimer la consommation énergétique (voir **[71]** pour l'installation et utilisation de Avrora).

3 Implémentations

Afin de sécuriser l'approche ROS, notre travail s'est déroulé en trois phases :

- Implémentation du protocole PBS basé-offset.
- Implémentation du protocole SPBS basé sur la cryptographie à clé symétrique.
- Implémentation du protocole SPBS basé sur la cryptographie à clé publique.

3.1 Implémentation du protocole PBS basé-offset

Dans cette section, nous décrivons les structures de données ainsi que les principaux commandes et événements nécessaires pour l'implémentation du protocole PBS basé-offset.

Structures de données

Le paquet dans TinyOS est envoyé dans une structure de type message_t, qui est contenue dans un champ « int8_t data[TOSH_DATA_LENGTH] ». Nous utilisons deux structures de données, une structure pour le message de synchronisation et l'autre pour le message d'accusé de réception.

1) Le message de synchronisation

```
typedef nx_struct pair_sync_msg
{
  nx_uint16_t       src_add_msg ; / / l'identificateur du nœud A

 } pair_sync_msg ;
```

2) Le message d'accusé de réception

```
typedef nx_struct pair_ack_msg
{
 nx_uint16_t      src_add_msg ; / / l'identificateur du nœud P
 nx_uint32_t      ts_msg2 ; / / La valeur de temps de réception du message de
                            synchronisation à la couche MAC
 nx_uint32_t      ts_msg3 ; / / La valeur de temps d'émission du message d'accusé de
                            réception à la couche MAC
} pair_ack_msg ;
```

Evénements et commandes

Nous citons ici les principaux événements utilisés pour l'implémentation de PBS basé-offset.

Evénement	Sortie	Fonction
Timer.fired()	Void	Déclencher un round de synchronisation
SendPairSyncMsg.send(AM_BROADCAST_ADDR, &sndPwSyncMsgBuffer, sizeof(pair_sync_msg))	SUCCESS ou non	Diffuser le message de synchronisation
ReceivePairSyncMsg.receive(message_t* msg, void* payload, uint8_t len)	message_t*	Réception du message de synchronisation
SendPairAckMsg.send(AM_BROADCAST_ADDR, &sndPwAckMsgBuffer, sizeof(pair_ack_msg))	SUCCESS ou non	Diffuser le message d'accusé de réception
ReceivePairAckMsg.receive(message_t* msg, void* payload, uint8_t len)	message_t*	Réception du message d'accusé de réception

Estampillage à la couche MAC

Pour obtenir les estampilles à la couche MAC, nous choisissons l'interface PacketTimeStamp de Tinyos. Cette interface permet d'estampiller le temps de transmission et le temps de réception d'un paquet à la radio. Elle élimine l'erreur de synchronisation produit par : temps d'envoi, temps d'accès et temps de réception.

3.2 Implémentation du protocole SPBS

Dans cette section, nous donnons les commandes et les évènements nécessaires pour l'implémentation du protocole SPBS, ainsi que les modules et les outils de sécurité utilisés pour assurer les services de sécurité intégrés dans ce protocole.

En plus des champs utilisés dans les structures de données mis en place pour PBS basé-offset, SPBS a besoin des informations supplémentaires. Dans ce qui suit, nous définissons ces informations. Nous verrons deux versions de SPBS pour l'intégrité et l'authentification de données : SPBS basé sur la cryptographie à clé symétrique et SPBS basé sur la cryptographie à clé publique.

3.2.1 SPBS basé sur la cryptographie à clé symétrique

Dans cette section, nous décrivons les structures de données ainsi que les principaux commandes et événements nécessaires pour l'implémentation du protocole SPBS basé sur la cryptographie à clé symétrique.

Structures de données

Pour la sécurité, nous utilisons trois structures de données, une structure pour le message de synchronisation, une structure pour le message d'accusé de réception et l'autre pour le message d'estampillage qui nous avons l'ajouté au PBS basé-offset.

1) Le message de synchronisation

```
typedef nx_struct pair_sync_msg
{
  nx_uint16_t       src_add_msg ; / / l'identificateur du nœud A
  nx_uint32_t       Mac ; / / pour l'authentification et l'intégrité de message de
                            synchronisation
  nx_uint8_t        NA ; / / pour assurer la fraîcheur de message d'estampillage
 } pair_sync_msg ;
```

2) Le message d'accusé de réception

```
typedef nx_struct pair_ack_msg
{
  nx_uint16_t       src_add_msg ; / / l'identificateur du nœud P
  nx_uint8_t        NP ; / / pour assurer la fraîcheur de message d'estampillage
} pair_ack_msg ;
```

3) Le message d'estampillage

```
typedef nx_struct pair_ts_msg
{
   nx_uint16_t      src_add_msg ; / / l'identificateur du nœud P
   nx_uint32_t      Mac ; / / pour l'authentification et l'intégrité de message d'estampillage
```

```
  nx_uint8_t        NA ; / / pour assurer la fraîcheur de message d'estampillage
  nx_uint8_t        NP ; / / pour assurer la fraîcheur de message d'estampillage
} pair_ts_msg ;
```

Evénements et commandes

Nous citons ici les principaux événements utilisés pour l'implémentation du SPBS basé la cryptographie à clé symétrique.

1) Pour l'intégrité et l'authentification des messages de synchronisation et d'estampillage

Evénement	Sortie	Fonction
DataIntegrity.GenerateSecretKey()	uint32_t	Générer la clé secrète partagée entre les nœuds
DataIntegrity.Mac_generate(uint32_t *msg, uint16_t length, uint32_t key)	uint32_t	Calculer le code MAC
DataIntegrity.Mac_verify(uint32_t *msg, uint16_t length, uint32_t MacMsgR, uint32_t key)	Bool	Vérifier l'égalité entre le MAC reçu et celui calculé

2) Pour la fraîcheur de message d'estampillage

Evénement	Sortie	Fonction
DataFreshness.Nonce()	uint8_t	Calculer le nonce
DataFreshness.Freshness(uint8_t NA, uint8_t NA_R, uint8_t NP, uint8_t NP_R)	bool	Vérifier l'égalité entre les nonces reçus et ceux obtenus précédemment

3) Pour le non-retardement des messages de synchronisation et d'accusé de réception

Evénement	Sortie	Fonction
DataDelay.Delay(uint32_t delay, uint32_t delay_max, uint32_t delay_min)	bool	Vérifier l'exactitude de délai de propagatoion

Services de sécurité

Dans ce qui suit, nous donnons les algorithmes choisis pour assurer les services de sécurité intégrés dans SPBS basé sur la cryptographie à clé symétrique.

1) Authentification et intégrité des messages de synchronisation et d'accusé de réception

Pour la génération des codes MAC, nous avons choisi l'algorithme HMAC basé sur la fonction de hachage Poly32. Ce choix revient au système TinyOS qui implémente seulement deux fonctions

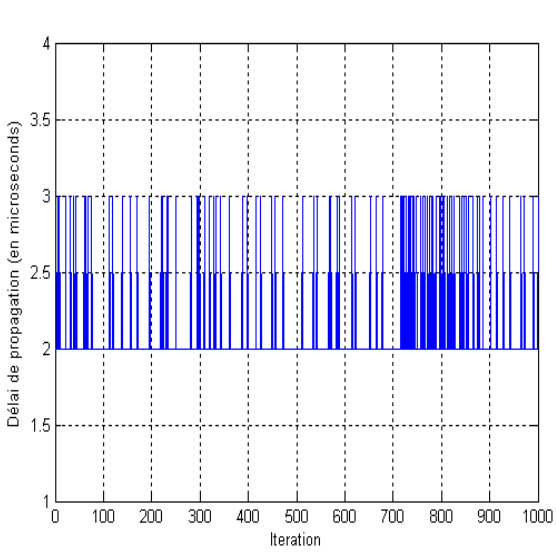
de hachage MMH et Poly32. L'utilisation de la fonction MMH a besoin le chiffrement par bloc ou le chiffrement en flot qui ajoute un coût de calcul. Pour cela, nous utilisons Poly32 pour générer le code MAC : *Mac ← HMAC-Poly32*.

2) La fraîcheur de message d'estampillage

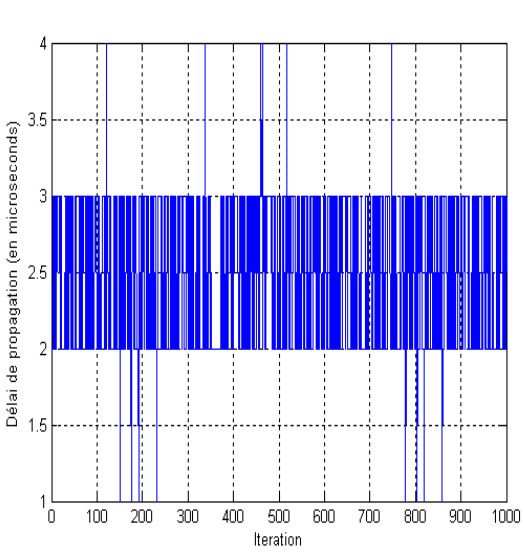
Afin d'assurer ce service, nous avons utilisés la commande *Random.rand16()* de TinyOS qui génère des nombres aléatoires (les nonces).

