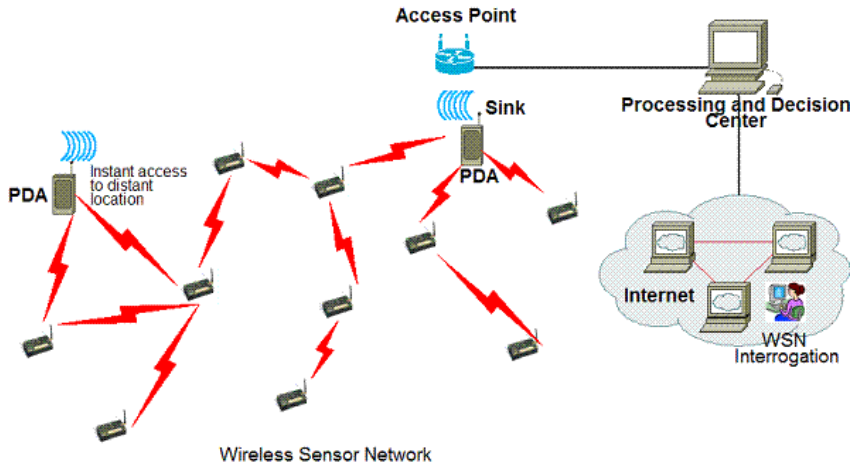
Systèmes Intélligents pour le Transport

Réseaux de Capteurs Sans Fils

Version 1

SIT60



YACINE CHALLAL

17/11/2008

Table des matières

Table des matières *3*

I - Contributeurs et remerciements *9*

II - Réseaux de capteurs sans fils : Architectures et Applications *11*

A. Architectures et Applications....*12*
1. Architecture d'un RCSF....12
2. Applications des RCSF....12
3. Anatomie d'un noeud capteur....13
4. Contraintes de conception des RCSF....14

III - Le système d'exploitation pour RCSF : TinyOS *17*

A. Aperçus sur TinyOS....*17*
1. Pourquoi un nouveau OS pour les "motes"....17
2. La solution TinyOS....18

B. nesC : le langage de programmation de TinyOS....*19*
1. Aperçus sur nesC....19
2. Interface....21
3. Modules....22
4. Configurations....23
5. Notion de tâche et concurrence dans nesC....24
6. Compiler et exécuter une application nesC....26

IV - La communication dans les RCSF *29*

A. Modèle en couches....*29*

B. ZigBee / IEEE 802.15.4....*31*
1. Pourquoi une nouvelle norme ?....31
2. Caractéristiques de ZigBee....32
3. Architecture ZigBee/IEEE 802.15.4....33
4. Mécanisme d'accès au canal....35
5. L'algorithme CSMA/CA....35

C. Protocoles de routage dans les RCSF....*37*
1. Taxonomie des protocoles de routage....37
2. Exemples de protocoles de routage dans les RCSF....38

V - L'économie d'énergie et tolérance aux pannes dans les RCSF *47*

A. La tolérance aux pannes dans les RCSF.....47
1. Les pannes.....47
2. La tolérance aux pannes.....49
B. Approches et solutions tolérantes aux pannes dans les RCSF.....51
1. Classification des solutions de tolérance aux pannes dans les RCSF.....51
2. Approches tolérantes aux pannes dans la couche MAC.....53
3. Approches tolérantes aux pannes dans la couche réseau.....55
VI - La sécurité des échanges dans les RCSF 69
A. La sécurité dans les RCSF : taxonomie des menaces et des solutions.....69
B. La gestion de clés dans les RCSF.....71
1. La fonction de gestion de clés dans les RCSF.....71
2. Schéma aléatoire de pré-distribution de clés de L.ESCHENAUER et D.GLIGOR 75
3. LEAP.....79
C. Sécurité du routage dans les RCSF.....80
1. Le routage dans les RCSF : menaces et solutions.....80
2. INSENS (Intrusion-tolerant routing for wireless sensor networks).....83
3. SecRoute.....86
D. Sécurité de l'agrégation dans les RCSF.....88
1. Attaques sur l'agrégation de données dans les RCSF.....88
2. SAWN (Secure Aggregation for Wireless Networks).....90
3. Protocoles basés sur le cryptage de bout en bout.....92
VII - Ateliers pratiques dans un environnement TinyOS 95
A. Découverte de TinyOS.....95
1. TinyOS : Concepts et définitions.....95
2. Application Blink.....97
3. Compilation et exécution de Blink.....102
4. Exercice.....102
B. Développement d'un protocole de routage (many-to-one) pour RCSF. .102
1. Protocole de routage "many-to-one" simple.....102
2. Construction de l'arbre de routage par relais de la requête de construction 105
3. Loop free routing protocol.....106
4. Evitement des collisions dans la propagation de la requête de construction 106
5. Tolérance aux pannes par reconstruction de l'arbre de routage.....107
Bibliographie 109

Objectifs

- Analyser une architecture de réseaux de capteurs sans fils
- Analyser les protocoles de transport dans les RCSF
- Développer des applications pour réseaux de capteurs sans fils
- Analyser les vulnérabilités et solutions de sécurité pour RCSF
- Analyser les vulnérabiltés et solutions de tolérance aux pannes dans RCSF

Introduction

Depuis leur création, les réseaux de communication sans fil ont connu un succès sans cesse croissant au sein des communautés scientifiques et industrielles. Grâce à ses diverses avantages, cette technologie a pu s'instaurer comme acteur incontournable dans les architectures réseaux actuelles. Le média hertzien offre en effet des propriétés uniques, qui peuvent être résumées en trois points : la facilité du déploiement, l'ubiquité de l'information et le coût réduit d'installation. Au cours de son évolution, le paradigme sans fil a vu naître diverses architectures dérivées, telles que : les réseaux cellulaires, les réseaux locaux sans fils et autres. Durant cette dernière décennie, une architecture nouvelle a vu le jour : les réseaux de capteurs sans fil. Ce type de réseaux résulte d'une fusion de deux pôes de l'informatique moderne : les systèmes embarqués et les communications sans fil. Un réseau de capteurs sans fil (RCSF), ou "Wireless Sensor Network" (WSN), est composé d'un ensemble d'unités de traitements embarquées, appelées "motes", communiquant via des liens sans fil. Le but général d'un WSN est la collecte d'un ensemble de paramètres de l'environnement entourant les motes, telles que la température ou la pression de l'atmosphère, afin de les acheminer vers des points de traitement. Les RCSF sont souvent considérés comme étant les successeurs des réseaux ad hoc. En effet, les RCSF partagent avec les MANET (Mobile Ad hoc NETworks) plusieurs propriétés en commun, telles que l'absence d'infrastructure et les communications sans fil. Mais l'une des différences clés entre les deux architectures est le domaine d'application. Contrairement aux réseaux MANET, qui n'ont pas pu connaître un vrai succès, les RCSF ont su attirer un nombre croissant d'industriels, vu leur réalisme et leur apport concret. En effet, le besoin d'un suivie continu d'un environnement donné est assez courant dans diverses activités de la société. Les processus industriels, les applications militaires de tracking, le monitoring d'habitat, ainsi que l'agriculture de précision ne sont que quelques exemples d'une panoplie vaste et variée d'applications possibles du suivi continu offert par les RCSF. Grâce à ce potentiel riche en applications, les RCSF on su se démarquer de leur origine MANET et attirer de grandes firmes à travers le monde, telles que IBM, Sun, Intel et Philips. Malheureusement, les RCSF ne sont pas parfaits ! A cause de leur faible coût et leur déploiement dans des zones parfois hostiles, les motes sont assez fragiles et vulnérables à diverses formes de défaillances : cassure, faible énergie, ... etc. Ces problèmes rendent les RCSF des systèmes à fragilité innée, qui doit être considérée comme une propriété normale du réseau.

Contributeurs et remerciements

I

Ce support de cours :

- est une collection de travaux d'états de l'art réalisés par des étudiants doctorants et de master : Abdelraouf Ouadjaout, Miloud Bagaa, Noureddine Lasla, Boushra Maala, Fatima Zohra Benhamida
- Reprend des schéma et des contenus rendus publiques par leurs auturs: Chenyang Lu, Kemal Akkaya and Mohamed Younis, Romit Roy Choudhury, Martin Haenggi, David Culler, David Gay, Bob Heile, Samuel Nelson
- reprend des contenus réalisés par des collègues et co-auteurs d'articles publiés ensemble :
  - Hatem Bettahar (UTC)
  - Abdelmadjid Bouabdallah (UTC)
  - Abdelmalik Bachir (Imperial College London)
  - Lyes Khelladi (CERIST-Algérie)

Réseaux de capteurs sans fils : Architectures et Applications

II

Les récentes avancées dans les domaines des technologies sans-fil et électroniques ont permis le développement à faible coût de minuscules capteurs consommant peu d'énergie (solution low-cost et low-power). Ces capteurs ont 3 fonctions :

- Capter des données (de type son, vibration, lumière,...)
- Calculer des informations à l'aide de ces valeurs collectées
- Les communiquer à travers un réseau de capteurs

Un réseau de capteurs est composé d'un nombre souvent très important de nSuds qui sont, soit posés à un endroit précis, soit dispersés aléatoirement (souvent déployés par voie aérienne à l'aide d'avions ou hélicoptères). Ce dispersement aléatoire des capteurs nécessite que le protocole utilisé pour les réseaux de capteurs possède des algorithmes d'auto organisation. Afin de résister aux déploiements, ces capteurs doivent être très solides et de plus, ils doivent aussi pouvoir survivre dans les conditions les plus extrêmes dictées par leur environnement d'utilisation (feu ou eau par exemple). En plus des contraintes environnementales, une contrainte très importante est l'économie de batterie. En effet, un réseau de capteurs ne peut survivre si la perte de nSuds est trop importante car ceci engendre des pertes de communication dues à une trop grande distance entre les capteurs. Donc il est très important que les batteries durent le plus longtemps possible étant donné que dans la plupart des applications ils sont placés aléatoirement (impossible de retourner changer les batteries). Cette utilisation liée à l'autonomie des capteurs (1 année maximum pour les technologies actuelles) fait intervenir un paramètre non négligeable qui est le prix. Aucune application ne serait rentable si le rapport heures d'utilisation / prix était trop élevé. Il a donc été nécessaire d'allier technologie et low-cost. Les réseaux de capteurs peuvent être programmés à un grand nombre de fins, telles que le contrôle d'intrusions, le calcul de températures, le calcul de changements climatiques, la surveillance des déplacements d'animaux (avec récepteurs GPS), surveillance de malades,....

A. Architectures et Applications

1. Architecture d'un RCSF

Définition

Un RCSF est composé d'un ensemble de noeuds capteurs. Ces nSuds capteurs sont organisés en champs « sensor fields » (voir figure suivante). Chacun de ces noeuds a la capacité de collecter des données et de les transférer au nSud passerelle (dit "sink" en anglais ou puits) par l'intermédiaire d'une architecture multi-sauts. Le puits transmet ensuite ces données par Internet ou par satellite à l'ordinateur central «Gestionnaire de taches» pour analyser ces donner et prendre des décisions.

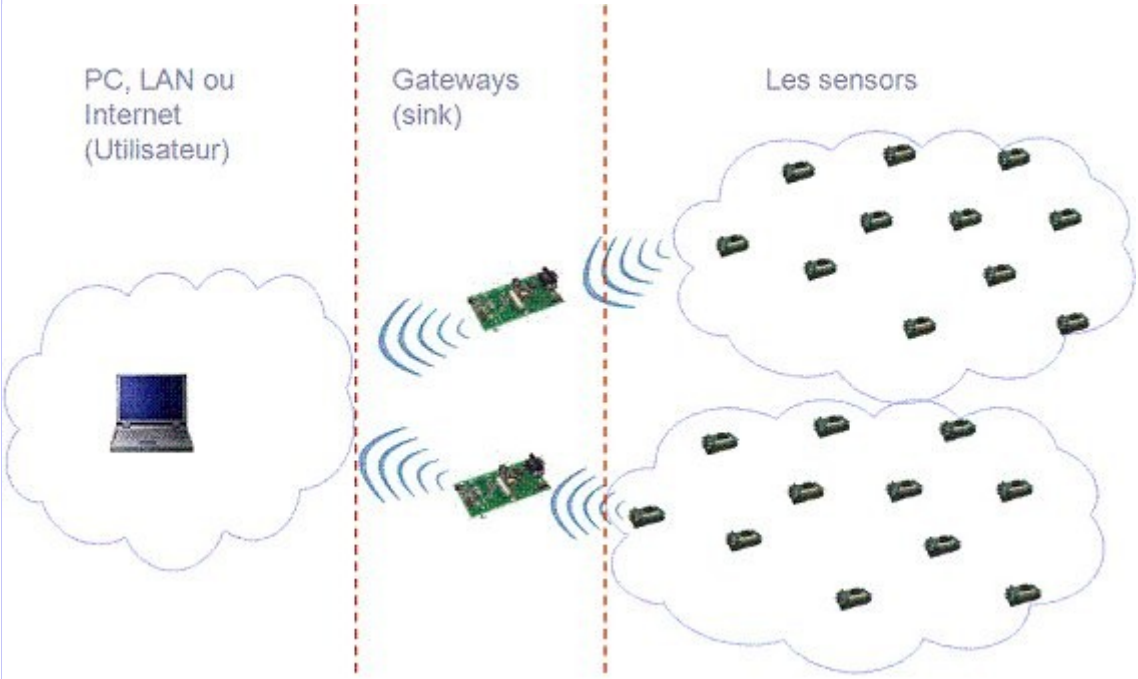


Image 1 : Architecture d'un RCSF

2. Applications des RCSF

Les RCSF peuvent avoir beaucoup d'applications (voir figure suivantes). Parmi elles, nous citons :

- Découvertes de catastrophes naturelles : On peut créer un réseau autonome en dispersant les nSuds dans la nature. Des capteurs peuvent ainsi signaler des événements tel que feux de forêts, tempêtes ou inondations. Ceci permet une intervention beaucoup plus rapide et efficace des secours.
- Détection d'intrusions : En plaçant, à différents points stratégiques, des capteurs, on peut ainsi prévenir des cambriolages ou des passages de gibier sur une voie de chemin de fer (par exemple) sans avoir à recourir à de coûteux dispositifs de surveillance vidéo.
- Applications métier : On pourrait imaginer devoir stocker des denrées nécessitant un certain taux d'humidité et une certaine température (min ou max). Dans ces applications, le réseau doit pouvoir collecter ces différentes

informations et alerter en temps réel si les seuils critiques sont dépassés.