3) Le non-retardement des messages de synchronisation et d'accusé de réception

Afin d'assurer le non-retardement, nous avons utilisé deux seuils de délai de propagation (voir chapitre précèdent). Pour estimer ces seuils, nous avons implémenté nos protocoles et l'exécutés plusieurs fois pour plusieurs nœuds différents, ceci pour éliminer les spécifications du matériel. L'exécution a été faite dans un environnement réel (le nœud capteur est le mote MICAz) de 10 nœuds capteur. Nous avons choisi un round de synchronisation de 20 seconds. Pour chaque round de synchronisation, les nœuds *A* et *B* calculent le délai de propagation *d1* et *d2*, respectivement. La **Figure VI.1** (a et b) montre les délais de propagation *d1* et *d2* pour 1000 rounds de synchronisation.



(a) Le délai de propagation *d1* (délai entre le nœud *A* et le nœud *P*)



(b) Le délai de propagation *d2* (délai entre le nœud *B* et le nœud *P*)

Figure VI.1 : Estimation des délais de propagation.

Délai de propagation	d1	d2
Délai maximal	3 µs	4 µs
Délai minimal	2 µs	1 µs
Délai moyen (d_{moy})	2,08 µs	2,48 µs
L'écart type (σ)	0,29 µs	0,52 µs

Tableau VI. 1 : Statistiques des délais de propagation.

Le **Tableau VI.1** ci-dessus résume les statistiques de mesure des délais de propagation *d1* et *d2*. L'hypothèse de la distribution gaussienne (voir le chapitre précédent) du délai de propagation nous permet de calculer les seuils suivants :

Pour le nœud *A* : $d^*_{max} = d_{moy}+3\sigma \approx 3\ \mu s$ et $d^*_{min} = d_{moy}-3\sigma \approx 1{,}21\ \mu s$

Pour le nœud *B* : $d^*_{max} = d_{moy}+3\sigma \approx 4\ \mu s$ et $d^*_{min} = d_{moy}-3\sigma \approx 0{,}91\ \mu s$

Par conséquent, la valeur de l'impact maximum d'attaque pour le nœud *A* est égale à $\Delta=12\sigma=3{,}48\ \mu s$, et pour le nœud *B* est $\Delta=6\sigma=3{,}12\ \mu s$.

3.2.2 SPBS basé sur la cryptographie à clé publique

Dans cette section, nous décrivons les structures de données ainsi que les principaux commandes et événements nécessaires pour l'implémentation du protocole SPBS basé sur la cryptographie à clé publique.

Structures de données

1) Le message de synchronisation

```
typedef nx_struct pair_sync_msg
{
  nx_uint16_t       src_add_msg ; / / l'identificateur du nœud A
  nx_uint8_t      r[21] ;
  nx_uint8_t      s[21] ;
  / / <r,s> est la signature digitale pour l'authentification et l'intégrité de message de
      synchronisation
  nx_uint8_t       NA ; / / pour assurer la fraîcheur de message d'estampillage
 } pair_sync_msg ;
```

2) Le message d'accusé de réception

C'est la même structure vue dans la cryptographie à clé symétrique.

3) Le message d'estampillage

```
typedef nx_struct pair_ts_msg
{
   nx_uint16_t src_add_msg ; / / l'identificateur du nœud P
   nx_uint8_t r[21] ;
  nx_uint8_t  s[21] ;
   / / <r,s> est la signature digitale pour l'authentification et l'intégrité de message
       d'estampillage
  nx_uint8_t    NA; / / pour assurer la fraîcheur d'estampillage
  nx_uint8_t    NP ; / / pour assurer la fraîcheur d'estampillage
} pair_ts_msg ;
```

Evénements et commandes

L'implémentation des services de la fraîcheur de données et le non-retardement pour la cryptographie à clé publique est la même vue dans SPBS basé sur la cryptographie à clé symétrique. Dans ce qui suit, nous citons les principaux événements utilisés pour l'implémentation des services de l'intégrité de données et l'authentification de sources de messages pour la cryptographie à clé publique.

Evénement	Sortie	Fonction
DataIntegrity.GeneratePrivateKey()	uint8_t	Générer la clé privée
DataIntegrity.GeneratePublicKey(uint8_t *pKey)	uint8_t x[21] et uint8_t y[21]	Générer la clé publique
DataIntegrity.msg_sign(uint8_t *msg, uint8_t length, uint8_t *pkey)	uint8_t r[21] et uint8_t s[21]	Calculer la signature <r,s>
DataIntegrity.msg_verify(uint8_t *msg, uint8_t length, uint8_t *r, uint8_t *s, Point *pubKey)	Bool	Vérifier l'égalité entre la signature reçue et celle calculée

Services de sécurité

Pour générer la signature digitale et les clés privée et publique, nous avons utilisé l'algorithme de signature digitale à courbe elliptique (ECDSA) de la bibliothèque TinyECC **[32]**. L'algorithme ECDSA génère une signature digitale <*r*,*s*> pour assurer l'authentification et l'intégrité des messages de synchronisation et d'estampillage. Le **Tableau VI.2** ci-dessous résume le temps d'exécution de chaque opération de TinyECC:

Opération	Temps écoulé
Initialisation de l'algorithme	5,5 secondes
Calcul de la clé publique	1,788 secondes
Calcul de la signature digitale <r,s>	1,916 secondes
Vérification de la signature	2,431 secondes

Tableau VI. 2 : Temps d'exécution de TinyECC. [32]

Dans notre expérimentation, nous avons aussi calculé le temps d'exécution de TinyECC. Les résultats obtenus sont les mêmes résultats trouvés dans **[32]**.

4 Métriques à évaluer

Dans **[45]**, les métriques d'évaluation pour la synchronisation de temps sécurisée sont la précision, le temps de convergence, le taux de synchronisation et le coût de communication. Le temps de convergence et le taux de synchronisation sont évalués pour la synchronisation à multi-saut (réseau étendu). Tandis que pour nos protocoles utilisent la synchronisation à un saut. En outre, la consommation énergétique est un paramètre primordial pour la détermination de la durée de vie d'un RCSF (à cause de faible batterie). Par conséquent, pour l'analyse de nos protocoles dans les environnements bénis et hostiles, nous avons utilisé les métriques suivantes : le coût de communication, la consommation énergétique et la précision de synchronisation.

- **Coût de communication**. Elle consiste à calculer, pour chaque nœud du réseau, le nombre de messages envoyés et reçus. Et ceci dans le but d'avoir une idée sur la complexité de messages dans le réseau.

- **Consommation énergétique**. Le choix de cette métrique, comme étant un critère de performance, revient à sa nécessité dans la durée de vie d'un RCSF où l'énergie consommée est très critique. Pour mesurer l'énergie consommée dans chaque nœud (*A*, *P* et *B*), nous calculons l'énergie pour:
  - L'envoi et la réception des messages.
  - Les calculs effectués.
- **Précision de synchronisation**. Le choix de cette métrique, comme étant un critère de performance, revient à sa nécessité dans les applications RCSF où l'exactitude de la notion de temps commun est très importante.

  La précision de synchronisation est calculée de la manière suivante:
  - On calcule la différence entre les paires d'horloges synchronisées.
  - La moyenne de ces valeurs nous donne la précision.

5 Scénarios, résultats et interprétations

Dans cette section, nous décrivons une analyse de performance en évaluant d'abord les métriques déjà citées. Nous présentons une comparaison de performance entre nos protocoles et entre des protocoles existants (FTSP **[19]** et SPS **[42,43]**). Nous choisissons le protocole FTSP qu'il offre une haute précision ($1{,}48\ \mu s$). Tandis que le choix de SPS est justifié par sa robustesse à une erreur $\geq 30\mu s$.

5.1 Coût de communication

Dans cette section, nous décrivons la complexité de messages de nos protocoles, FTSP et SPS. Dans un réseau à un saut de *L* nœud où tous les nœuds peuvent communiquer entre eux, le nombre de messages échangés pour un round de synchronisation dans nos protocoles est 2 ou 3 messages. Tandis que le nombre de messages de FTSP et SPS est *L* et *2*(*L*-1), respectivement. Ainsi, de manière statistique nous obtenons avec *L* est égale de 2 à 12 nœuds, le graphe **Figure VI.2**.

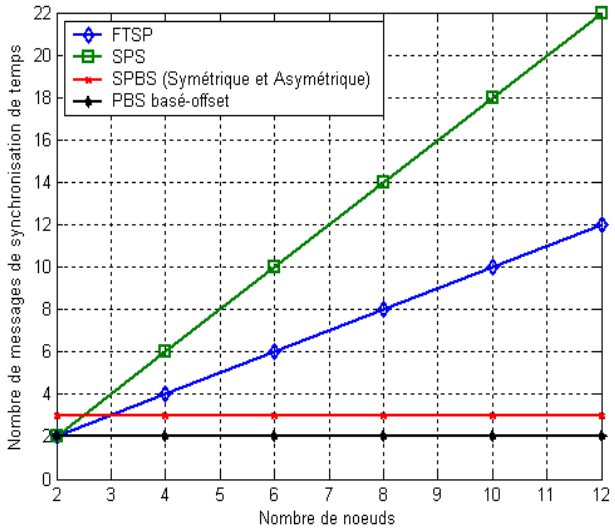


Figure VI. 2 : Comparaison du coût de communication dans le réseau.