- Contrôle de la pollution : On pourrait disperser des capteurs au-dessus d'un emplacement industriel pour détecter et contrôler des fuites de gaz ou de produits chimiques. Ces applications permettraient de donner l'alerte en un temps record et de pouvoir suivre l'évolution de la catastrophe.
- Agriculture : Des nSuds peuvent être incorporés dans la terre. On peut ensuite questionner le réseau de capteurs sur l'état du champ (déterminer par exemple les secteurs les plus secs afin de les arroser en priorité). On peut aussi imaginer équiper des troupeaux de bétail de capteurs pour connaître en tout temps, leur position ce qui éviterait aux éleveurs d'avoir recours à des chiens de berger.
- Surveillance médicale : En implantant sous la peau de mini capteurs vidéo, on peut recevoir des images en temps réel d'une partie du corps sans aucune chirurgie pendant environ 24h. On peut ainsi surveiller la progression d'une maladie ou la reconstruction d'un muscle.
- Contrôle d'édifices : On peut inclure sur les parois des barrages des capteurs qui permettent de calculer en temps réel la pression exercée. Il est donc possible de réguler le niveau d'eau si les limites sont atteintes. On peut aussi imaginer inclure des capteurs entre les sacs de sables formant une digue de fortune. La détection rapide d'infiltration d'eau peut servir à renforcer le barrage en conséquence. Cette technique peut aussi être utilisée pour d'autres constructions tels que ponts, voies de chemins de fer, routes de montagnes, bâtiments et autres ouvrages d'art.

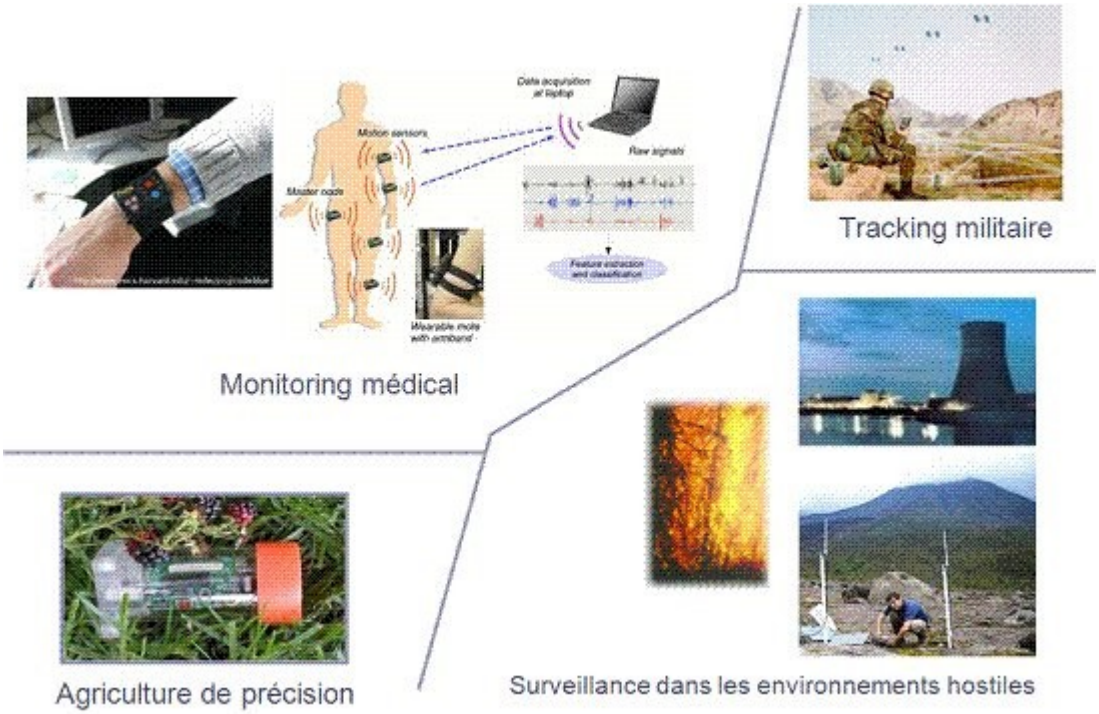


Image 2 : Applications des RCSF

3. Anatomie d'un noeud capteur

Un noeud capteur (dit "mote" en anglais) est composé principalement d'un processeur, une mémoire, un émetteur/récepteur radio, un ensemble de capteurs, et une pile (voir figure suivante). Il existe plusieurs modèles commercialisés dans le marché. Parmi les plus célèbres, les "mote" MICAx et TelosBe de Crossbow (http://

www.xbow.com).

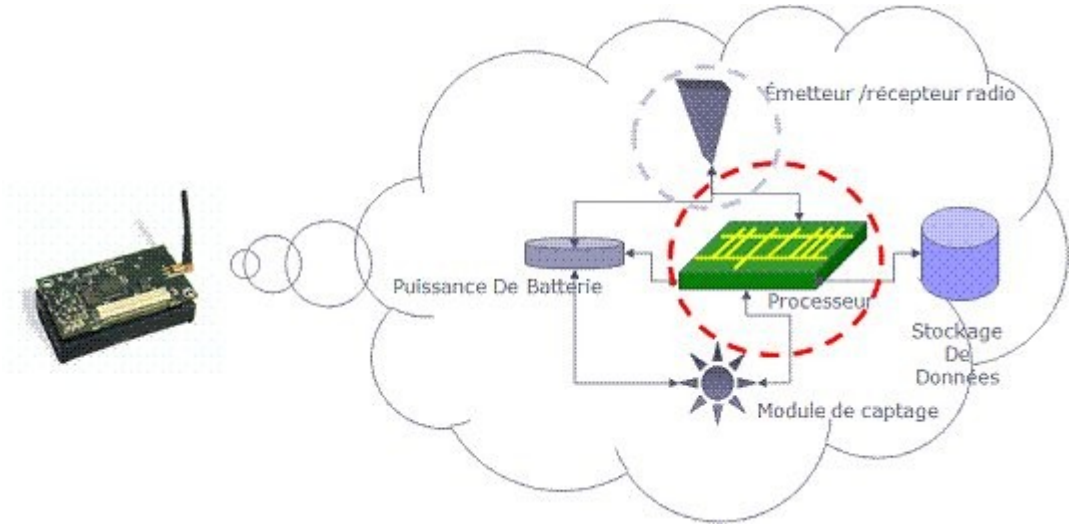


Image 3 : Anatomie d'un noeud capteur

Exemple

Les figures suivantes illustrent les composants d'un noeud capteur TelosB de CrossBow :

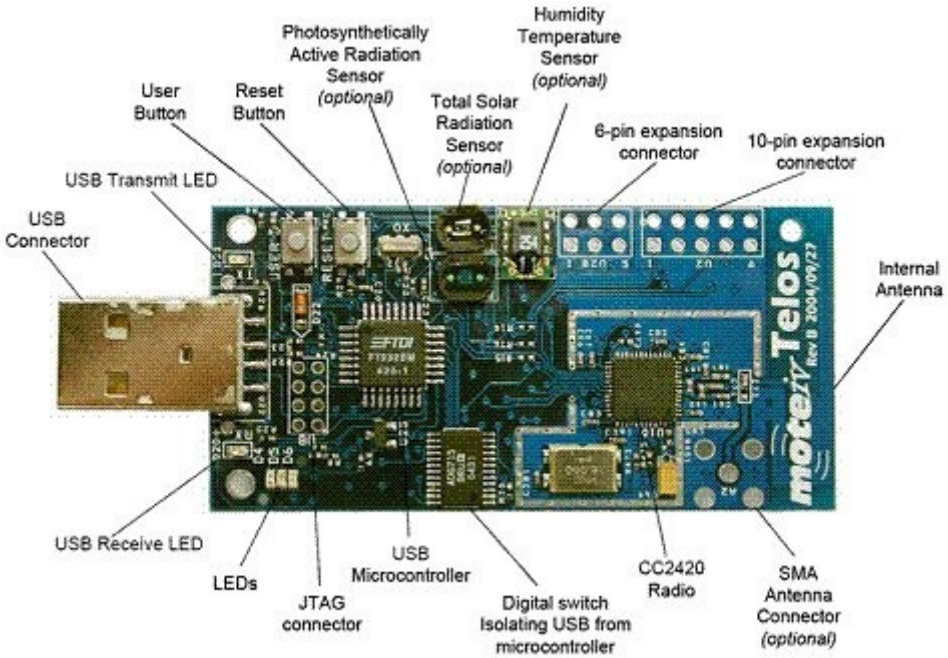


Image 4 : TelosB Recto

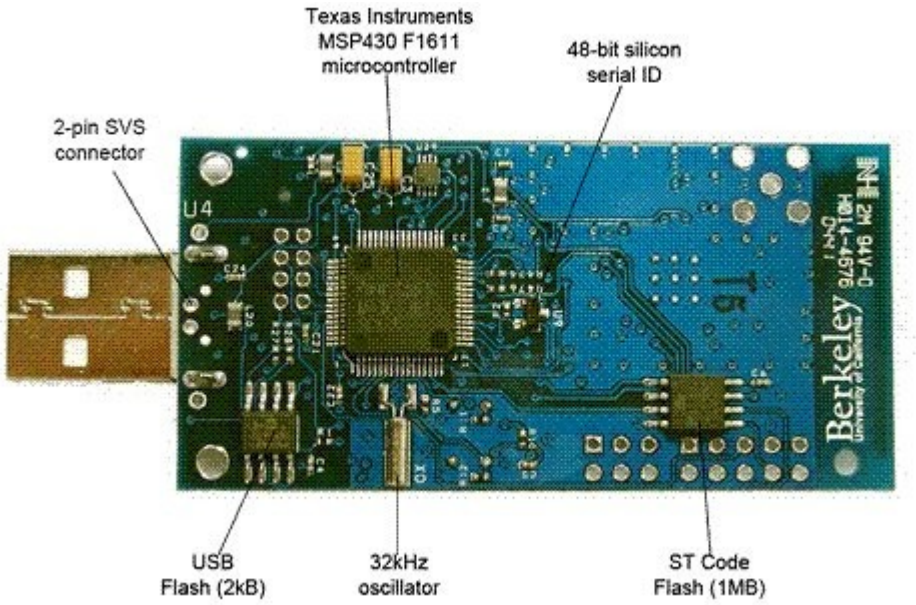


Image 5 : TelosB Verso

4. Contraintes de conception des RCSF

Remarque

Les principaux facteurs et contraintes influençant l'architecture des réseaux de capteurs peuvent être résumés comme suit:

- La tolérance de fautes : Certain nSuds peuvent générer des erreurs ou ne plus fonctionner à cause d'un manque d'énergie, un problème physique ou une interférence. Ces problèmes n'affectent pas le reste du réseau, c'est le principe de la tolérance de fautes. La tolérance de fautes est la capacité de maintenir les fonctionnalités du réseau sans interruptions dues à une erreur intervenue sur un ou plusieurs capteurs.
- L'échelle : Le nombre de nSuds déployés pour un projet peut atteindre le million. Un nombre aussi important de nSuds engendre beaucoup de transmissions inter nodales et nécessite que le puits "sink " soit équipé de beaucoup de mémoire pour stocker les informations reçues.
- Les coûts de production : Souvent, les réseaux de capteurs sont composés d'un très grand nombre de nSuds. Le prix d'un nSud est critique afin de pouvoir concurrencer un réseau de surveillance traditionnel. Actuellement un nSud ne coûte souvent pas beaucoup plus que 1$. A titre de comparaison, un nSud bluetooth, pourtant déjà connu pour être un système low-cost, revient environ à 10$.
- L'environnement : Les capteurs sont souvent déployés en masse dans des endroits tels que des champs de bataille au delà des lignes ennemies, à l'intérieur de grandes machines, au fond d'un océan, dans des champs biologiquement ou chimiquement souillés,... Par conséquent, ils doivent pouvoir fonctionner sans surveillance dans des régions géographiques éloignées.
- La topologie de réseau : Le déploiement d'un grand nombre de nSuds nécessite une maintenance de la topologie. Cette maintenance consiste en trois phases : Déploiement, Post-déploiement (les capteurs peuvent bouger, ne plus fonctionner,...), Redéploiement de nSuds additionnels
- Les contraintes matérielles : La principale contrainte matérielle est la taille

du capteur. Les autres contraintes sont que la consommation d'énergie doit être moindre pour que le réseau survive le plus longtemps possible, qu'il s'adapte aux différents environnements (fortes chaleurs, eau,..), qu'il soit autonome et très résistant vu qu'il est souvent déployé dans des environnements hostiles.

- Les médias de transmission : Dans un réseau de capteurs, les nSuds sont reliés par une architecture sans-fil. Pour permettre des opérations sur ces réseaux dans le monde entier, le média de transmission doit être normé. On utilise le plus souvent l'infrarouge (qui est license-free, robuste aux interférences, et peu onéreux), le bluetooth et les communications radio ZigBee.
- La consommation d'énergie : Un capteur, de par sa taille, est limité en énergie (< 1.2V). Dans la plupart des cas le remplacement de la batterie est impossible. Ce qui veut dire que la durée de vie d'un capteur dépend grandement de la durée de vie de la batterie. Dans un réseau de capteurs (multi-sauts) chaque nSud collecte des données et envoie/transmet des valeurs. Le dysfonctionnement de quelques nSuds nécessite un changement de la topologie du réseau et un re-routage des paquets. Toutes ces opérations sont gourmandes en énergie, c'est pour cette raison que les recherches actuelles se concentrent principalement sur les moyens de réduire cette consommation.

Le système d'exploitation pour RCSF : TinyOS

III

Aperçus sur TinyOS 17
nesC : le langage de programmation de TinyOS 19

TinyOS est un système d'exploitation intégré, modulaire, destiné aux réseaux de capteurs miniatures. Cette plate-forme logicielle ouverte et une série d'outils développés par l'Université de Berkeley est enrichie par une multitude d'utilisateurs. En effet, TinyOS est le plus répandu des OS pour les réseaux de capteurs sans-fil. Il est utilisé dans les plus grands projets de recherches sur le sujet (plus de 10.000 téléchargements de la nouvelle version). Un grand nombre de ces groupes de recherches ou entreprises participent activement au développement de cet OS en fournissant de nouveaux modules, de nouvelles applications,... 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. TinyOS est en grande partie écrit en C mais on peut très facilement créer des applications personnalisées en langages C, NesC, et Java.

A. Aperçus sur TinyOS

1. Pourquoi un nouveau OS pour les "motes"

Rappel : Caractéristiques des OS classiques

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, on peut citer:

- Architecture Multi-thread=>Mémoire importante
- Modèle E/S
- Séparation entre espace noyau et utilisateur
- Pas de contraintes d'énérgie
- Ressources disponibles

Fondamental : Contraintes d'un "mote"

Les OS classiques ne sont pas appropriés aux "motes" (noeuds capteurs), vus que

ces derniers sont caractérisés par :

- Ressources énergétiques basses
- Mémoire limitée
- CPU lente
- Petite taille
- Parallélisme matériel limité
- Communication radio
  - Bande-passante faible
  - Portée radio courte

Propriétés de l'OS souhaité pour les "motes"

- Image mémoire petite
- Efficacité en calcul et consommation d'énergie
- La communication est fondamentale
- Temps-réel
- Construction efficace d'applications

2. La solution TinyOS

Caractéristiques de TinyOS

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

Aperçus générale de TinyOS

- Système d'exploitation pour réseaux de capteurs embarqués
- Ensemble de composants logiciels qui peuvent être reliés ensemble en un seul exécutable sur un mote

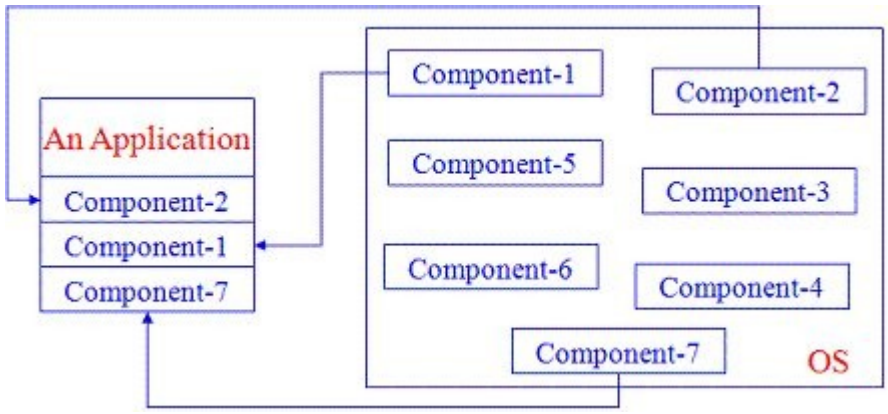


Image 6 : TinyOS : un ensemble de composants logiciels

- Fonctions minimales
  - Deux threads: tâches et handlers d'événements matériels

- Pas de gestion de la mémoire...

Modèle mémoire de TinyOS

- Allocation statique de la mémoire
  - Pas de heap (malloc)
  - Pas de pointeur sur function
  - Pas d'allocation dynamique
- Variables globales
  - Disponibles per-frame
  - Conservation de la mémoire
  - Utilisation de pointeurs
- Variables locales
  - Sauvegardées sur la pile (stack)
  - Déclarées dans une méthode

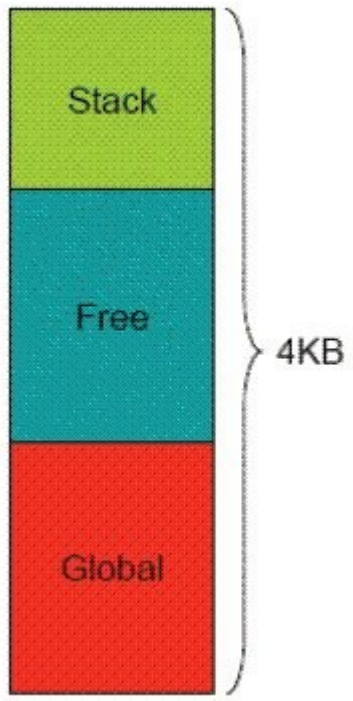


Image 7 : Organisation de la mémoire dans TinyOS

B. nesC : le langage de programmation de TinyOS

1. Aperçus sur nesC

nesC est un langage conçus pour incarner les concepts structurant et le modèle d'exécution de TinyOS. C'est une extension du langage C orientée composant ; il support alors la syntaxe du langage C et il est compilé vers le langage C avant sa compilation en binaire.

Fondamental : Concepts de nesC

- L'unité de code de base de nesC est le composant "component"

- Un composant
  - Exécute des Commandes
  - Lance des Events
  - Dispose d'un Frame pour stocker l'état local
  - Utilise la notion de Tasks pour gérer la concurence
- Un Composant implémente des interfaces utilisées par d'autres composants pour communiquer avec ce composant

Remarque : Composant

Il existe deux types de composants

1. Module : composant implémenté avec du code
2. Configuration : composants reliés ensemble pour former un autre composant

Exemple : Application TinyOS

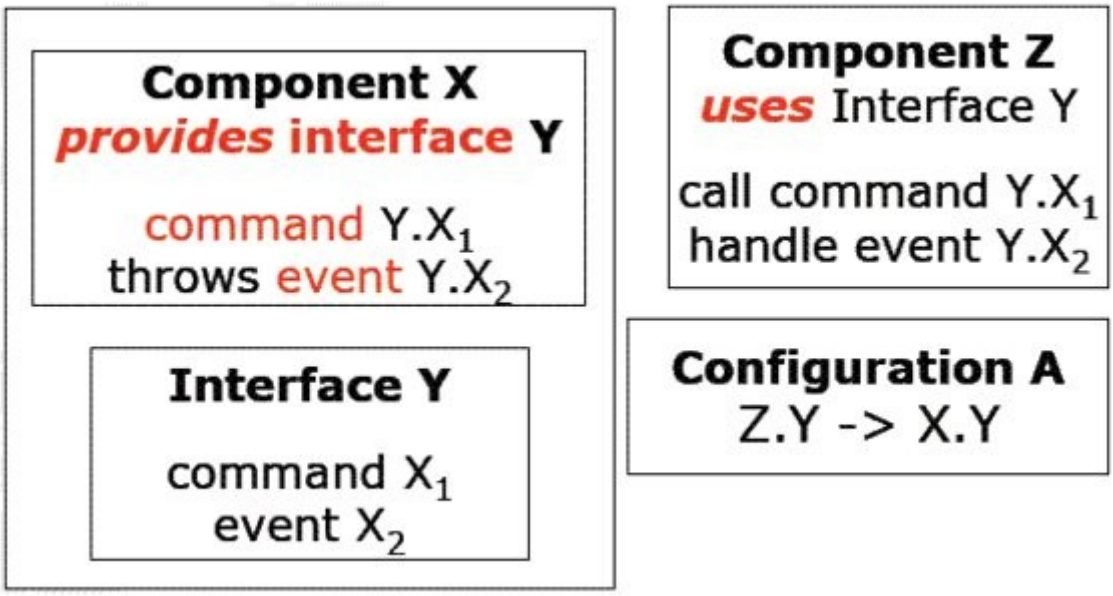


Image 8 : Exemple d'une application TinyOS

Rappel : Résumé vocabulaire

- **Application**: un ou plusieurs composants reliés ensemble pour former un exécutable
- **Composant** : un élément de base pour former une application nesC. Il existe deux types de composants: modules et configurations
  - **Module** : composant qui implémente une ou plusieurs interfaces
  - **Configuration** : composant qui relie d'autres composant ensemble
- **Interface** : définie d'une manière abstraite les interactions entre deux composants

2. Interface

Définition : Interface

Une interface définie les interactions entre deux composants.

Les interfaces sont bidirectionnelles

Elles spécifient un ensemble de fonctions à implémenter par les composants fournisseurs de l'interface (commands), et un ensemble à implémenter par les composants utilisateurs de l'interface (events)

Remarque : command vs. event

Les commands font typiquement des appels du haut vers le bas (des composants applicatifs vers les composants plus proches du matériel), alors que les events remontent les signaux du bas vers le haut.

Exemple : Exemple d'interface

```
interface SendMsg {
command result_t send(uint16_t address, uint8_t length,
TOS_MsgPtr msg);
event result_t sendDone(TOS_MsgPtr msg, result_t success);
}
```

Exemple : Appeler une commande et signaler un événement

Appeler une commande

```
call Send.send(1, sizeof(Message), &msg1);
```

Signaler un event

```
signal Send.sendDone(&msg1, SUCCESS);
```

3. Modules

Définition

Un module est un composant qui implemente une ou plusieurs interfaces et peut utiliser une ou plusieurs interfaces

Syntaxe

```
module Provider
{
  provides interface StdControl;
  provides interface X;
  uses interface Z;
}
implementation
{
  // C code
  ....
}
```

Image 9 : Syntaxe d'un module

Exemple : Schamatisation de la déclaration de trois modules

```
module C1 {
requires interface triangle;
} implementation { ... }
module C2 {
provides interface triangle in;
requires { interface triangle out; interface rectangle
side; }
} implementation { ... }
module C3 {
provides interface triangle;
provides interface rectangle;
} implementation { ... }
```

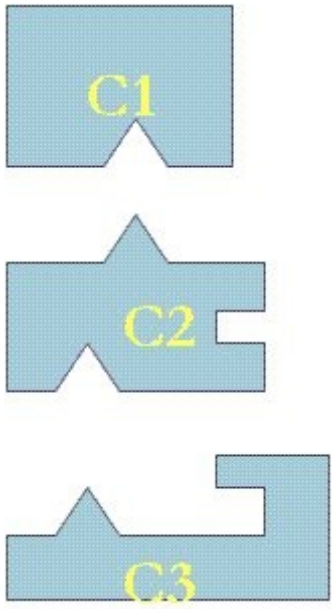


Image 10 : Exemple de déclaration de modules

4. Configurations

Concept de configuration

Dans nesC, deux composants sont reliés ensemble en les connectant (wiring). Les interfaces du composant utilisateur sont reliées (wired) aux mêmes interfaces du composant fournisseur. Il existe 3 possibilités de connexion (wiring statements) dans nesC:

- endpoint1 = endpoint2
- endpoint1 -> endpoint2
- endpoint1 <- endpoint2 (equivalent: endpoint2 -> endpoint1)

Syntaxe

Les éléments connectés doivent être compatibles : Interface à interface, "command" à "command", "event" à "event". Il faut toujours connecter un utilisateur d'une interface à un fournisseur de l'interface.

Exemple : Illustration d'une configuration

```
configuration app { }
implementation {
uses c1, c2, c3;
c1 -> c2; // implicit interface selection.
c2.out -> c3.triangle;
c3 <- c2.side;
}
```

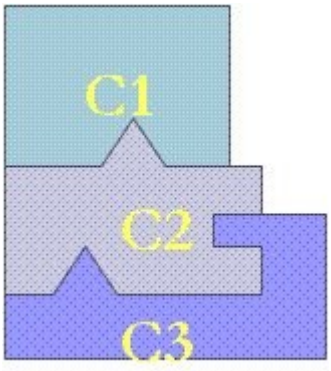


Image 11 : Illustration d'une configuration basée sur la connexion de 3 modules

5. Notion de tâche et concurrence dans nesC

Définition : Tâche

Une tâche est un élément de contrôle indépendant défini par une fonction retournant void et sans arguments:

```
task void myTask() { ... }
```

Les tâches sont lancées en les préfixant par post:

```
post myTask();
```

Commands vs. Events vs. Tasks

Command

- Ne doit pas être bloquante i.e. prend les paramètres, commence le traitement et retourne dans l'application;
- Reporte le travail qui consomme du temps en postant une tâche
- Peut appeler des commandes sur d'autres composants

Event

- Peut appeler des commandes, signaler d'autres événements, poster des tâches mais ne peut pas être signalé par des commandes
- Peut interrompre une tâche mais pas l'inverse.

Task

- Ordonnancement FIFO
- Non préemptive par une autre tâche, préemptive par un événement
- Utilisée pour réaliser un travail qui nécessite beaucoup de calculs
- Peut être postée par une "command" ou un "event".

Définition : Instruction "atomic"

L'instruction "atomic" guarantie que l'exécution de l'instruction se fait comme si aucun autre calcul ne se fait simultanément. Une instruction "atomic" doit être courte. nesC interdit dans une instruction atomique: call, signal, goto, return, break, continue, case, default, label

Exemple : Accès à une ressource critique avec "atomic"

```
bool busy; // global
void f() {
bool available;
atomic {
available = !busy;
busy = TRUE;
}
if (available) do_something;
atomic busy = FALSE;
}
```

6. Compiler et exécuter une application nesC

Compilation et exécution

Le compilateur traîte les fichiers nesC en les convertissant en un fichier C gigantesque. Ce fichiet contiendra l'application et les composants de l'OS utilisés par l'application. Ensuite un compilateur spécifique à la plateforme cible compile ce fichier C. Il deviendra alors un seul exécutable. Le chargeur installe le code sur le "mote" (Mica2, Telos, etc.)

Convention de nommage en nesC

- Extension des fichiers nesC: .nc
- Clock.nc : soit une interface (ou une configuration)
- ClockC.nc : une configuration
- ClockM.nc : un module

Mots clés de nesC

interface	A collection of event and command definitions
module	A basic component implemented in nesC
configuration	A component made from wiring other components
implementation	Contains code & variables for module or configuration
components	List of components wired into a configuration
provides	Defines interfaces provided by a component
uses	Defines interfaces used by a module or configuration
as	Alias an interface to another name
command	Direct function call exposed by an interface
event	Callback message exposed by an interface

Image 12 : Mots clés de nesC

call	Execute a command
signal	Execute an event
post	Put a task on the execution queue
task	A function to be executed in the background

includes	Include a header file

async	For commands, events executed asynchronously
atomic	Guarantees atomic execution of a statement
norace	Eliminates warnings of race conditions nesC detected

Image 13 : Mots clés de nesC pour l'implémentation des modules

La communication dans les RCSF

IV

Modèle en couches	29
ZigBee / IEEE 802.15.4	31
Protocoles de routage dans les RCSF	37

A. Modèle en couches

Le rôle de ce modèle consiste à standardiser la communication entre les composants du réseau afin que différents constructeurs puissent mettre au point des produits (logiciels ou matériels) compatibles. Ce modèle comprend 5 couches qui ont les mêmes fonctions que celles du modèle OSI ainsi que 3 couches pour la gestion de la puissance d'énergie, la gestion de la mobilité ainsi que la gestion des tâches (interrogation du réseau de capteurs). Le but d'un système en couches est de séparer le problème en différentes parties (les couches) selon leur niveau d'abstraction. Chaque couche du modèle communique avec une couche adjacente (celle du dessus ou celle du dessous). Chaque couche utilise ainsi les services des couches inférieures et en fournit à celle de niveau supérieur.