D'après ce graphe (**Figure VI.2**), on voit que le nombre de messages envoyés augmente proportionnellement avec le nombre de nœuds dans FTSP et SPS. Tandis que celui de PBS basé-

offset et SPBS reste toujours constant et est égale à 2 et 3 messages, respectivement ; ceci est dû à l'indépendance du nombre de messages par rapport au nombre de nœuds. On remarque aussi que SPBS ajoute un seul message par rapport au PBS basé-offset. Donc, SPBS n'ajoute aucun coût de communication important. Concernant les protocoles FTSP et SPS, le nombre de messages est élevé, car le nombre de messages dépend du nombre de nœuds dans le réseau.

D'après les résultats obtenus, on constate que contrairement aux protocoles FTSP et SPS, SPBS et PBS basé-offset minimisent considérablement le nombre de messages, ce qui réduit la complexité de messages dans le réseau et par conséquent la consommation d'énergie. Dans la section suivante, nous verrons l'impact du nombre de messages envoyés dans le réseau sur l'énergie consommée.

5.2 Consommation énergétique

La consommation énergétique est une métrique très importante dans RCSF à cause de faible batterie des nœuds capteurs. Pour l'estimer, nous avons utilisé le simulateur Avrora. Il donne des résultats détaillés pour l'énergie consommée dans les différents composants, comme Radio, CPU, .etc. Nous avons obtenu par Avrora l'énergie consommée dans MICAz pour: émission d'un message ($C_{émission}$), réception d'un message ($C_{réception}$) et calculs effectués par les opérations de TinyECC ($C_{signature}$ et $C_{vérification}$). L'énergie consommée pour les calculs de code MAC est négligeable.

Opération	**Energie (µjoule)**
$C_{signature}$	1,705
$C_{verifiction}$	2
$C_{émission}$	22,404
$C_{réception}$	15,281

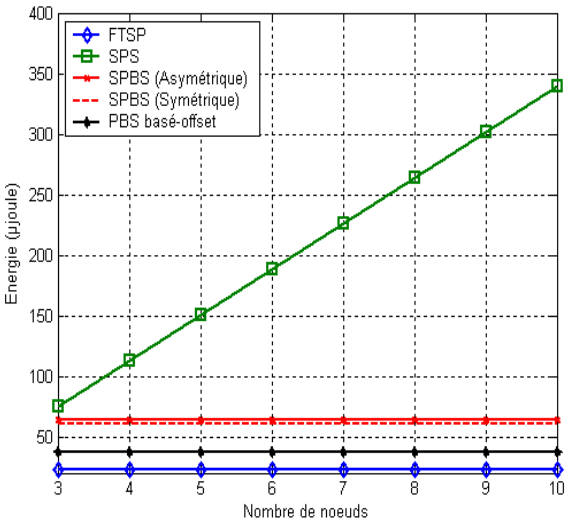
Tableau VI. 3: Energie consommée pour émission, réception et calculs.

D'après ces résultats de l'énergie consommée (émission, réception et calcul), nous pouvons déduire l'énergie consommée pour un seul round de synchronisation dans les protocoles suivants: PBS basé-offset, SPBS (Symétrique et Asymétrique), FTSP et SPS. Nous utilisons un réseau de *L* nœuds capteurs où tous les nœuds peuvent être communiqués.

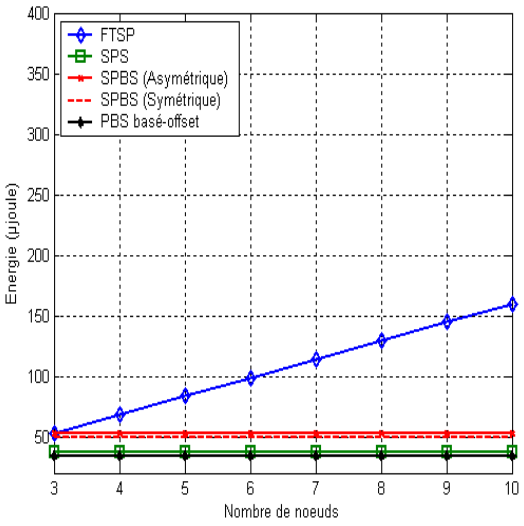
Protocole	**Energie consommée**	
	Nœud référence	**Nœud à synchroniser**
PBS basé-offset	$C_{émission} + C_{réception}$	$(C_{émission} + C_{réception}+2*C_{réception}) / 2$
SPBS (symétrique)	$2*C_{émission} + C_{réception}$	$(5*C_{réception}+ C_{émission}) / 2$
SPBS (asymétrique)	$C_{réception}+C_{vérification} + 2*C_{émission} + C_{signature}$	$(3*(C_{réception}+C_{vérification})+ C_{émission} + C_{signature} +2*C_{réception}) / 2$
FTSP	$C_{émission}$	$C_{émission} + (L-1)*C_{réception}$
SPS	$(L-1)*(C_{réception}+C_{émission})$	$(C_{réception}+C_{émission})$

Tableau VI. 4: Estimation de la consommation énergétique.

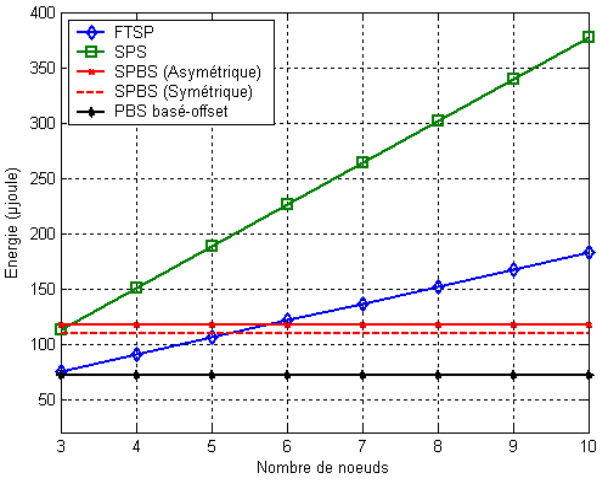
Ainsi de manière statistique avec le nombre de nœuds *L* est égale de 3 à 10, nous traçons les graphes (a), (b) et (c) de la **Figure VI.3**.



(a) Energie consommée dans le nœud référence



(b) Energie consommée dans chaque nœud à synchroniser



(c) Energie consommée dans un nœud pour la synchronisation multi-saut

Figure VI. 3 : Comparaison de la consommation énergétique dans les nœuds.

Le graphe (a) de la **Figure VI.3** montre l'énergie consommée dans le nœud référence. On remarque que le nœud référence dans SPS consomment plus, ceci est parce que le nœud référence synchronise les nœuds en échangeant deux messages pour chaque nœud du réseau. En outre, le nœud référence dans FTSP consomme moins car il s'échange un message seulement avec tous les nœuds voisins. Le nœud référence dans SPBS basé cryptographie asymétrique consomme plus que dans PBS basé-offset et SPBS basé cryptographie symétrique à cause de l'énergie consommée par les opérations cryptographiques de TinyECC.

Le graphe (b) de la **Figure VI.3** montre l'énergie consommée dans chaque nœud à synchroniser. On remarque que les nœuds à synchroniser consomment plus d'énergie dans FTSP, ceci parce que

le nœud à synchroniser se synchronise en recevant tous les messages des nœuds voisins. Tandis que le nœud à synchroniser dans les autres protocoles consomme moins d'énergie.

Le graphe (c) de la **Figure VI.3** rassemble les deux graphes (a) et (b). Il montre l'énergie consommée dans un nœud qui est un nœud référence et un nœud à synchroniser en même temps. Et ceci dans la synchronisation multi-saut où un nœud se synchronise avec le nœud référence ensuite il synchronise d'autres nœuds. Par exemple dans le graphe (c) de la **Figure VI.3**, l'énergie consommée dans un nœud pour le nombre de nœuds égale à 5 est l'énergie consommée par un nœud qui se synchronise dans un groupe de 5 nœuds, plus l'énergie pour qu'il synchronise un autre groupe de 5 nœuds. D'après ce graphe (c), on remarque que l'énergie consommée dans FTSP et SPS augment proportionnellement avec le nombre de nœuds. Tandis que celui de PBS basé-offset et SPBS (Symétrique et Asymétrique) reste toujours constant. L'énergie consommée dans SPBS basé sur la cryptographie à clé publique est plus par rapport aux PBS basé-offset et SPBS basé sur la cryptographie à clé symétrique à cause des calculs effectués pour la signature digitale. Comme montre le graphe (c), SPBS basé sur la cryptographie asymétrique est meilleur par rapport SPS (qui utilise la cryptographie symétrique) dans la synchronisation multi-saut.

On constate que le protocole SPBS (Symétrique et Asymétrique) minimise considérablement la consommation énergétique dans la synchronisation de temps multi-saut par rapport les protocoles existants (FTSP et SPS).

5.3 Erreur de synchronisation

Les applications RCSF exigent une forte exactitude de la notion de temps commune, car une petite erreur de temps entre les nœuds capteur conduit à une exécution incorrecte de l'application. Pour cela l'erreur de synchronisation est une métrique importante pour évaluer une approche de synchronisation de temps.