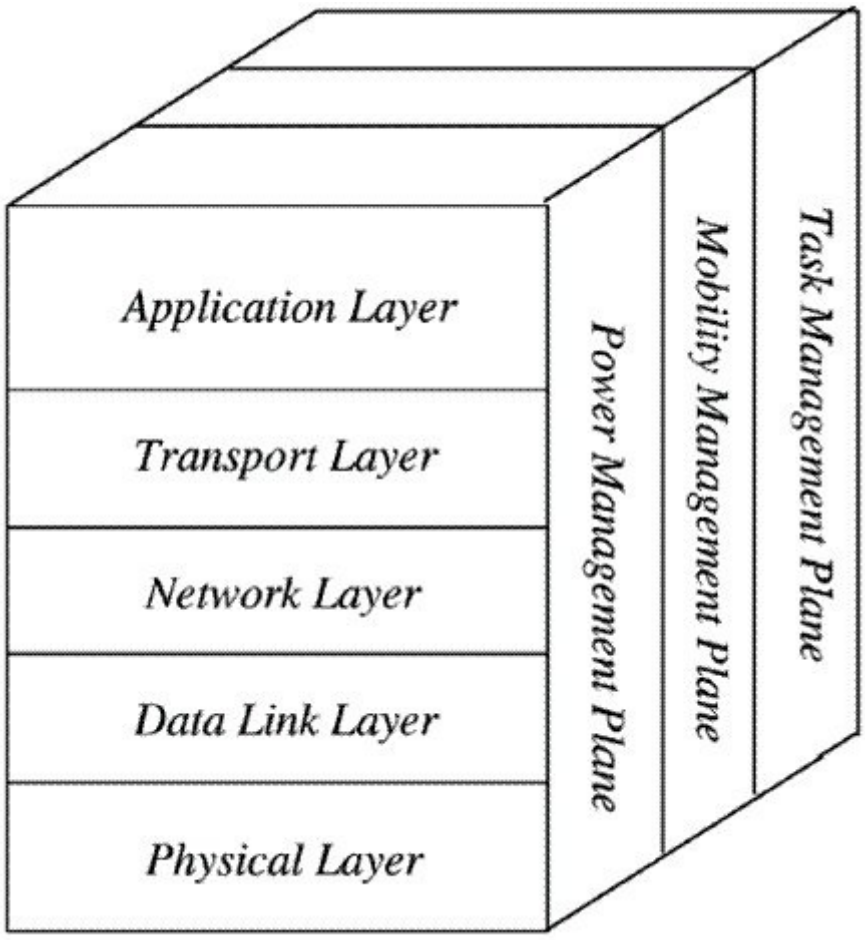


Image 14 : Modèle en couches pour la communication dans les RCSF

Rappel : Rôle des couches

- La couche physique : Spécifications des caractéristiques matérielles, des fréquences porteuses, etc...
- La couche liaison : Spécifie comment les données sont expédiées entre deux noeuds/routeurs dans une distance d'un saut. Elle est responsable du multiplexage des données, du contrôle d'erreurs, de l'accès au media,... Elle assure la liaison point à point et multi-point dans un réseau de communication.
- La couche réseau : Dans la couche réseau le but principal est de trouver une route et une transmission fiable des données, captées, des noeuds capteurs vers le puits "sink" en optimisant l'utilisation de l'énergie des capteurs. Ce routage diffère de celui des réseaux de transmission ad hoc sans fils par les caractéristiques suivantes:
  - il n'est pas possible d'établir un système d'adressage global pour le grand nombre de nSuds.
  - les applications des réseaux de capteurs exigent l'écoulement des données mesurées de sources multiples à un puits particulier.
  - les multiples capteurs peuvent produire de mêmes données à proximité d'un phénomène (redondance).
  - les nSuds capteur exigent ainsi une gestion soigneuse des ressources.

  En raison de ces différences, plusieurs nouveaux algorithmes ont été proposés pour le problème de routage dans les réseaux de capteurs
- La couche transport : Cette couche est chargée du transport des données,

de leur découpage en paquets, du contrôle de flux, de la conservation de l'ordre des paquets et de la gestion des éventuelles erreurs de transmission.

- La couche application : Cette couche assure l'interface avec les applications. Il s'agit donc du niveau le plus proche des utilisateurs, géré directement par les logiciels.

Plans de gestion

Les plans de gestion d'énergie, de mobilité et de tâche contrôlent l'énergie, le mouvement et la distribution de tâche au sein d'un nSud capteur. Ces plans aident les nSuds capteurs à coordonner la tâche de captage et minimiser la consommation d'énergie. Ils sont donc nécessaires pour que les nSuds capteurs puissent collaborer ensemble, acheminer les données dans un réseau mobile et partager les ressources entre eux en utilisant efficacement l'énergie disponible. Ainsi, le réseau peut prolonger sa durée de vie.

- Plan de gestion d'énergie : contrôle l'utilisation de la batterie. Par exemple, après la réception d'un message, le capteur éteint son récepteur afin d'éviter la duplication des messages déjà reçus. En outre, si le niveau d'énergie devient bas, le nSud diffuse à ses voisins une alerte les informant qu'il ne peut pas participer au routage. L'énergie restante est réservée au captage ;
- Plan de gestion de mobilité : détecte et enregistre le mouvement du nSud capteur. Ainsi, un retour arrière vers l'utilisateur est toujours maintenu et le nSud peut garder trace de ses nSuds voisins. En déterminant leurs voisins, les nSuds capteurs peuvent balancer l'utilisation de leur énergie et la réalisation de tâche ;
- Plan de gestion de tâche : balance et ordonnance les différentes tâches de captage de données dans une région spécifique. Il n'est pas nécessaire que tous les nSuds de cette région effectuent la tâche de captage au même temps ; certains nSuds exécutent cette tâche plus que d'autres selon leur niveau de batterie.

B. ZigBee / IEEE 802.15.4

1. Pourquoi une nouvelle norme ?

Historique

Un des soucis majeurs traités dans les communications sans fils, jusqu'à un futur proche, est la bande-passante. Néanmoins, quelques applications, comme: home automation, sécurité, agriculture, monitoring etc. relaxent le besoin en bande-passante pour un besoin de moindre coût et de basse consommation d'énergie. Les standards existants n'étaient pas appropriés à cause de leur compléxité, leur coût élevé, et la consommation d'énergie induite.

Alliance Zigbee

Dans ce contexte, l'alliance ZigBee est née : c'est une association de companies travaillant ensemble pour développer un standard global et ouvert pour les communications sans fils avec un coût réduit et une basse consommation de l'énergie.

Applications types

- Home automation : Chauffage, ventilation, air conditionné, sécurité,

éclairage, et contrôle d'objets.

- Industrielles : Détection de situation d'urgence, contrôle de machines
- Automotive : Contrôle de la pression des pneus, etc.;
- Agriculture : Mesure de l'humidité du sol, détection de situations pour l'usage des intrants, mesure de la salinité du sol, etc.
- Autres : Contrôle d'équipement électronique, communication entre PC et périphériques, etc.

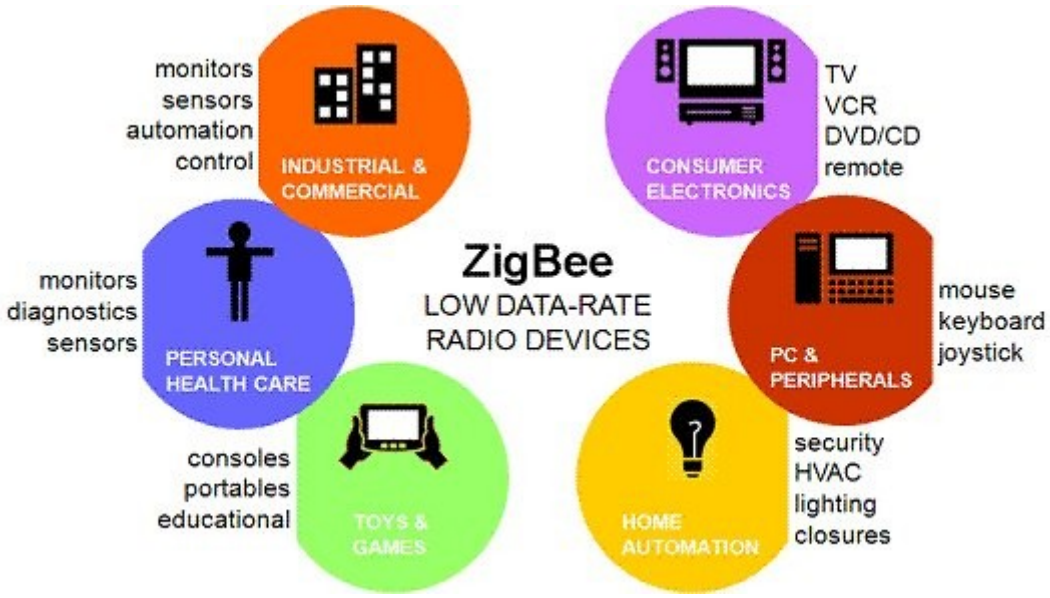


Image 15 : Applications de ZigBee

Remarque : Débit

Ces applications nécessitent généralement un débit de 115.2 kb/s jusqu'à moins de 10 kb/s.

2. Caractéristiques de ZigBee

Carcactéristiques

Les objectifs visés par ZigBee peuvent être résumés dans les points suivants

- Usage sans restrictions géographiques
- Pénétration à travers les murs et plafonds
- Installation automatique/semi-automatique
- Possibilité de rajouter/retirer des dispositifs
- Coût avantageux
- Débit : 10kbps-115.2kbps
- Portée radio: 10-75m
- Jusqu'à 65k noeuds par réseau
- Jusqu'à 100 réseaux co-localisés
- Jusqu'à 2 ans de durée de vie de batterie standards Alkaline

Positionnement

La figure suivante illustre le positionnement de ZigBee par rapport à d'autres standards :

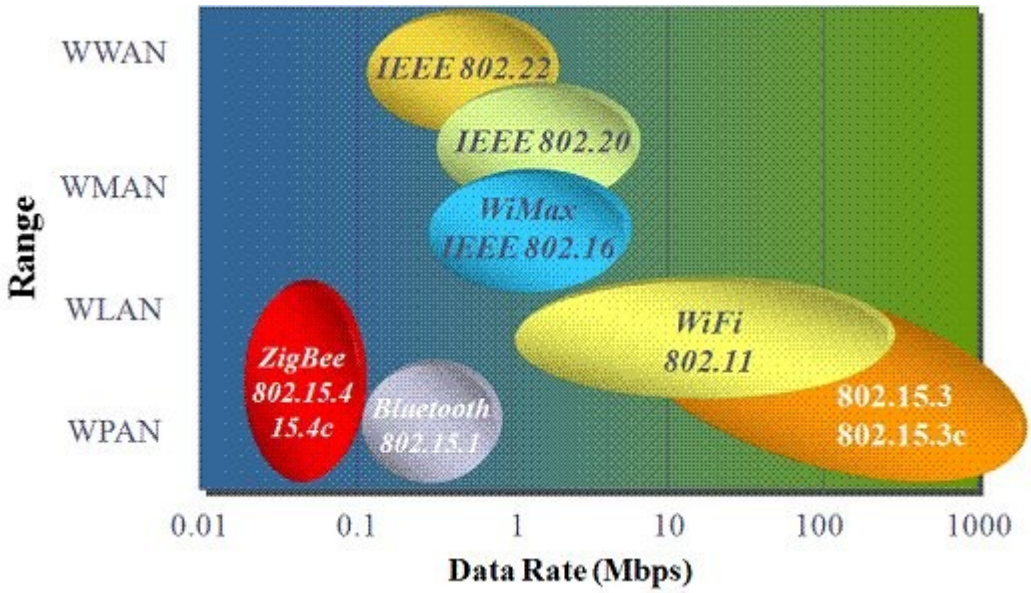


Image 16 : positionnement de ZigBee

3. Architecture ZigBee/IEEE 802.15.4

Le IEEE 802.15.4 Working Group a comme rôle la définition des couches basses: MAC et PHY. La ZigBee Alliance qui regroupe plus de 50 compagnies avait pour rôle la définition des couches supérieures: du routage à l'application. La figure suivante illustre l'architecture en couches de ZigBee/IEEE 802.15.4

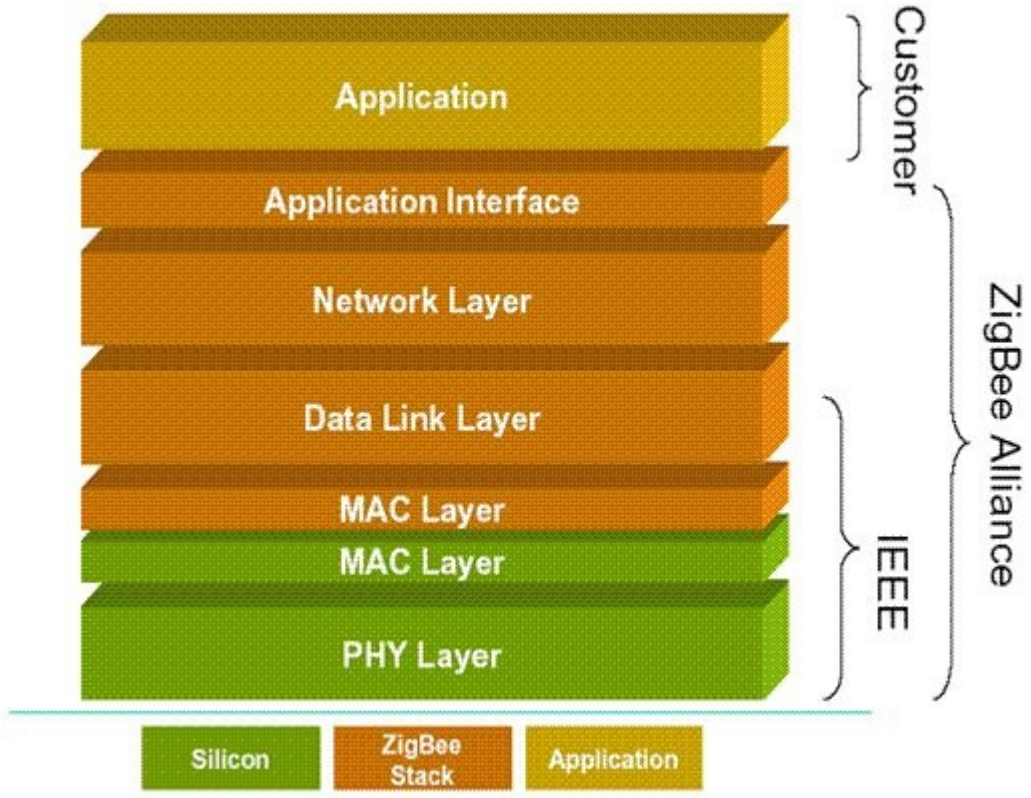


Image 17 : Architecture ZigBee/IEEE 802.15.4

Remarque : Aperçus de MAC IEEE 802.15.4

La couche MAC IEEE 802.15.4 :

- Utilise deux modes d'adressage IEEE 64-bit & 16-bit
- Accès canal CSMA-CA
- Utilise une structure de trame simple
- Permet d'utiliser le mécanisme de beaconing; réveil périodique, vérification de l'arrivé d'un beacon
- Economise l'énergie à travers la mise en veille entre deux beacons, et les noeuds ne devant pas router ou recevoir les données aléatoirement peuvent se mettre en veille.
- Assure une transmission fiable de données
- Offre une sécurité AES-128

Remarque : Network Beacon

- Identifie le réseau
- Décrit la structure de la super-trame
- Indique la présence de données
- Présent uniquement lorsque le réseau est actif
- Il est optionnel

4. Mécanisme d'accès au canal

IEEE 802.15.4 utilise CSMA/CA décliné en deux versions selon la configuration du réseau:

- Si le "beaconing" n'est pas utilisé, IEEE 802.15.4 utilise CSMA/CA sans slots
- Si le "beaconing" est utilisé, IEEE 802.15.4 utilise CSMA/CA avec slots et la structure super-trame

Remarque

La super-trame est composée de deux parties (voir figure suivante):

- Inactive: toutes les stations dorment
- Active: Période active composée de 16 slots. On distingue deux parties dans les 16 slots
  - Contention access period (CAP)
  - Contention free period (CFP)

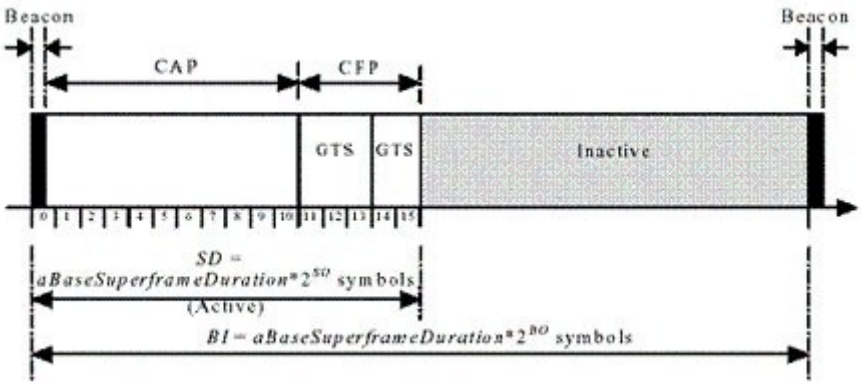


Image 18 : Structure de la super-trame

Deux paramètres déterminent la longueur de la super-trame:

- SO: détermine la longueur de la période active
- BO: détermine la longueur d'une période de beacon.

Dans CFP, un GTS (Guaranteed Time Slots) peut être constitué de plusieurs slots, attribués à un seul noeud, pour transmission (t-GTS) ou réception (r-GTS).

Dans CAP, le concept de slots n'est pas utilisé. CAP est divisé en de plus petits slots de contention. Chaque slot de contention a une longueur de 20 symboles. C'est la plus petite unité de contention backoff. Dans ce cas, les noeuds entrent en contention suivant CSMA/CA avec slots.

5. L'algorithme CSMA/CA

Chaque noeud doit maintenir trois variables pour chaque tentative de transmission

- NB: nombre de fois l'algorithme CSMA/CA fait backoff durant la tentative de transmission en cours
- BE (Backoff Exponent) : détermine le nombre de périodes backoff qu'un noeud doit attendre avant de tenter d'accéder au canal.
- CW (Contention Window) : Longueur de la fenêtre de contention ; le nombre de slots backoff sans aucune activité de canal avant de commencer la transmission. CW est Initialisé à 2 et remis à 2 si le canal est détecté occupé. Une station doit détecter 2 CCA (Clear Channel Activity) avant d'entrer en contention.

Organigramme de CSMA/CA

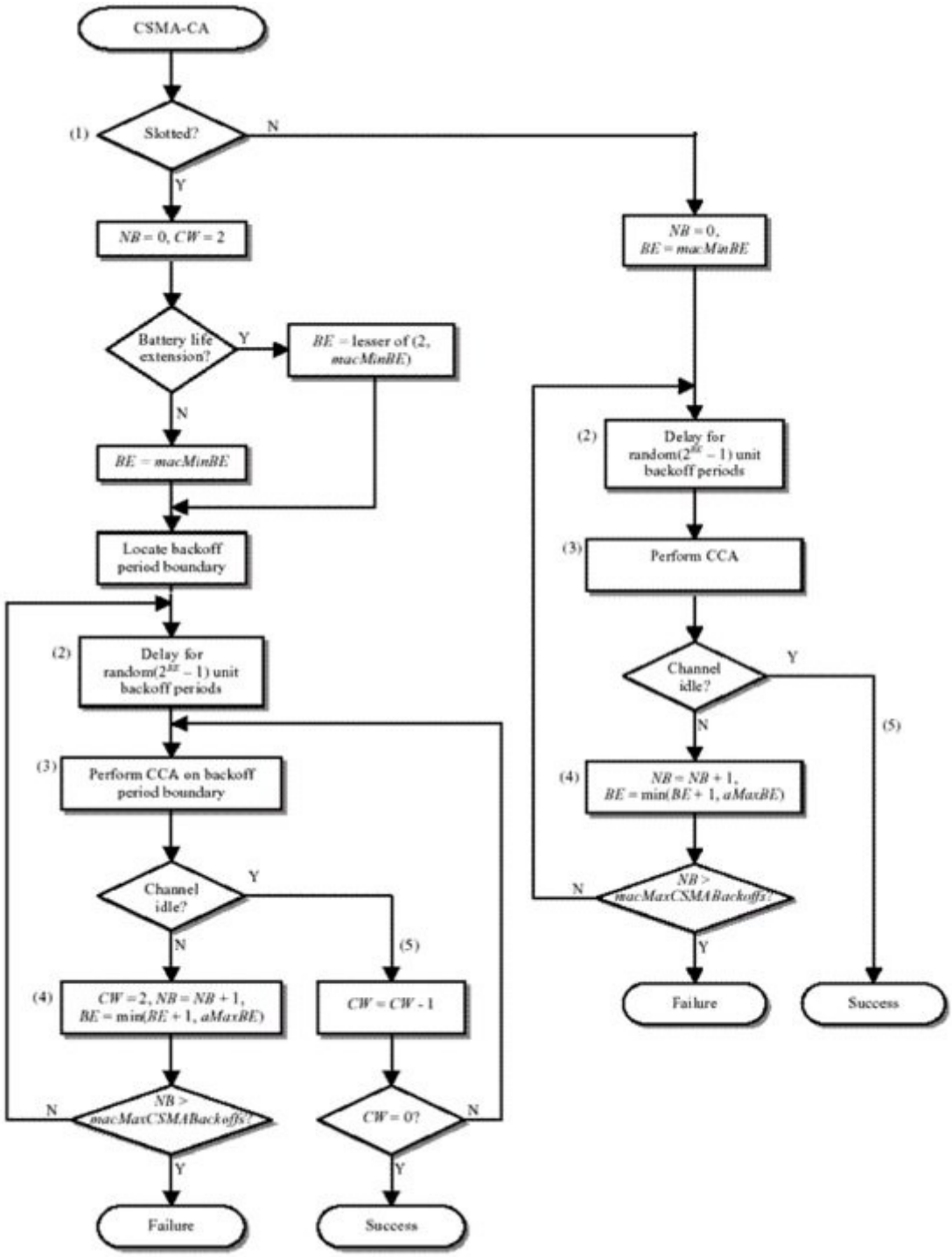


Image 19 : Orgaigramme de CSMA/CA

C. Protocoles de routage dans les RCSF

La propagation et la délivrance des données dans un RCSF représente la fonctionnalité la plus importante du réseau. Elle doit prendre en considération toutes les caractéristiques des capteurs afin d'assurer les meilleures performances du système : durée de vie, fiabilité, temps de réponse, ... etc. Vu la spécificité des

RCSF, un nombre important de recherches sont orientées vers une violation du découpage en couches protocolaires indépendantes, et introduisent la notion de cross layer optimization. Par exemple, en utilisant des mécanismes d'agrégation, les routeurs intermédiaires doivent accéder à la donnée afin d'établir des résumés des lectures de la région.

1. Taxonomie des protocoles de routage

Taxonomie

Récemment, les protocoles de routage pour les RCSF ont été largement étudiés, et différentes études ont été publiées. Les méthodes employées peuvent être classifiées suivant plusieurs critères comme illustré sur la figure suivante :

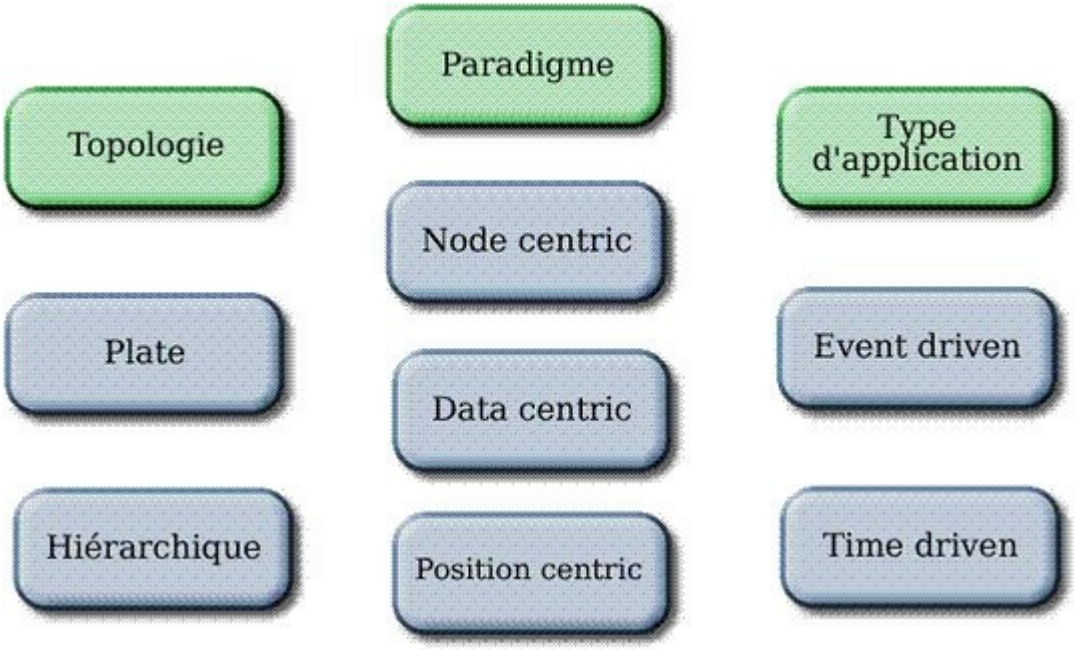


Image 20 : Classification des protocoles de routage pour RCSF

Topologie du réseau

La topologie détermine l'organisation des capteurs dans le réseau. Il existe deux principales topologies dans les protocoles de routage pour les RCSF.

- Topologie plate : dans une topologie plate, tous les noeuds possèdent le même rôle. Les noeuds sont semblables en termes de ressources.s
- Topologie hiérarchique : afin d'augmenter la scalabilité du système, les topologies hiérarchiques ont été introduites en divisant les noeuds 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.

Paradigme de communication

Dans les RCSF, il existe trois paradigmes de communication :

- Node centric : ce paradigme est celui employé dans les réseaux conventionnels, où les communications se basent sur l'identification des noeuds participants, qui se fait à l'aide d'adresses IP.
- Data centric : dans un RCSF, la donnée est plus importante que le noeud lui-même, ce qui rend son identification inutile. Dans le paradigme data centric,

les communicants sont identifiés par leurs données, et donc tout le système (routage, interrogation, . . . etc) doit être régit par cette propriété. Ainsi, le système peut être vu comme une base de données distribuée, où les noeuds forment des tables virtuelles, alimentées par les données captées.

- Position centric : dans cette approche, les positions des noeuds représentent le moyen principal d'adressage et de routage. Dans certaines applications, il est plus intéressant d'interroger le système en utilisant les positions des noeuds, que leurs adresses IP. Dans ce cas, le routage s'effectue grâce à des techniques géométriques afin d'acheminer l'information d'une zone géographique vers une autre.

Type d'application

La méthode de captage des données dans un RCSF dépend de l'application et de l'importance de la donnée. De ce fait, les RCSF peuvent être catégorisés comme time-driven ou event-driven.

- Application time-driven : un réseau time-driven est approprié pour des applications qui nécessitent un prélèvement périodique des données. Par exemple, cela est utile dans des applications de monitoring (feu, météo) afin d'établir des rapports périodiques.
- Application event-driven : dans des applications temps réel, les capteurs doivent réagir immédiatement à des changements soudains des valeurs captées. Un prélèvement périodique des données est inadapté pour ce type de scénarios. Pour cela, le protocole doit être réactif et doit donner des réponses rapides à l'occurrence d'un certain nombre d'évènements.

2. Exemples de protocoles de routage dans les RCSF

a) Contraintes de conception de protocoles de routage pour RCSF

Plusieurs contraintes doivent êtres prises en compte dans la conception des RCSF :

- Limitations :
  - Contraintes d'énérgie : toutes les couches doivent tenir compte de la limitation d'énergie pour maximiser de la durée de vie du réseau.
  - Bande-passante
- Absence d'adressage global
- Données redondantes
- Réseau à sources multiples / destination unique
- Gestion des ressources
- Capacité de calcul
- Stockage

b) SPIN

Heinzelman et al. ont proposé une famille de protocoles appelée SPIN (Sensor Protocols for Information via Negotiation), reposant sur un modèle de négociation afin de propager l'information dans un réseau de capteurs. Le but de SPIN est de pallier aux problèmes de l'inondation, qui sont :

- L'implosion due à la duplication inutile des réceptions d'un même message.
- Le chevauchement lié au déploiement dense des capteurs. En utilisant l'inondation, les capteurs d'une zone émettrons tous la même donnée (ou presque).