Dans ce qui suit, nous montrons l'erreur de synchronisation de nos protocoles implémentés précédemment. Pour calculer l'erreur de synchronisation, nous avons exécuté nos protocoles avec plusieurs rounds de synchronisation de 20 secondes. L'exécution a été faite dans un environnement réel de 10 nœuds capteurs de la plate-forme MICAz. Deux parmi les nœuds sont désignés comme le nœud référence (nœud *P*) et le nœud source (nœud *A*), respectivement. Les autres nœuds sont désignés comme les nœuds récepteurs (nœuds *B*). Pour chaque round de synchronisation, les nœuds calculent la différence d'horloge et la corrige. Ensuite, juste après la correction de l'horloge, un des nœuds récepteurs diffuse un message vers tous les nœuds (*P*, *A* et *B*). La différence entre le temps de réception de ce message dans les différents nœuds nous donne l'erreur de synchronisation.

A la fin, nous traçons un tableau qui résume la précision de synchronisation de nos protocoles et de protocoles déjà existants pour faire la comparaison entre eux.

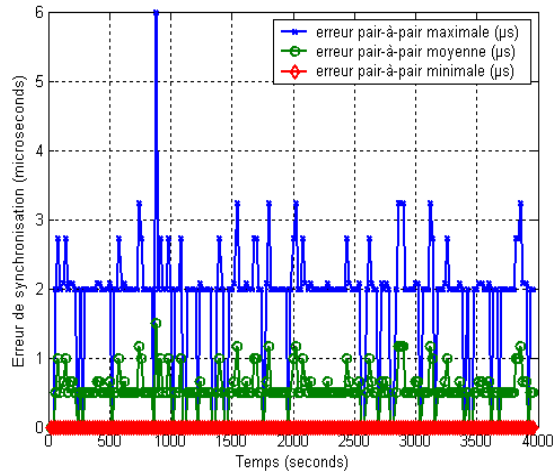
5.3.1 SPBS basé cryptographie symétrique et PBS basé-offset

Les protocoles PBS basé-offset et SPBS basé sur la cryptographie à clé symétrique donnent la même erreur de synchronisation, car il n'existe pas une grande différence entre eux pour le coût de communication ou le coût de calcul. Le graphe (a) de la **Figure VI.4** montre l'erreur de synchronisation entre les 9 nœuds capteur (le 10ème nœud est utilisé pour le calcul de l'erreur de synchronisation). On remarque que l'erreur de synchronisation maximale de ces protocoles est 6μs,

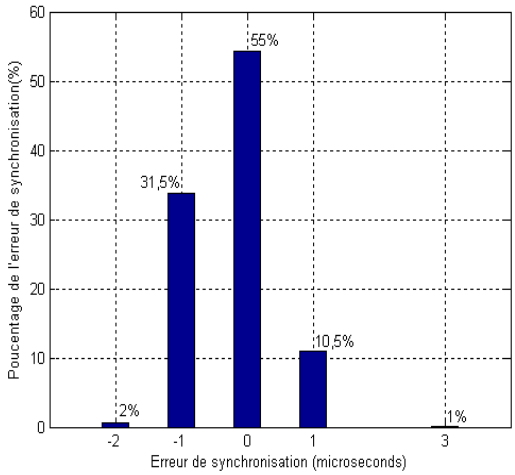
et l'erreur de synchronisation minimale est 0 µs. La valeur maximale de l'erreur de synchronisation moyenne est 1,5 µs (la précision de synchronisation).

Le graphe (b) de la **Figure VI.4** montre le pourcentage de l'erreur de synchronisation, on remarque que le pourcentage des nœuds qui sont synchronisés parfaitement est de 55%. Le pourcentage des nœuds qui sont synchronisés avec une erreur de ±1µs par rapport nœud référence est de 42%. Donc 97% des nœuds sont synchronisés avec une erreur égale à 0µs ou 1µs par rapport nœud référence.

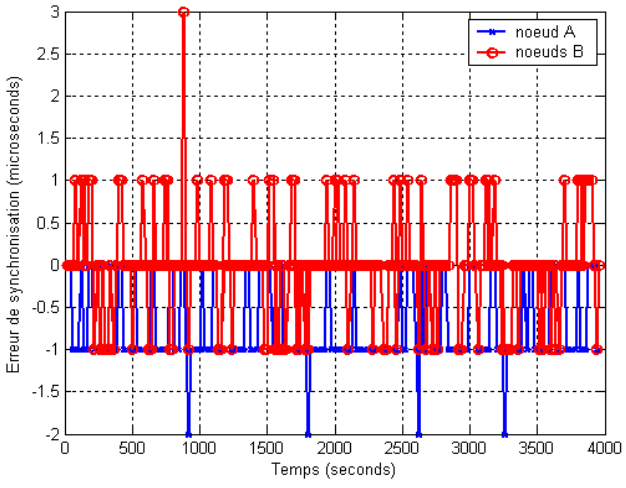
Le graphe (c) de la **Figure VI.4** montre la différence d'horloge par rapport au nœud référence. On remarque que l'erreur de synchronisation entre le nœud référence *P* et les nœuds *A* ou *B* ne dépasse pas 3 µs.



(a) Erreur de synchronisation en fonction de temps



(b) Statistiques de l'erreur de synchronisation de temps



(c) La différence d'horloge relative par rapport au nœud référence après la synchronisation

Figure VI. 4: L'erreur de synchronisation des protocoles PBS basé-offset et SPBS basé cryptographie symétrique.

D'après ces résultats, notre protocole SPBS basé sur la cryptographie à clé symétrique donne une haute précision et sécurisé contre les attaques. Donc, on constate que ce protocole est utilisé pour les applications réseaux de capteurs qui demandent une haute précision.

5.3.2 SPBS basé sur la cryptographie à clé publique

Dans cette section, nous montrons l'erreur de synchronisation du protocole SPBS basé sur la cryptographie à clé publique. Avec le simulateur Avrora, le protocole SPBS basé sur la cryptographie à clé publique offre la même erreur de synchronisation obtenue dans SPBS basé symétrique. Dans ce qui suit, nous analysons les résultats obtenus par l'expérimentation.

Les graphes (a), (b), (c) et (d) de la **Figure VI.5** montrent la différence d'horloge par rapport au nœud référence. Le graphe (a) trace la différence d'horloge pour le nœud *A*, et les autres graphes (b, c et d) tracent la différence d'horloge pour trois nœuds récepteurs comme le nœud *B*. On remarque que l'erreur de synchronisation est très élevée et elle est dépend du nœud à synchroniser. Donc, la précision sera basse.

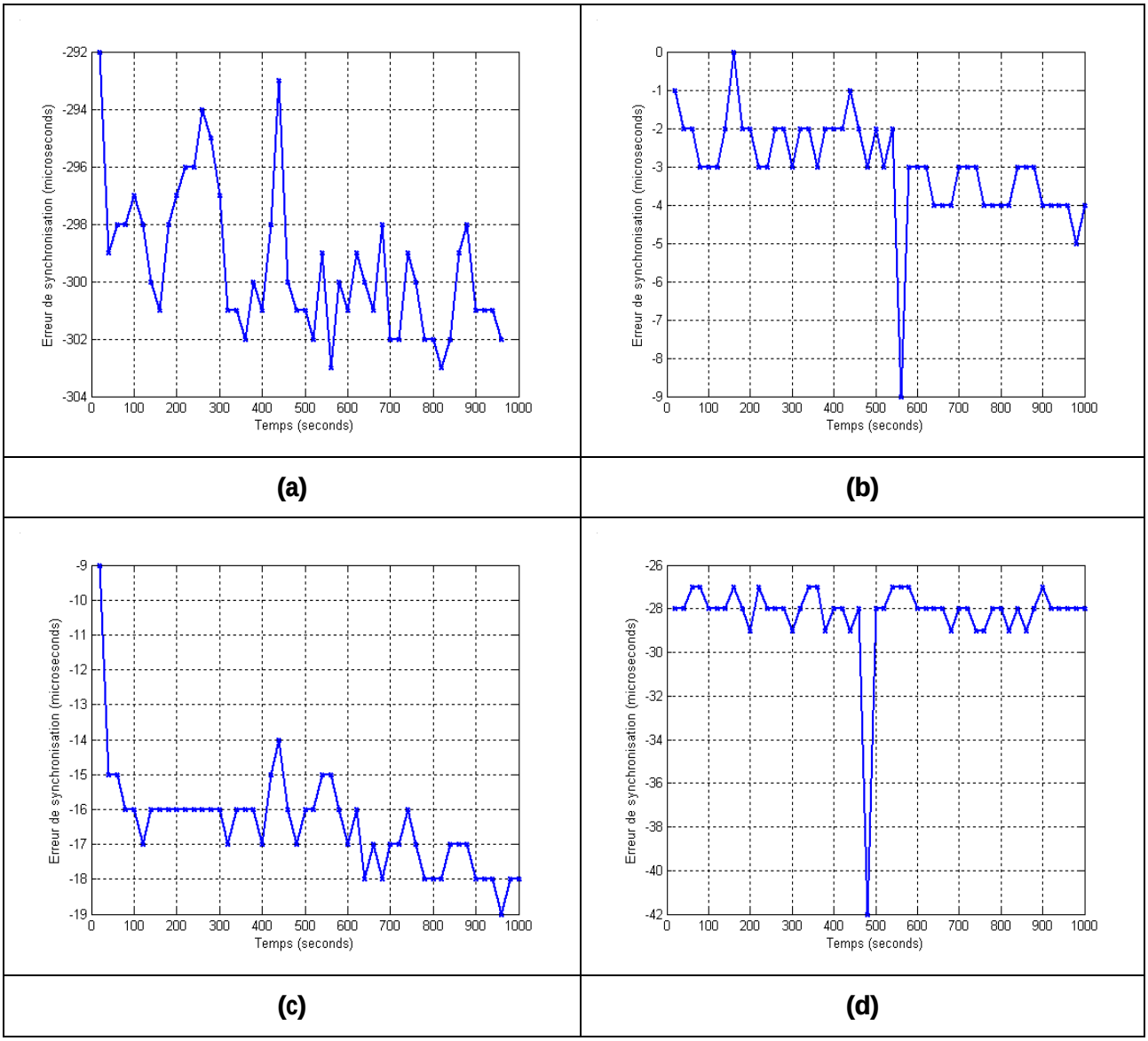


Figure VI. 5 : L'erreur de synchronisation du protocole SPBS basé sur la cryptographie à clé publique.

Comme montre la **Figure VI.6**, le nœud *P* signe le message d'estampillage <ts> ensuite le diffuse. Le temps de cette signature est 1,9 secondes. Après, les nœuds *A* et *B* reçoivent le message d'estampillage et vérifient l'intégrité de ce message dans 2,4 secondes. La différence d'horloge calculée par les nœuds *A* et *B* est correspondent au temps X. Tandis que dans le temps Y, les nœuds *A* et *B* corrigent l'horloge. Donc, la durée entre le temps de calcul de la différence d'horloge et le temps de la correction de l'horloge est 4,3 secondes. Dans la pratique, l'horloge peut diverger jusqu'à 100 µs dans une seconde à cause de variation de la fréquence de l'horloge. Par conséquent dans 4,3 secondes, les horloges locales des nœuds *A* et *B* peuvent diverger jusqu'à 430 µs par rapport nœud référence *P*. Ceci nous a donné une basse précision de synchronisation et cette précision dépend de la différence entre les fréquences d'horloges (c'est-à-dire: le skew) comme le montre les graphes de la **Figure V.5**.

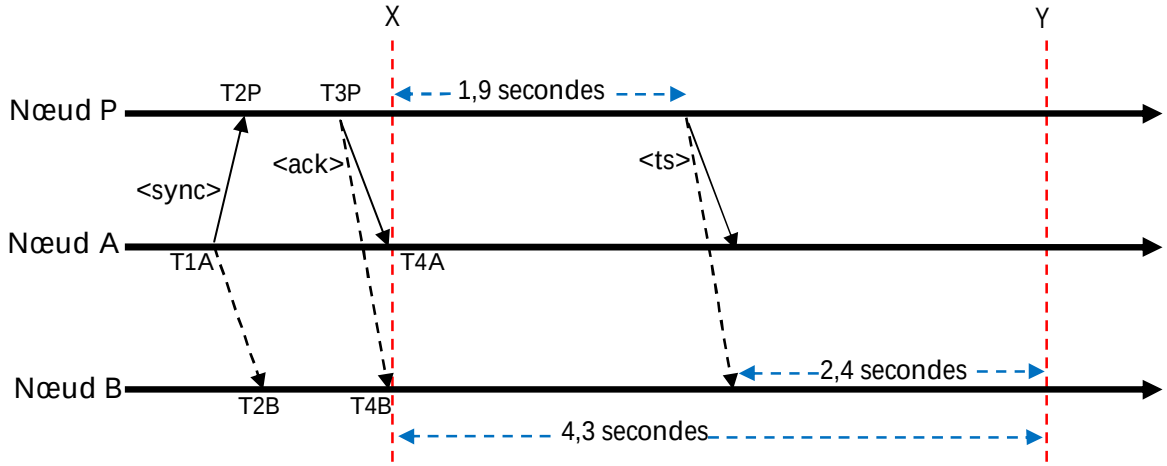


Figure VI. 6: Détail du SPBS basé cryptographie asymétrique.

Nous avons utilisé la cryptographie à clé publique pour une forte robustesse, mais elle a un inconvénient pour la précision à cause du temps écoulé par cette cryptographie. Donc, on constate que la cryptographie à clé publique pour la synchronisation de temps est utilisée par les applications réseaux de capteurs qui exigent la sécurité avant la précision.

5.3.3 Comparaison de la précision de synchronisation

Le **Tableau VI.5** résume la précision de synchronisation de nos protocoles et des protocoles existants.

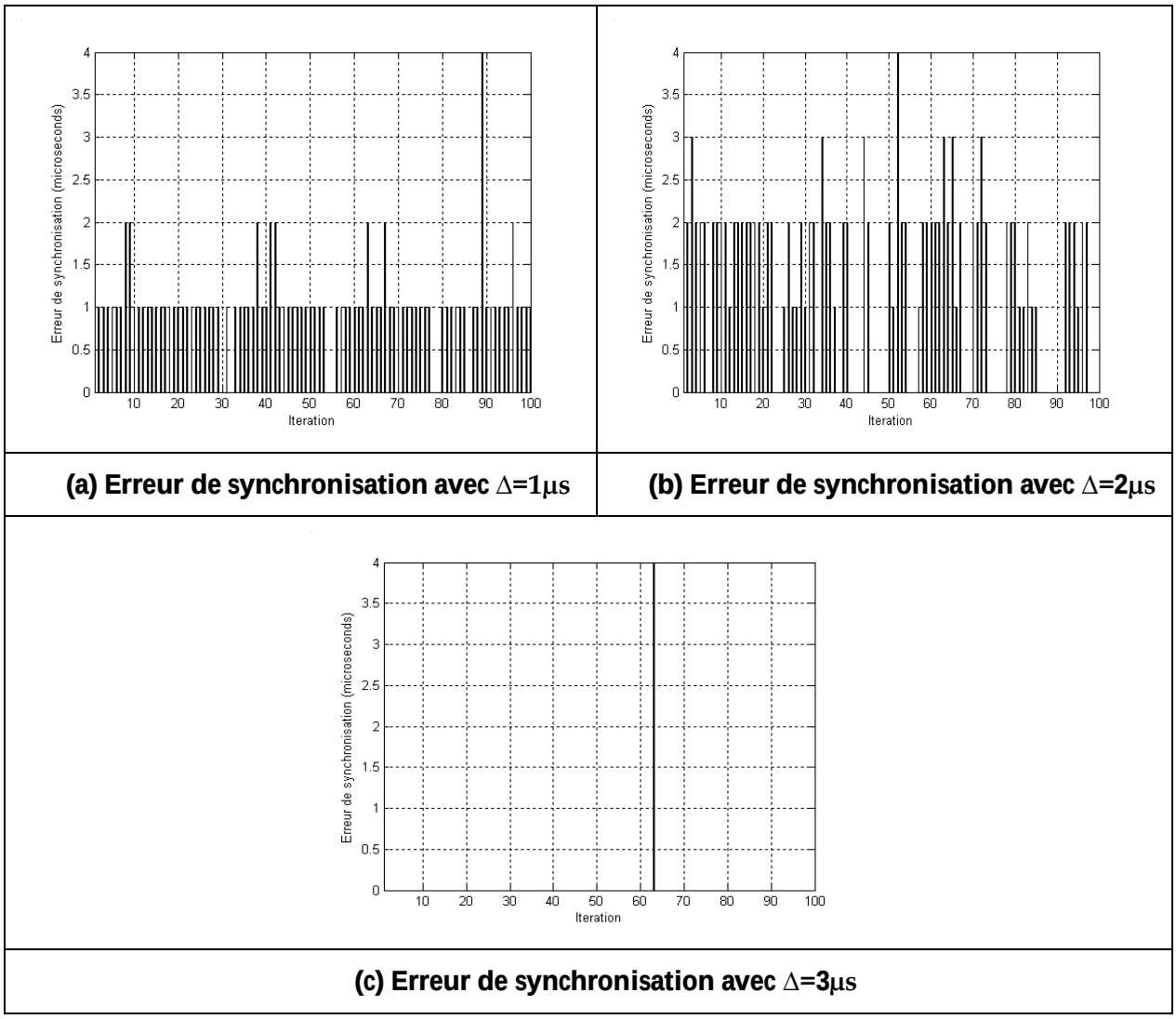
Protocole de synchronisation de temps	Précision de synchronisation	
RBS	29,1 µs (Estampillage à la couche application) 1,9 µs (Estampillage à la couche MAC)	
TPSN	16,9 µs	
FTSP	1,48 µs	
PBS	Même que RBS	
PBS basé-offset	1,5	
SPS	12,05 µs	
SPBS	Symétrique	1,5 µs
	Asymétrique	[0µs,440µs]

Tableau VI. 5 : Comparaison de la précision de synchronisation.

On remarque que les protocoles SPBS basé cryptographie symétrique et PBS basé-offset offrent la même précision que FTSP qu'il est actuellement utilisé par le TinyOS. En outre, SPBS basé cryptographie symétrique a une précision élevée que SPS qu'il est plus robuste aux attaques. Le PBS est moins précis que PBS basé-offset qu'il utilise l'estampillage à la couche MAC. Pour la version de SPBS basé sur la cryptographie à clé publique, on remarque que la précision est très basse et elle dépend de la fréquence de l'horloge comme on a vu précédemment.