- L'ignorance des ressources, car d'inondation ne prend pas en considération les ressources des noeuds.

Ces trois problèmes affectent grandement la durée de vie et les performances du réseau. Pour les résoudre, SPIN adopte deux principes :

- La négociation : pour éviter le problème d'implosion, SPIN précède l'émission d'une donnée par sa description, en utilisant la notion de méta-données. Le récepteur aura le choix par la suite d'accepter la donnée ou non. Ce mécanisme permet aussi de régler le problème de chevauchement.
- L'adaptation aux ressources : d'une manière continue, les noeuds contrôlent leur niveau d'énergie. Le protocole SPIN accommode son exécution suivant l'énergie restante du capteur, et modifie en conséquence le comportement du noeud.

Méthode : Fonctionnement de SPIN

Les communications dans SPIN se font en trois étapes :

- Lorsqu'un noeud veut émettre une donnée, il émet d'abord un message ADV contenant une description de la donnée en question.
- Un noeud recevant un message ADV, consulte sa base d'intérêt. S'il est intéressé par cette information, il émet un message REQ vers son voisin.
- En recevant un message REQ, l'émetteur transmet à l'intéressé la donnée sous forme d'un message DATA.

La figure suivante illustre ces trois étapes :

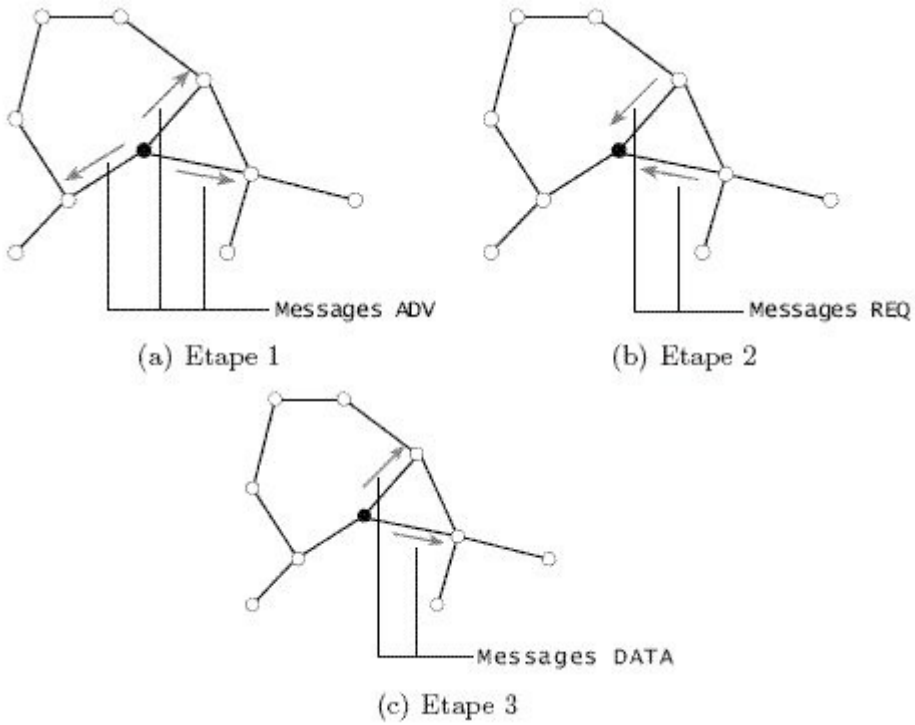


Image 21 : Fonctionnement de SPIN

Remarque

Lorsque le noeud s'aperçoit que son énergie est descendu sous un certain seuil, il change son mode de fonctionnement, et ne répond à aucun message ADV.

c) Directed Diffusion

Aperçu

Directed Diffusion est un protocole de propagation de données, permettant d'utiliser plusieurs chemins pour le routage d'information. Le puits diffuse un intérêt sous forme de requête, afin d'interroger le réseau sur une donnée particulière. Il se base sur le modèle publish/subscribe. DD repose sur quatre éléments : nomination des données, propagation des intérêts et établissement des gradients, propagation des données et renforcement des chemins.

i La nomination de données

Nomination des données

L'adressage dans DD utilise un schéma attribut-valeur afin de décrire les intérêts et les rapports de données.

Exemple : Nomination des données

Par exemple, dans une application de protection de forêts, une requête peut être effectuée sous cette forme :

```
Type = GetTemperature
Zone = [100, 100, 120, 120]
Interval = 10 ms
Durartion = 1 mn
```

Une réponse d'un capteur pourra être formulée ainsi :

```
Type = GetTemperature
Location = (110, 115)
Temperature = 32
Timestamp = 11:32:10
```

ii Propagation des intérêts et établissement des gradients

Lorsqu'un puits requiert une donnée du réseau, il propage un intérêt, contenant sa description ainsi que le débit d'information désiré. Initialement, le puits spécifie un grand intervalle, dans un but d'exploration. Cela permet d'établir les gradients et de découvrir d'éventuelles sources, sans pour autant encombrer le réseau.

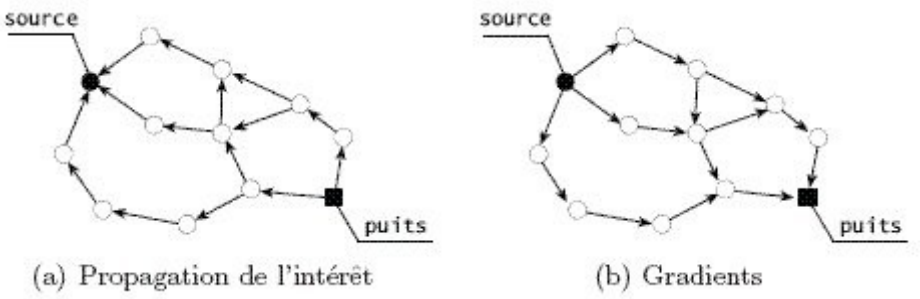


Image 22 : Propagation des intérêts et établissement des gradients

Méthode : Propagation des l'intérêts

Afin de propager l'inéerêt, DD emploie l'inondation globale du réseau. Chaque noeud maintient localement un cache d'intérêt contenant les informations suivantes :

- La description de l'intérêt, en utilisant le schéma de nomination.

- Un ensemble de gradients.

Définition : Gradient

Un gradient est un vecteur représentant l'intérêt. Il est caractérisé par une direction et une amplitude : la direction est modélisée par le voisin émetteur de l'intérêt, et l'amplitude est représentée par le débit de données. En plus, chaque entrée contient un champ limitant la durée de validité du gradient.

Méthode : Etablissement des gradients

Lorsqu'un noeud reçoit un intérêt, il parcourt son cache :

- Si le cache ne contient aucune entrée relative à l'intérêt reçu, une nouvelle entrée est créée avec un gradient vers le voisin émetteur.
- Dans le cas contraire, le noeud recherche un gradient vers le voisin émetteur, et met à jour en conséquence l'entrée en question.

Après le traitement du cache, le noeud relaie l'intérêt vers ses voisins. La méthode la plus simple est d'utiliser l'inondation.

iii Propagation des données

Lorsque l'intérêt atteint les sources ciblées, les capteurs commencent la récolte d'information. Pour un intérêt donné, un capteur calcule le débit le plus élevé et prélève les données en conséquence. En consultant les gradients relatifs à l'intérêt, le noeud détermine les prochains sauts vers les puits (chacun avec son propre débit).

Lorsqu'un noeud reçoit une donnée, il recherche un intérêt équivalent dans son cache. Si aucune entrée n'est trouvée, le paquet est supprimé. Dans le cas contraire, en consultant la liste des gradients, le noeud relaie la donnée vers ses voisins, suivant le débit de chacun d'eux.

Remarque : Evitement des boucles

Avant de relayer une donnée à ses voisins, un noeud utilise son cache de données. Ce cache enregistre les données récemment émises par les voisins. Cela évite la création de boucles, en supprimant les données déjà rencontrées.

iv Renforcement des chemins

Renforcement positif

Lorsque le puits reçoit les premières données, il renforce le chemin vers le voisin émetteur, en augmentant le débit de captage. Cela permet de clôturer la phase d'exploration, et d'entamer la phase de récolte d'information. Le renforcement ne doit pas s'arrêter au niveau des voisins du puits, mais doit se propager éventuellement jusqu'aux sources. Pour ce faire, lorsqu'un noeud reçoit un message de renforcement, il consulte son cache d'intérêt. Si le débit spécifié dans le message est plus grand que tous les autres débits des gradients présents, le noeud doit renforcer un de ses voisins. Le voisin est choisi en utilisant le cache de données.

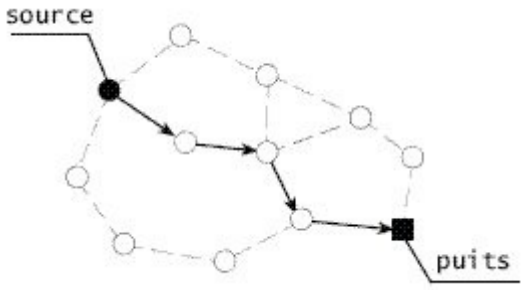


Image 23 : Renforcement des chemins

Renforcement négatif

Dans le cas de panne d'un lien (perte de paquet, débit réduit, etc.) le puits peut envoyer un renforcement négatif sur le chemin en panne en spécifiant le débit de base (exploratoire), et en procédant à un renforcement positif d'un chemin alternatif.

d) MCFA : Minimum Cost Forwarding Algorithm

Aperçus général

Ye et al. ont proposé l'algorithme MCFA (Minimum Cost Forwarding Algorithm), recherchant un chemin minimal entre la source et le puits, tout en considérant les limites des réseaux de capteurs. Le protocole vise à atteindre trois principaux buts :

- L'optimalité : en acheminant les données sur des chemins à coût minimum.
- La simplicité : qui se traduit par une faible consommation en mémoire, et la non nécessité d'une identification des noeuds.
- La scalabilité : étant donnée la faible consommation en mémoire et l'absence d'identificateur de noeuds, le protocole peut être utilisé pour un grand nombre de noeuds. En plus, la phase de construction des routes ne consomme qu'un message par capteur.

Chaque noeud maintient une variable de coût, qui détermine le coût minimal vers le puits sur le chemin optimal. Plusieurs mesures peuvent être employées, suivant l'application voulue : nombre de sauts, consommation d'énergie, . . . etc. L'algorithme se déroule en deux phases : calcul des coûts, relais des paquets.

i Etablissement des valeurs de coût

Valeur locale

Une solution simple pour la détermination des valeurs locales des coûts est d'utiliser l'inondation. Initialement, le puits émet un message ADV contenant un coût nul. Tous les autres noeuds initialisent leur coût à une valeur infinie. Lorsqu'un noeud reçoit un message ADV, il vérifie si la valeur reçue additionnée au coût du lien est plus petite que la valeur locale. Dans ce cas, le noeud met à jour sa valeur locale et émet un nouveau message ADV.

Remarque : Optimum global vs. consommation d'énergie

Pour prendre en compte les contraintes d'énergie des capteurs, MCFA utilise une méthode qui consomme moins de messages de contrôle. En effet, le problème de l'inondation est la prise de décision en tenant en compte seulement des optimums locaux. Ce qui engendre plusieurs émissions des messages ADV par le même noeud. Pour éviter cela, MCFA utilise un mécanisme de backoff. Le backoff permet de retarder la prise de décision sur la valeur locale du coût, en attendant la valeur optimale globale. L'intervalle du backoff dépend du coût du lien de réception : plus

le coût est grand, plus on a de chance de recevoir une valeur plus optimale, donc on doit attendre plus de temps :

```
Init
Self.cost <- inf
OnRecvADV(cost,from)
If (cost+linkCost(from)<self.cost)
Self.cost <- cost+linkCost(from)
initBackof(cost+linkCost(from))
onBackoffTimeout()
broadcastADV(self.cost)
```

ii Relais des paquets

Le relais dans MCFA n'utilise aucune identification des noeuds et aucune table de routage. Ce qui rend MCFA assez adapté aux environnements de capteurs. Lorsqu'un paquet est émis par une source vers le puits, il contient le coût minimal local du noeud. A la réception d'un paquet de donnée, le noeud vérifie si son coût local est égal au coût reçu moins le coût du lien de réception. Dans ce cas, le noeud relaie le paquet en remplaçant la valeur du coût par sa valeur locale :

```
OnRecvData(cost,from)
If (cost-linkCost(from)=self.cost)
broadcastData(self.cost)
```

e) Rumour Routing

Aperçus

Les protocoles précédents utilisent une forme d'inondation pour la propagation des intérêts ou des données. Le protocole Rumour Routing essaie de trouver un compromis entre l'inondation des intérêts et la propagation des données.

Fondamental

Les auteurs ont utilisé un procédé probabiliste, reposant sur le fait suivant : Des simulations basées sur la méthode de Monte-Carlo ont montré que la probabilité que deux lignes se croisent au sein d'une région rectangulaire est 0.69. De plus, si on utilise 5 lignes passant par un point, la probabilité qu'une autre ligne croise l'une des cinq lignes est 0.997 !

Par conséquent, si on considère le puits et la source comme deux points, et en établissant un nombre réduit de mi-chemins depuis la source et le puits, on aura une forte chance que deux mi-chemins se joignent, créant ainsi un chemin complet entre la source et la destination, tout en évitant l'inondation. La création de ces mi-chemins se base sur la notion d'agent. Un agent est un paquet avec une grande portée (TTL) qui traverse le réseau de noeud en noeud afin d'établir des tables de relais. Il existe deux types d'agents : d'évènement et de requête.

Agents d'évènements

Chaque noeud maintient une table de relais locale, qui contient, pour chaque intérêt, le prochain saut vers le puits et vers la source, ainsi qu'une métrique qui représente le nombre de saut vers chaque extrémité. Lorsqu'un noeud observe un nouvel évènement, il crée un nouvel agent suivant une certaine probabilité. L'agent contient la table d'évènements parcourus au sein du chemin ainsi que le nombre de sauts vers la source de chaque évènement. De plus, l'agent doit transporter avec lui la liste des noeuds parcourus ainsi que leurs voisin directs. La source choisit un voisin aléatoire et lui émet l'agent. Lorsqu'un noeud reçoit un agent, il effectue les opérations suivantes :

- Si l'agent contient un nouvel évènement, une nouvelle entrée dans la table locale est créée.
- Le noeud met à jour sa table locale et/ou la table de l'agent pour les entrées communes, suivant le nombre de sauts optimal : i.e. si l'agent possède une route plus courte vers un certain évènement, le noeud met à jour sa table, et vice-versa. Cette méthode permet d'optimiser des routes déjà établies par d'autres agents.
- Si le noeud connaît des évènements non connus par l'agent, il ajoute les entrées nécessaires dans la table de l'agent.
- Le noeud choisit comme prochain saut un de ses voisins n'appartenant pas à la liste des noeuds de l'agent, et modifie en conséquence sa table locale pour le prochain saut vers le puits (i.e. le noeud choisi représente le prochain vers le puits).
- Le noeud ajoute à la liste des noeuds parcourus son identificateur, ainsi que ceux de ses voisins.
- Le message est envoyé au noeud choisi.

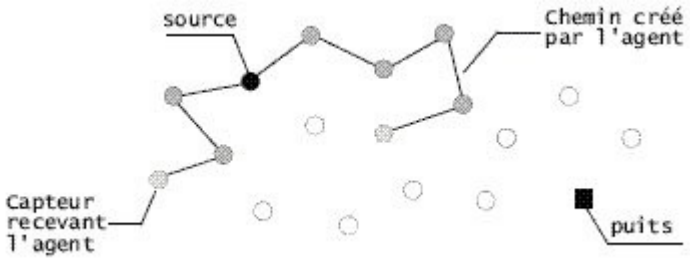


Image 24 : Agents d'événements

Agents de requêtes

Lorsque le puits désire prélever une donnée du réseau, il consulte sa table locale pour une route fraîche. Si aucune entrée n'est trouvée, il initialise un agent de requête. L'agent contient seulement la liste des noeuds visités. Lorsqu'un noeud reçoit un agent de requête, il vérifie l'existence d'un chemin dans sa table locale. Si ce n'est pas le cas, il choisit un voisin aléatoire et lui transmet l'agent, tout en ajoutant son identificateur dans la liste transportée par l'agent.

V

L'économie d'énergie et tolérance aux pannes dans les RCSF

La tolérance aux pannes dans les RCSF 47
Approches et solutions tolérantes aux pannes dans les RCSF 51

La limitation d'énergie dans les capteurs sans fil, et les environnements hostiles dans lesquels ils pourraient être déployés, sont des facteurs qui rendent ce type de réseaux très vulnérables. Ainsi, la perte de connexions sans fils peut être due à une extinction d'un capteur suite à un épuisement de sa batterie, ou tout simplement à une destruction physique accidentelle ou intentionnelle par un ennemi. Par ailleurs, l'absence de sécurité physique pour ce type de capteurs, et la nature vulnérable des communications radios sont des caractéristiques qui augmentent les risques de pannes sur ce type de réseau.

A. La tolérance aux pannes dans les RCSF

Certains nSuds capteurs peuvent être bloqués ou tomber en panne à cause d'un manque d'énergie, d'un dégât matériel ou d'une interférence environnementale. La panne d'un nSud capteur ne doit pas affecter le fonctionnement global de son réseau. C'est le problème de fiabilité ou de tolérance aux pannes. La tolérance aux pannes est donc la capacité de maintenir les fonctionnalités du réseau sans interruption due à une panne d'un nSud capteur.

1. Les pannes

Définition : Faute, erreur et faille

Une faille (ou panne) du système se produit lorsque son comportement devient inconsistant et ne fournit pas le résultat voulu. La panne est une conséquence d'une ou plusieurs erreurs. Une erreur représente un état invalide du système du à une faute (défaut). La faute est donc la première cause de l'erreur, cette dernière provoque la faille du système.

Remarque

Le but de la tolérance aux pannes est d'éviter la faille totale du système malgré la présence de fautes dans un sous ensemble de ses composants élémentaires. La tolérance de panne est d'autant meilleure que le nombre de composants en panne est grand (avec la garantie du bon fonctionnement du système).

Classification des pannes

Il est utile de classifier les pannes selon différents critères. Le schéma suivant montre une classification générale selon la durée, la cause ou le comportement d'une panne :

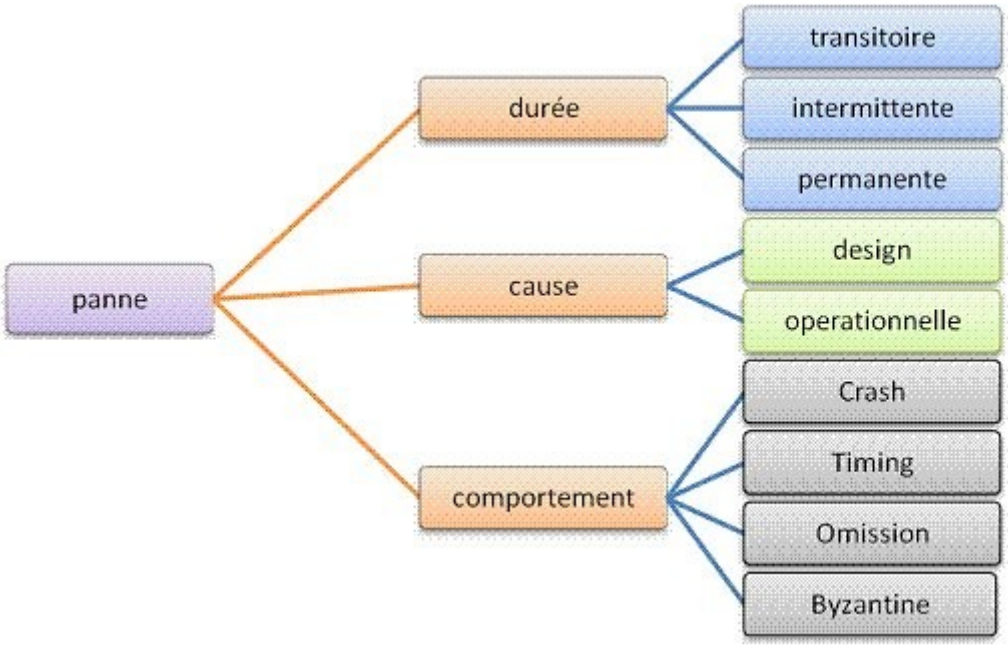


Image 25 : Classification des pannes

Pannes selon durée

Basée sur sa durée, la panne peut être classifiée en :

- **Transitoire** : conséquence d'un impact environnemental temporaire, elle peut éventuellement disparaître sans aucune intervention. La radiation cosmique est un exemple de panne transitoire ;
- **Intermittente** : variante de la panne transitoire, elle se produit occasionnellement et de façon imprévisible. Elle est généralement due à l'instabilité de certaines caractéristiques matérielles ou à l'exécution du programme dans un espace particulier de l'environnement ;
- **Permanente** : continue et stable dans le temps, la panne permanente persiste tant qu'il n'y a pas d'intervention externe pour l'éliminer. Un changement physique dans un composant provoque une panne matérielle permanente.

Pannes selon la cause

On distingue deux types de pannes selon leur cause :

- Panne de design : due à une mauvaise structuration du réseau ou du composant en particulier. En pratique, ce genre de panne ne devrait pas exister grâce aux tests et simulations avant la réalisation finale du réseau ;
- Panne opérationnelle : qui se produit durant le fonctionnement du système.

Elle est généralement due aux causes physiques. En outre, on peut distinguer, spécialement pour les réseaux de capteurs, trois principales causes:

- Energie : l'épuisement de la batterie cause l'arrêt du capteur. La consommation d'énergie est très importante pour déterminer la durée de vie d'un nSud capteur, et donc de tout le réseau ;
- Sécurité : la destruction physique accidentelle ou intentionnelle pas un ennemi peut être une cause de panne. L'absence de sécurité dans les réseaux de capteurs augmente le risque des pannes de ce type ;
- Transmission : la nature vulnérable de transmission radio, la présence d'obstacles dans les environnements hostiles ainsi que les interférences électriques peuvent être la source d'une faute lors du transfert de données.

Pannes selon le comportement résultant

Après l'occurrence d'une panne, on distingue quatre différents comportements possibles du composant concerné :

- Panne accidentelle (Crash) : le composant soit, s'arrête complètement de fonctionner ou bien continue mais sans retourner à un état stable (valide) ;
- Panne d'omission : le composant n'est plus capable d'améliorer son service (échec total) ;
- Panne de synchronisation (Timing) : le composant effectue son traitement mais fournit le résultat en retard ;
- Panne Byzantine : cette panne est de nature arbitraire ; le comportement du composant est donc imprévisible. Du à des attaques très malicieuses, ce type de pannes est considéré le plus difficile à gérer.

2. La tolérance aux pannes

La conception d'une procédure pour la tolérance aux pannes dépend de l'architecture et des fonctionnalités du système. Cependant, certaines étapes générales sont exécutées dans la plupart des systèmes; tel que c'est illustré dans la figure suivante :

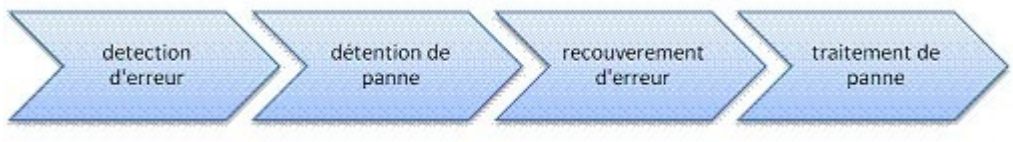


Image 26 : Procédures de tolérance aux pannes

Détection d'erreur

C'est la première phase dans chaque schéma de tolérance aux pannes, dans laquelle on reconnaît qu'un événement inattendu s'est produit. Les techniques de détection de pannes sont généralement classifiées en deux catégories : en ligne et autonome (offline). La détection offline est souvent réalisée à l'aide de programmes de diagnostic qui s'exécutent quand le système est inactif. La détection en ligne vise l'identification de pannes en temps réel et est effectuée simultanément avec l'activité du système.

Détention de la panne

Cette phase établit des limites des effets de la panne sur une zone particulière afin d'empêcher la contamination des autres régions. En cas de détection d'intrusion,

par exemple, l'isolation des composants compromis minimise le risque d'attaque des composants encore fonctionnels.

Recouvrement d'erreur

C'est la phase dans laquelle on effectue des opérations d'élimination des effets de pannes. Les deux techniques les plus utilisées sont « masquage de panne » et « répétition »

- Masquage de panne : utilise l'information redondante correcte pour éliminer l'impact de l'information erronée ;
- Répétition : après que la panne soit détectée, on effectue un nouvel essai pour exécuter une partie du programme, dans l'espoir que la panne soit transitoire.

Traitement de panne

Dans cette phase, la réparation du composant en panne isolé est effectuée. La procédure de réparation dépend du type de la panne. Les pannes permanentes exigent une substitution du composant avec un autre composant fonctionnel. Le système doit contenir un ensemble d'éléments redondants (ou en état standby) qui servent à remplacer les nSuds en panne.

Exemple : Tolérance aux pannes dans un RCSF

Le problème de fusion dans un réseau de capteurs multimodal tolérant aux pannes utilisant des capteurs numériques binaires peut être modélisé par l'exemple illustré dans la figure suivante. On considère un réseau de capteurs pour la reconnaissance de personnes déployé dans une société pour identifier ses employés. Six personnes nommées A, B, C, D, E et F travaillent dans cette société. Le système de reconnaissance utilise deux types différents de capteurs : 1) capteur de taille (grandeur) ; 2) capteur pour la reconnaissance de voix qui demande à chaque entrant d'introduire une phrase secrète donnée à l'aide d'un microphone. La figure ci-dessous montre les six personnes ainsi que leurs caractéristiques (taille et voix) représentées dans le graphe.

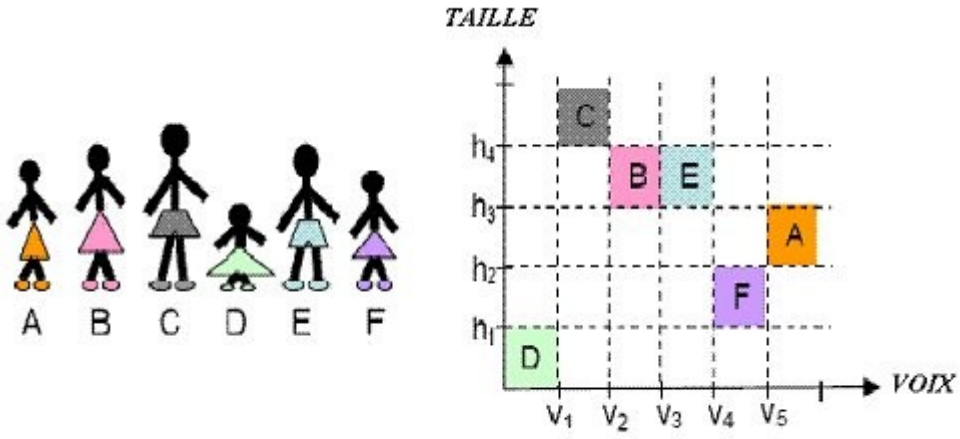


Image 27 : Exemple d'un RCSF tolérant aux pannes

Il est évident de constater que le système peut distinguer entre deux personnes P1 et P2 si elles sont représentées dans deux surfaces différentes sur le graphe. Selon notre exemple, si tous les capteurs fonctionnent correctement, chaque personne va occuper une surface différente. En outre, dans la plupart des cas, et malgré la défaillance de l'un des capteurs de taille ou de voix, la reconnaissance de toutes les

personnes est encore possible. Ceci grâce à la tolérance aux pannes hétérogène où le capteur en panne d'un certain type peut être remplacé par la fonctionnalité d'un capteur de l'autre type. Cependant, pour le cas des personnes B et E, qui ont la même taille, la voix est le seul critère pour les distinguer ; d'où, le système ne devrait avoir aucune tolérance aux pannes pour le capteur V3 qui distingue entre B et E. Si on exclut l'un de B ou E du personnel de la société, alors le système sera complètement tolérant aux pannes.

B. Approches et solutions tolérantes aux pannes dans les RCSF

La tolérance aux pannes a connu une importance considérable parmi les différents domaines de recherche dans les réseaux de capteurs sans fil; du à leurs contraintes d'énergie, d'environnement et de déploiement. Ce dernier étant d'un coût prohibitif, présente un handicape pour la réorganisation du réseau en cas de panne d'un ou plusieurs de ses capteurs. D'où, il était impératif d'introduire un mécanisme de tolérance aux pannes dans tous les protocoles implémentés au niveau des différentes couches de l'architecture RCSF afin de garantir le bon fonctionnement du réseau même après la faille de certains de ses composants.