6 Robustesse aux attaques

Dans cette section, nous étudions les conséquences d'attaquer le protocole SPBS. Les attaques manipulation de messages, rejeu et Sybil sont détectées par les mécanismes (Nonces, code MAC et TinyECC) vues précédemment. Tandis que les attaques delay, wormhole et fallacieuse ne peuvent pas être empêchées par la cryptographie, nous avons utilisé une méthode statistique (deux seuils de délai de propagation) pour faire face contre elles. Dans cette section, nous nous sommes intéressés à la robustesse des seuils de délai de propagation utilisés. Pour cela, nous présentons des résultats d'expérimentation obtenus en utilisant la plate-forme MICAz afin d'évaluer la résilience contre les attaques (delay, wormhole et fallacieuse) qui retardent les messages et ne peuvent pas être détecté par la cryptographie.



(a) Erreur de synchronisation avec Δ=1µs

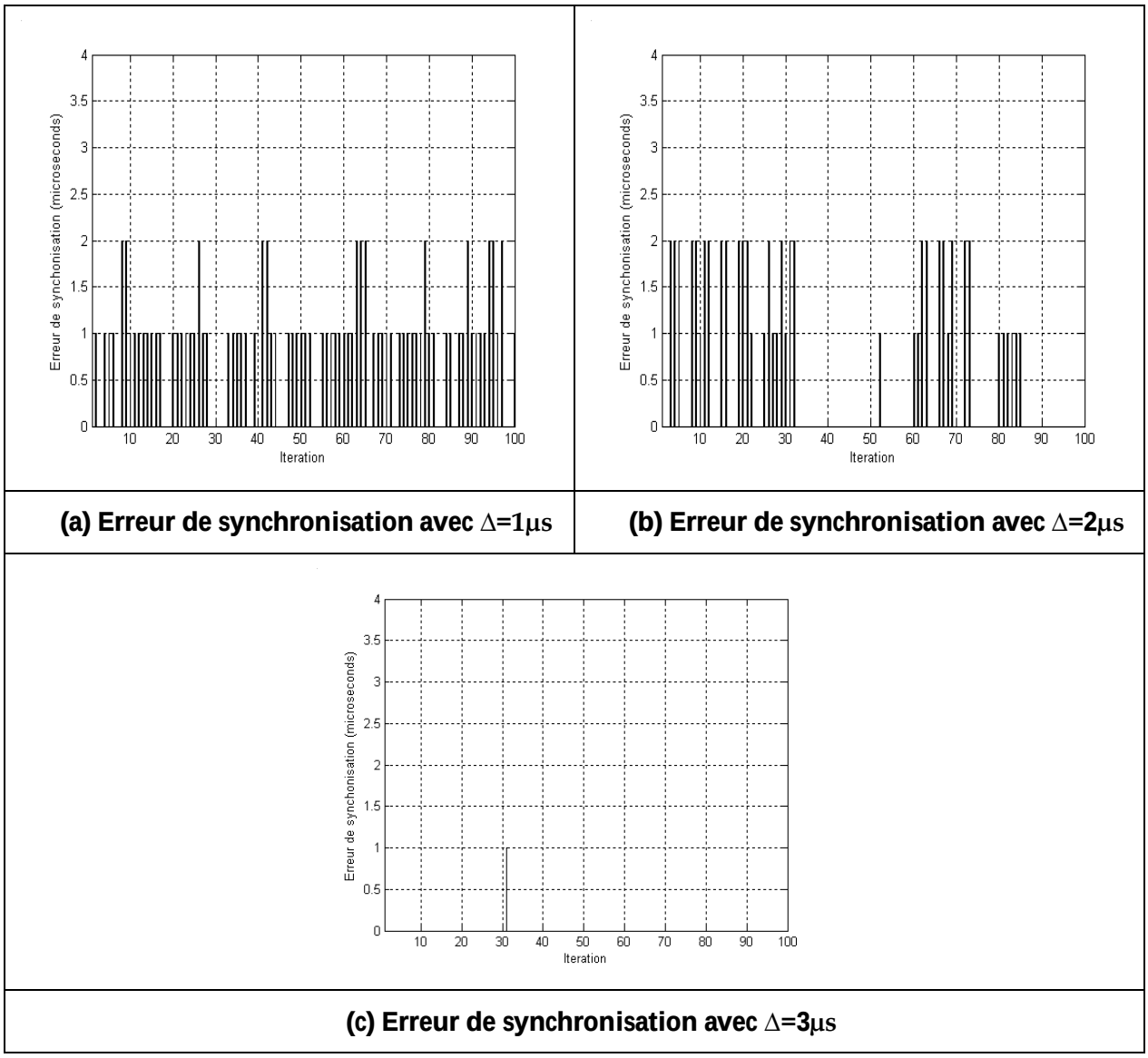
(b) Erreur de synchronisation avec Δ=2µs

(c) Erreur de synchronisation avec Δ=3µs

Figure VI. 7 : Erreur de synchronisation du nœud *A* avec la présence d'attaque.

Nous avons utilisé trois nœuds capteurs MICAz. L'un des nœuds est désigné comme le nœud référence *P*. Les deux autres nœuds sont le nœud *A* et le nœud *B*. Pour l'attaque, nous avons programmé les nœuds *A* et *B* pour modifier la valeur de l'estampille (ex: $T2^{(P)}$), ensuite il calcule la différence d'horloges et fait la correction. S'il détecte l'erreur introduite, alors il annule la synchronisation, sinon il se synchronise. Nous avons exécuté le protocole pour 100 itérations, et pour chaque itération nous calculons l'erreur de synchronisation des nœuds synchronisés. Pour calculer l'erreur de synchronisation, nous avons utilisé un autre nœud (comme *B*) qui diffuse un message après chaque synchronisation. La différence entre le temps de réception de ce message dans le nœud *P*, et dans les nœuds *A* et *B* nous donne l'erreur de synchronisation par rapport au nœud référence *P*.

Les graphes (a), (b) et (c) des **Figure VI.7** et **Figure VI.8** montrent l'erreur de synchronisation calculée avec une attaque de 1, 2 et 3 µs, respectivement. On remarque que les rounds de synchronisation annulés augmentent proportionnellement avec l'erreur introduite.



(a) Erreur de synchronisation avec Δ=1µs

(b) Erreur de synchronisation avec Δ=2µs

(c) Erreur de synchronisation avec Δ=3µs

Figure VI. 8 : Erreur de synchronisation du nœud *B* avec la présence d'attaque.

Le **Tableau VI.6** résume les statistiques de l'erreur de synchronisation calculée et la probabilité de détection pour chaque erreur introduite. L'attaquant réussite à introduire l'erreur lorsque l'erreur Δ est inferieur ou égale à l'impact maximum ($\approx$3,5 μs). D'après ce tableau (**Tableau VI.6**), les nœuds *A* et *B* détectent l'attaque avec une probabilité de détection de 99% lorsque l'erreur introduite égale 3μs. On remarque aussi que l'erreur de synchronisation ne dépasse pas 4 μs.

D'après **[42,43]**, le protocole SPS détecte l'attaque avec une probabilité de détection de 100% lorsque l'erreur introduite égale 30μs. Par conséquent, notre protocole SPBS est plus robuste que SPS.

L'erreur introduite	Erreur maximale (μs)		Erreur minimale (μs)		Erreur moyenne (μs)		Probabilité de détection (%)	
	A	B	A	B	A	B	A	B
Δ= 1 μs	4	4	0	0	1,04	0,96	1%	2%
Δ= 2 μs	4	2	0	1	1,87	1,58	25%	59%
Δ= 3 μs	4	4	4	4	4	4	99%	99%

Tableau VI. 6 : Statistiques de l'erreur de synchronisation avec la présence d'attaques.

7 Conclusion

Dans ce chapitre, nous avons présenté l'implémentation ainsi que l'évaluation des trois protocoles PBS basé-offset, SPBS basé sur la cryptographie à clé symétrique et SPBS basé sur la cryptographie à clé publique. Le système d'exploitation TinyOS est utilisé. Sa conception a été entièrement réalisée en langage NesC. Nous avons analysé notre approche dans un environnement réel en utilisant la plate-forme MICAz.

Pour l'évaluation, nous avons aussi utilisé la simulation, mais elle nous a donnée des résultats incorrectes pour le cas de la cryptographique à clé publique.

Les résultats d'expérimentation montrent que notre approche offre de meilleures performances (coût de communication, la consommation énergétique et l'erreur de synchronisation) par rapport au protocole de synchronisation FTSP (protocole utilisé actuellement par le Tinyos-2.x) qui donne une haute précision. En plus, notre protocole présente une meilleure résilience que le protocole SPS (le protocole le plus utilisé dans la synchronisation multi-saut).

Conclusion Générale et Perspectives

La synchronisation de temps sécurisée dans les réseaux de capteurs est essentielle pour différentes applications réseaux de capteurs. Récemment, Ce problème est largement étudié dans les réseaux de capteurs sans fil. En outre, les contraintes des réseaux de capteurs sont des facteurs essentiels pour construire un protocole de synchronisation de temps sécurisée et par conséquence, parmi les solutions proposées, il y a des approches qui ont un coût de communication élevé et une efficacité énergétique basse.

A travers notre étude des différentes approches de synchronisation de temps, nous avons vu qu'il y a trois approches de base pour la synchronisation de temps dans les réseaux de capteurs. Ces approches sont : la synchronisation *Emetteur-Récepteur* (SRS), la synchronisation *Récepteur-Récepteur* (RRS) et la synchronisation *Seulement-Récepteur* (ROS). L'approche ROS offre de meilleures performances par rapport aux approches SRS et RRS. En outre, à traves notre étude des approches de sécurité pour la synchronisation de temps, nous avons trouvé qu'aucune approche de sécurité n'est fournie pour ROS.

Dans ce projet, nous nous sommes intéressés à la sécurisation de l'approche ROS. Nous avons établi un modèle de synchronisation de temps qui nous avons appelé PBS basé-offset. Il se base sur l'approche ROS. Ensuite, nous avons défini les différentes attaques (Sybil, manipulation de message, rejeu, delay, wormhole et fallacieuse) contre ce modèle et les mécanismes de sécurité pour faire face à ces attaques. Ainsi, nous avons obtenu un protocole qui nous avons appelé SPBS (*Secure Pairwise Broadcast Synchronisation*). Les mécanismes de sécurité utilisés permettent d'assurer la fraîcheur de données, l'intégrité et l'authentification de données, et le non-retardement de messages. Pour la fraîcheur de données, nous avons utilisé deux nonces, et pour le non-retardement deux seuils de délai de propagation sont utilisés (un seuil maximal et seuil minimal). Pour l'intégrité et l'authentification de données, nous avons proposé deux versions de SPBS, une pour la cryptographie à clé symétrique et l'autre pour la cryptographie à clé publique. Pour l'analyse et la réalisation, nous avons implémenté et analysé notre protocole dans un environnement réel en utilisant la plate-forme MICAz. Les résultats d'expérimentation montrent que notre approche offre de meilleures performances (coût de communication, consommation énergétique et erreur de synchronisation) par rapport au protocole de synchronisation FTSP qui offre une haute précision. En plus, notre protocole est plus précis et robuste à l'attaque delay par rapport au protocole SPS. SPS détecte l'attaque delay avec une probabilité de détection de 99% lorsque l'erreur introduite égale $35\mu s$. $\mathrm{Tandis\ que\ SPBS}$ détecte l'attaque delay avec une probabilité de détection de 99% lorsque l'erreur introduite égale $3\mu s$.

Nous avons utilisé la cryptographie à clé publique pour une forte robustesse, mais les résultats d'expérimentation montrent que le SPBS basé cryptographie asymétrique a un inconvénient pour la précision à cause du temps écoulé par les opérations de cette cryptographie. Dans le travail futur, nous pensons d'améliorer la précision du SPBS basé cryptographie asymétrique en estimant le skew ou en réduisant le temps requis pour la génération et/ou la vérification de la signature digitale. En plus, nous ferons l'extension de SPBS pour un réseau étendu en utilisant les différentes approches d'extension proposées pour PBS **[69,70]**.

Glossaire

μTESLA	Timed Efficient Stream Loss-tolerant Authentication
AES	Advanced Encryption Standard
ATSP	Attack-tolerant Time-Synchronization Protocol
CPU	Central Processing Unit
CRT	Chinese Remainder Theorem
CSMA	Carrier Sense Multiple Access
CTP	Classless Time Protocol
DAS	Detection Analysis Self-healing
DES	Data Encryption Standard
DoS	Denial of Service
DSP	Digital Signal Processor
ESD	Extreme Studentized Deviate
FTSP	Flooding Time Synchronization Protocol
GESD	Generalized ESD
GPS	Global Position System
HBS	H-sensor Broadcast Synchronization
HMAC	keyed-Hash Message Authentication Code
HSN	Heterogeneous Sensor Network
IBE	Identity-Based Encryption
ICTS	Interactive Convergence Time Synchronization
IEEE	Institute of Electrical and Electronic Engineer
IP	Internet Protocol
MAC	Medium Access Control
MAC	Message Authentication Code
MIC	Message Integrity Code
NesC	Network embedded system C
NTP	Network Time Protocol
OS	Operating System
PBC	Pairing-Based Cryptography
PBS	Pairwise Broadcast Synchronization
PDA	Personal Digital Assistant
RBS	Reference Broadcast Synchronization
RC4	Rivest Cipher 4
RCSF	Réseau de Capteurs Sans Fil

RLS	Recursive Least Square
ROS	Receiver-Only Synchronization
RRS	Receveir-Rceiver Synchronization
RSA	Rivest Shamir Adleman
SNTP	Simple NTP
SPBS	Secure Pairewise Broadcast Synchronization
SRS	Sender-Receiver Synchronization
TDMA	Time Division Multiple Access
TinyECC	Tiny Elliptic Curve Cryptography
TinyOS	Tiny Operating System
TinySecRSync	Tiny Secure Resilient Time Synchronization
TOSSIM	TinyOS Simulator
TPSN	Time-synch Protocol Sensor Networks
UTC	Universal Time Coordinates
WSN	Wireless Sensor Network

Bibliographie

[1] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci "A Survey on Sensor Networks", IEEE Communications Magazine, Août 2002.

[2] http : / / www.shockfish.com/ site Internet de Shockfish SA.

[3] http : / / www.xbow.com / site Internet de Crossbow Technology, Inc.

[4] Imrich Chlamtac, Iacopo Carreras et Hagen Woesner "From internets to bionets : biological kinetic service oriented networks". The case study of Bionetic Sensor Networks. CREATE-NET Research Consortium, Trento, Italy 2005.

[5] J. Yick, B. Mukherjee, D. Ghosal. Wireless sensor network survey. Computer Networks 52 (2008) 2292–2330, 2008 Elsevier.

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System architecture directions for networked sensors", in Proceedings of Architectural Support for Programming Languages and Operating Systems, 2000, pp. 93.

[7] C. Han, R. Kumar, R. Shea, E. Kohler and M. Srivastava, "A Dynamic Operating System for Sensor Nodes" Proceedings of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys), 2005.

[8] P. Levis, D. Gay. "TinyOS Programming.". ISBN-13 : 9780521896061, Cambridge University Press, April 2009.

[9] Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks. In Proceedings of the 1st International Workshop on Sensor Network Protocols and Applications, May 2003.

[10] D. Tian and N. D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In First ACM International Workshop on Wireless Sensor Networks and Applications WSNA02, pages 32–41, September 2002.

[11] K. Römer, P. Blum, and L. Meier. Time synchronization and calibration in wireless sensor networks. In Ivan Stojmenovic, editor, Wireless Sensor Networks. John Wiley Sons, 2005.

[12] D.L.Mills. Modelling and Analysis of Computer Network Clocks. Technical Report, 92-5-2, Electrical Engineering Department, University of Delaware, May 1992.

[13] S.B.Moon, P.Skelly, and D.Towsley. Estimation and Removal of Clock Skew from Network Delay Measurements. Proc. IEEE INFOCOM, Vol 1, pp. 227-234, Mar. 1999.

[14] D.L. Mills. Internet time synchronization: The network time protocol. IEEE Transactions on Communications, 39(10):1482–1493, 1991.

[15] B. Sundararaman et al., "Clock synchronization for wireless sensor networks" : a survey, Ad-Hoc Networks, vol. 3, no. 3, pp. 281-323, Mar. 2005.

[16] Noh, K. L.Serpedin, E. ; Qaraqe, K. "PBS : A new approach for time synchronization in wireless sensor networks : pairwise broadcast synchronization". IEEE Trans. Wireless Commun. 2008, 9, 3318-3322.

[17] J. Elson, L. Girod, and D. Estrin. RBS: "Fine-grained network time synchronization using reference broadcasts," in Proc. Fifth Symposium on Operating System Design and Implementation (OSDI), Dec. 2002.

[18] Ganeriwawal, S., Kumar, R., Srivastava, M. "TPSN : Timing-Sync Protocol for Sensor Networks". The first ACM Conference on Embedded Networked Sensor Systems (SenSys), p. 138-149, November 2003.

[19] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. "FTSP : The flooding time synchronization protocol". In ACM Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSyS 2004), November 2004, pp. 39–49.

[20] Q. Li and D. Rus. Global clock synchronization in sensor networks. In Proceedings of IEEE INFOCOM 2004, pages 21k226, March 2004.

[21] Su, W. ; Akyildiz, I.F. "Time-diffusion synchronization protocol for wireless sensor networks". IEEE/ACM Trans. Netw. 2005, 2, 384-397.

[22] J. van Greunen and J. Rabaey. "Lightweight Time Synchronization for Sensor Networks". Proc. 2nd ACM Int. Workshop on Wireless Sensor Networks and Applications (WSNA '03), pp. 11-19, San Diego, California, Sept. 2003.

[23] D. Djenouri, L. Khelladi, N. Badache, « A survey of security issues in mobile ad hoc and sensor networks», IEEE Communications Surveys and Tutorials Journal, Page(s): 2-29, 2005.

[24] Jaydip Sen. A Survey on Wireless Sensor Network Security. International Journal of Communication Networks and Information Security (IJCNIS). Vol. 1, No. 2, August 2009.
J. Walters, Z. Liang, W. Shi, and V. Chaudhary. « Wireless Sensor Network Security: A Survey». Security in Distributed, Grid, and Pervasive Computing Yang Xiao,(Eds.); CRC Press, 2006.

[25] J. Newsome, R. Shi, D. Song and A. Perrig. The 2ybil attack in sensor networks: Analysis and defenses. In Proceedings of IEEE International Conference on Information Processing in Sensor Networks (IPSN 2004), April 2004.

[26] Y.C. Hu, A. Perrig, and D.B. Johnson. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. In Proceedings of INFOCOM 2003, April 2003.

[27] www.securiteinfo.com, « Le Grand Livre de SecuriteInfo.com », 19 Février 2004.

[28] Emmanuel. Bresson, «Cryptographie : chiffrement par flot», Séminaire de la cryptographie, Page(s) : 22-34, Laboratoire de cryptographie, Université de Paris XII, 2001/2002.

[29] D. Baker, H. X. Mel, «La cryptographie décryptée», Livre, Nombre de Pages : 413, Edition Campus Press, 2001.

[30] G. Gaubatz, et al. " Public Key Cryptography in Sensor Networks-Revisited", ESAS '04 : 1st European Wksp, Security in Ad-Hoc and Sensor Networks, 2004.

[31] Krzysztof Piotrowski et al. "How Public Key Cryptography Influences Wireless Sensor Node Lifetime". SASN'06, Alexandria, Virginia, USA, October 30, 2006.

[32] A. Liu and P. Ning. "TinyECC : Elliptic Curve Cryptography for sensor networks (version 0.3)", February, 2007, available at http : / / discovery.csc.ncsu.edu / software / TinyECC /

[33] Haodong Wang, Bo Sheng and Qun Li, "Elliptic curve cryptography-based access control in sensor networks", Int. J. Security and Networks, Vol. 1, Nos. ¾, 2006.

[34] A. Boukerche et D. Turgut. Taxonomy of Secure Time Synchronization Algorithms for Wireless Sensor Networks". Wiley & Sons, pp. 503-520, 2008.

[35] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In Proceedings of IEEE INFOCOM 2002, June 2002.

[36] S. Mishra and A. Nasipuri. An adaptive low power reservation based mac protocol for wireless sensor. In Proceedings of the IEEE International Conference on Performance Computing and Communications, pages 713–736, 2004.

[37] Hohlt, B., Doherty, L., Brewer, E. Flexible Power Scheduling for Sensor Networks. Information Processing in Sensor Networks (IPSN), April 2004, Berkeley, CA.

[38] Coleri, S. PEDAMACS: Power Efficient and Delay Aware Medium Access Protocol for Sensor Networks. M.S. Thesis, UC. Berkeley, December 2002.

[39] K. Holger and A. Willig, "Protocols and Architectures for Wireless Sensor Networks," Copyright 2005 by John Wiley & Sons Ltd., pp. 202, 384, July 2007.

[40] M. Manzo, T. Roosta, and S. Sastry. Time synchronization attacks in sensor networks. In The Third ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN 2005), November 2005, pp. 107–116.

[41] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient authentication and signing of multicast streams over lossy channels. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, May 2000.

[42] S. Ganeriwal, S. Capkun, C. Han, and M. B. Srivastava. Secure time synchronization service for sensor

networks. In Proceedings of 2005 ACM Workshop on Wireless Security (WiSe 2005), pages 97–106, September 2005.

[43] S. Ganeriwal, S. Capkun, and M. B. Srivastava. Secure time synchronization in sensor networks. ACM Transactions on Information and System Security (TISSEC), 11(4):1–35, 2008.

[44] IEEE Computer Society. IEEE 802.15.4: IEEE standard for information technology –telecommunications and information exchange between systems local and metropolitan area networks – specific requirements part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs).

[45] K. Sun, P. Ning, and C. Wang. Secure and resilient clock synchronization in wireless sensor networks. IEEE Journal on Selected Areas in Communications, 24(2), February 2006.

[46] K. Sun, P. Ning, C. Wang, A. Liu, and Y. Zhou, TinySeRSync: Secure and resilient time synchronization in wireless sensor networks. In Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06), November 2006, pp. 36–42.

[47] B. Parno, A. Perrig, and V. Gligor. Distributed detection of node replication attacks in sensor networks. In IEEE Symposium on Security and Privacy, May 2005, pp. 49–63.

[48] SmartRF CC2420 Datasheet (rev 1.3), 2005-10-03. http: / / www.chipcon.com / files / CC2420 Data Sheet 1 3.pdf.

[49] D. Liu, P. Ning, and W. Du, "Detecting malicious beacon nodes for secure location discovery in wireless sensor networks," in Proc. 25th Int. Conf. Distrib. Comput. Syst., Jun. 2005, pp. 609–619.

[50] X. Du, M. Guizani, Y. Xiao and H. Chen. "Secure and Efficient Time Synchronization in Heterogeneous Sensor Network". IEEE *Transactions on Vehicular Technology*, Vol. 57, no. 4, pp. 2387-2394, Jul. 2008.

[51] X. Du, M. Guizani, Y. Xiao, and S. Ci, H.H. Chen, "A Routing Driven Elliptic Curve Cryptography Based Key Management Scheme for Heterogeneous Sensor Networks," IEEE Transactions on Wireless communications, vol. 8, no. 3, pp. 1223-1229, Mar 2009.

[52] H. Song, S. Zhu, and G. Cao. Attack-resilient time synchronization for wireless sensor networks. Ad Hoc Networks Journal, 5(1):112–125, 2007.

[53] -B. Iglewicz, D.C. Hoaglin, How to detect and handle outliers, ASQC Basic References in Quality Control (1993). –B. Rosner, Percentage points for generalized ESD many outlier procedure, Technometrics.

[54] Y. Yang and Y. Sun. Securing Time-synchronization Protocols in Sensor Networks : Attack Detection and Self-healing. Global Telecommunications Conference. IEEExplore.IEEE.org , 2008.

[55] A. Josang and R. Ismail, "The beta-reputation system." In Proc. The 15th Bled Electronic Commerce Conference, Bled, Slovenia.

[56] X. Hu, T Park, KG Shin. Attack-Tolerant Time-Synchronization in Wireless Sensor Networks. IEEE INFOCOM, 2008.

[57] S. Haykin, Adaptive Filter Theory, 2nd ed. Prentice-Hall, 1991.

[58] -J. Douceur, "The Sybil Attack," in Proceedings of 1st International Workshop on Peer-to-Peer Systems, 2002. –J. Newsome, E. Shi, D. Song, and A. Perrig, "The Sybil Attack in Sensor Nework: Analysis & Defense," in Proceedings of 3rd IEEE / ACM Information Processing in Sensor Networks (IPSN'04), 2004.

[59] D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks," in CCS '03: Proceedings of the 10th ACM conference on Computer and communications security, 2003.

[60] M. Jadliwala, Q. Duan, S. Upadhyaya and J. Xu. "Towards a Theory for Securing Time Synchronization in Wireless Sensor Networks". Copyright 2009 ACM; March 16–18, 2009.

[61] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach, chapter Constraint Satisfaction Problems, pages 137–160. 2 edition, 2002.

[62] A. Tripathi and A. Agarwal. "An Approach towards Time Synchronization Based Secure Protocol for Wireless Sensor Network". Springer-Verlag Berlin Heidelberg 2010. F. Zavoral et al. (Eds.): NDT 2010, Part II, CCIS 88, pp. 321–332, 2010.

[63] M. Rahman, K. El-Khatib. "Secure Time Synchronization for Wireless Sensor Networks Based on Bilinear Pairing Functions". IEEE Transactions on Parallel and Distributed Systems, Manuscript ID.

IEEE, vol. xx, no. xx, pp. xxxx-xxxx, 201x IEEE. 2010.

[64] R. Sakai, K. Ohgishi, and M. Kasahara, "Cryptosystems Based on Pairing," in Symposium on Cryptography and Information Security (SCIS2000), pp. 26–28, Jan. 2000.

[65] A. Abbasi and M. Younis, "A survey on clustering algorithms For wireless sensor networks," Computer Communications, El Sevier Publication, Science Direct, pp. 2826–2841, 2007.

[66] Adam Dunkels, Björn Grönvall, Thiemo Voigt, « Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors », 29 th Annual IEEE International Conference on Local Computer Networks, Pages: 455–462, Swedish Institute of Computer Science, 2004.

[67] Duffy, Cormac J. Sreenan, John Herbert, Utz Roedig, « A Performance Analysis of MANTIS and TinyOS », Technical Report CS-2006-27-11, University College Cork, Ireland, November 2006.

[68] http: / / www1.cse.wustl.edu / ~jain / cse567-11 / ftp / sensor / index.html. « A Survey of Wireless Sensor Network Simulation Tools ».

[69] K. L. Noh, E. Serpedin, and K. Qaraqe. Extension of pairwise broadcast clock synchronization for multicluster sensor networks. EURASIP Journal on Advances in Signal Processing, 2008(Article 71):10, 2008.

[70] K. Cheng, K. Lui, Y. Wu and V. Tam. "A Greedy Distributed Time Synchronization Algorithm for Wireless Sensor Networks". The ICC 2008 proceedings.

[71] http: / / compilers.cs.ucla.edu / avrora /

[72] -http: / / www.5secondfuse.com / tinyos / install.html
-http: / / docs.tinyos.net / tinywiki / index.php / Installing_TinyOS_2.1.1

[73] http: / / sensnet.bsl.cerist.dz /

[74] D. Djenouri. R⁴Sync:Relative Referenceless Receiver / Receiver Time Synchronization in Wireless Sensor Networks. IEEE SIGNAL PROCESSING LETTERS, VOL. 19, NO. 4, April 2012.

[75] A. Ould Kara, « Sécurité des systèmes d'information », Cours, Institut National de formation en Informatique (I.N.I), Algérie, Mai 1999.