1. Classification des solutions de tolérance aux pannes dans les RCSF

Les protocoles tolérants aux pannes peuvent être vus de plusieurs angles différents. De ce fait, un ensemble de critères est défini pour les classifier. Nous citons, entre autre, des catégories de trois classifications distinctes

Classification selon la phase de traitement

Dans cette classification, nous divisons l'ensemble des algorithmes en deux principales catégories. Si le traitement est effectué avant la panne ; on parle donc d'algorithmes préventifs sinon, les algorithmes sont dits curatifs.

- **Algorithme préventif** : implémente des techniques tolérantes aux pannes qui tentent de retarder ou éviter tout type d'erreur afin de garder le réseau fonctionnel le plus longtemps possible. La conservation d'énergie à titre d'exemple, permet de consommer moins d'énergie et évite donc une extinction prématurée de la batterie ce qui augmente la durée de vie des nSuds ;
- **Algorithme curatif** : utilise une approche optimiste, où le mécanisme de tolérance aux pannes implémenté n'est exécuté qu'après la détection de pannes. Pour cela, plusieurs algorithmes de recouvrement après pannes sont proposés dans la littérature, par exemple : le recouvrement du chemin de routage, l'élection d'un nouvel agrégateur...etc.

Classification architecturale

Cette classification traite les différents types de gestion des composants, soit au niveau du capteur individuellement ou bien sur tout le réseau. Nous distinguons trois catégories principales :

- **Gestion de la batterie** : cette catégorie est considérée comme une approche préventive, où les protocoles définissent une distribution uniforme

pour la dissipation d'énergie entre les différents nSuds capteurs ; afin de mieux gérer la consommation d'énergie et augmenter ainsi la durée de vie de tout le réseau. En outre, le mécanisme de mise en veille est une technique de gestion de batterie. En effet, les protocoles déterminent des délais de mise en veille des nSuds capteurs inactifs pour une meilleure conservation d'énergie;

- **Gestion de flux** : cette catégorie regroupe les techniques qui définissent des protocoles de gestion de transfert des données (routage, sélection de canal de transmission...etc.). nous pouvons trouver des approches préventives ou curatives sur les différentes couches (réseau, liaison de données...etc.) telles que :
  - **Routage multi-chemin** : utilise un algorithme préventif pour déterminer plusieurs chemins depuis chaque capteur vers le nSud collecteur. Ceci garantit la présence de plus d'un chemin fiable pour la transmission et offre une reprise rapide du transfert en cas de panne sur le premier chemin sélectionné (choisir un des chemins qui restent);
  - **Recouvrement de route** : après détection de panne, une technique curative permet de créer un nouveau chemin qui soit le plus fiable pour retransmettre les données ;
  - **Allocation de canal** : cette solution, implémentée au niveau MAC, effectue une allocation du canal de transmission d'une manière à diminuer les interférences entre les nSuds voisins et éviter les collisions durant le transfert ;
  - **Mobilité** : certains protocoles proposent comme solution tolérante aux pannes la sélection d'un ensemble de nSuds mobiles chargés de se déplacer entre les capteurs et collecter les données captées. Ceci réduira l'énergie consommée au niveau de chaque capteur en éliminant sa tâche de transmission. Un nSud mobile est généralement doté d'une batterie plus importante que celle d'un nSud capteur.
- **Gestion des données** : les protocoles classés dans cette catégorie offrent une meilleure gestion de données et de leur traitement. Deux principales sous-catégories sont déterminées :
  - **Agrégation** : considérée comme approche préventive, l'opération d'agrégation effectue un traitement supplémentaire sur les données brutes captées depuis l'environnement. Un nSud agrégateur combine les données provenant de plusieurs nSuds en une information significative ; ce qui réduit considérablement la quantité de données transmises, demande moins d'énergie et augmente ainsi la durée de vie du réseau ;
    Clustering : une des importantes approches pour traiter la structure d'un réseau de capteurs est le clustering. Il permet la formation d'un backbone virtuel qui améliore l'utilisation des ressources rares telles que la bande passante et l'énergie. Par ailleurs, le clustering aide à réaliser du multiplexage entre différents clusters. En outre, il améliore les performances des algorithmes de routage. Plusieurs protocoles utilisent cette approche préventive (parfois considérée comme approche curative) qui sera détaillée plus tard.

Classification selon le niveau d'implémentation

Cette classification permet de répartir les protocoles sur les différentes couches de l'architecture des réseaux de capteurs . Ainsi, les algorithmes de routage sont au niveau réseau, les techniques de sélection de canal sur la couche MAC...etc.

2. Approches tolérantes aux pannes dans la couche MAC

Avoir un réseau dense augmente la tolérance aux pannes et la robustesse du système grâce à la notion de redondance. Cependant, une mauvaise gestion peut aboutir à plusieurs collisions durant la transmission aussi bien qu'à la congestion du réseau. D'où, il est impératif de concevoir un protocole MAC au niveau de la couche liaison de données qui garantit la livraison des messages via des liens sans fil. Les solutions de la littérature proposées à ce niveau de couche utilisent un algorithme préventif contre les pannes ; i.e. éviter les collisions durant la livraison des données (en éjectant des délais d'attente, des mécanismes d'écoute de canal ...etc.) afin de garantir la fiabilité de transmission. La tolérance aux pannes est donc assurée par une phase de prévention de pannes avant transmission.

a) DIN (Dynamic Interference Nullification)

Saffre et al. présente un algorithme distribué pour la sélection de canal de transmission avec génération de liens fiables dépourvus d'interférence entre les différents nSuds voisins, afin d'assurer l'émergence du réseau. Ainsi, DIN modélise la tolérance aux pannes par une opération de prévention contre les collisions en évitant au mieux l'effet d'interférence présente entre les voisins directs tout en minimisant la consommation d'énergie au niveau de chaque nSud. Basé sur les deux protocoles FDMA/TDMA, l'algorithme combine la sélection aléatoire d'intervalles de fréquences/temps avec différents niveaux de consommation d'énergie pour la transmission ; ceci pour minimiser la probabilité de collision au niveau du récepteur final mais en se basant uniquement sur l'information disponible localement (au niveau des voisins du nSud émetteur).

Fondamental

L'algorithme DIN est modélisé par le problème de coloration de graphe où chaque nSud doit occuper une couleur (ou canal) différente de celle de chacun de ses voisins directs ainsi que les voisins de distance 2 (les voisins de ses voisins) afin d'éviter les interférences au niveau de leur premier voisin commun. La difficulté dans cet algorithme vient de la nécessité d'accomplir cet objectif pour une distribution irrégulière (aléatoire) et imprévisible des composants du réseau de capteurs.

Complément

DIN a été simulé dans un réseau militaire pour détection de mouvement des véhicules ; il a montré une efficacité dans la sélection de canal de transmission en minimisant les interférences ainsi que la consommation d'énergie.

b) RMAC (Reliable Medium Access Control)

De même que l'algorithme DIN, le protocole RMAC utilise la technique de prévention de collisions pour garantir une transmission fiable des données, et offrir ainsi une bonne tolérance aux pannes dans tout le réseau. Pour cela, RMAC traite les différents problèmes au niveau MAC tels que la congestion, la collision, la latence ; puis définit des mécanismes spécifiques basés sur l'approche CSMA/CA ; il repose donc sur deux principaux concepts (voir figure suivante) :

- mécanisme d'écoute de la porteuse qui assure que le canal est disponible pour transmission (aucun nSud n'est entrain d'utiliser ce canal) ;
- utilisation du schéma de retard (backoff) pour réduire le conflit et éviter les collisions. L'idée est de retenir le nSud de l'accès au canal pour une durée aléatoire dans l'espoir que le canal soit libéré après cette période.

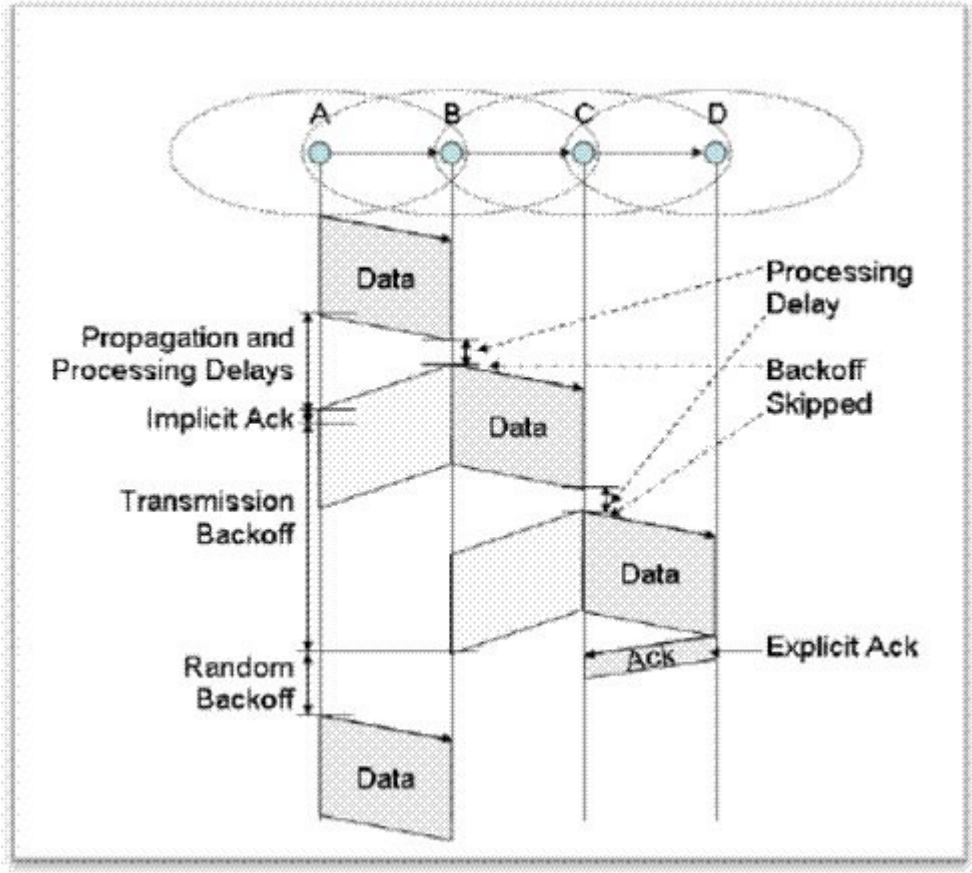


Image 28 : RMAC

3. Approches tolérantes aux pannes dans la couche réseau

Les réseaux de capteurs utilisent une communication multi-sauts. Comparée à une communication sans fil à longue distante, la communication multi-sauts présente une bonne solution pour le problème de propagation de signal et effets de dégradation de signal. De plus, il est recommandé pour ce type de communication de choisir le meilleur chemin qui soit de lien fiable, consomme le moins d'énergie et assure la livraison des données au collecteur. Les solutions proposées à ce niveau de couche sont classifiées en trois principales catégories : routage, agrégation et clustering ;

a) Solutions de routage tolérantes aux pannes

Les protocoles de routage permettent de choisir les meilleurs chemins pour acheminer la donnée depuis la source vers l'utilisateur final. Par ailleurs, ils permettent de sélectionner un chemin de remplacement en cas d'échec d'envoi sur la route initiale ; à cause d'une panne au niveau d'un ou plusieurs nSuds de cette route.

i PEQ : Periodic, Event-driven, Query-based

Aperçus

La motivation de cet algorithme vient du besoin de fournir un support pour toutes les contraintes : faible latence, fiabilité, recouvrement rapide en cas de panne et conservation d'énergie. PEQ combine la conservation d'énergie avec le routage multi-chemins en sélectionnant parmi toutes les routes disponibles, celles qui consomment moins d'énergie. En plus de ce mécanisme préventif qui permet un routage fiable, un mécanisme de recouvrement de pannes est implémenté. Ce dernier remplace le chemin en panne par une autre route qui soit de liens fiables et consomme moins d'énergie. Ainsi, le protocole PEQ couvre la procédure de tolérance aux pannes par la gestion de la consommation d'énergie, la sélection des meilleures routes puis leur recouvrement en cas de panne.

Fondamental : Publish/subscribe

PEQ introduit le paradigme Publish/Subscribe (voir figure suivante) pour l'interaction entre le collecteur et les nSuds capteurs. En effet, les capteurs envoient des notifications d'événements au collecteur, qui va souscrire son intérêt pour certaines de ces informations. Les capteurs concernés publient par la suite l'information désirée.

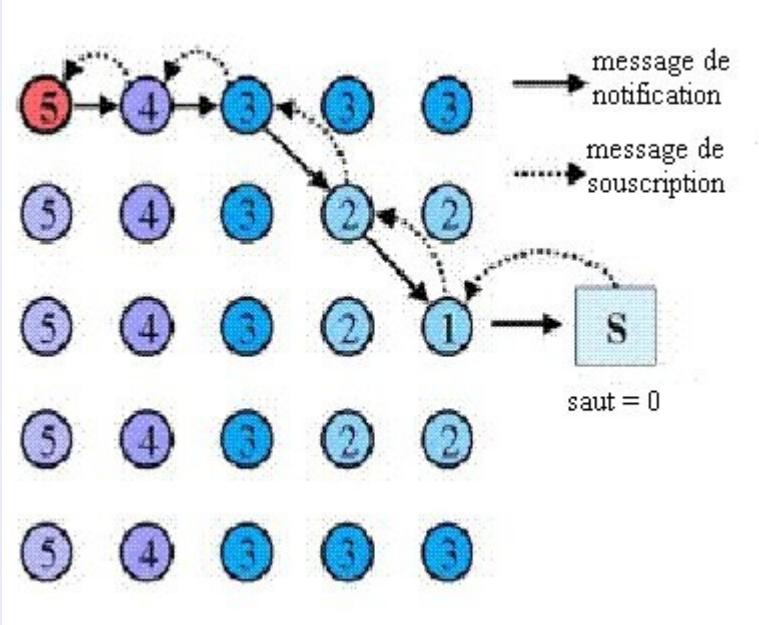


Image 29 : Mécanisme Publish / subscribe

Méthode

Les quatre principales phases du protocole sont:

1. **Construction de l'arbre de routage** : cet arbre permet de définir les différents chemins multi-sauts possibles pour acheminer les données. Le collecteur commence le processus en initialisant la variable « saut » à 0 ; par la suite, chaque nSud capteur prend la valeur du saut actuelle, l'incrémente puis l'envoie à tous ses voisins. Ainsi la valeur au niveau de chaque capteur désigne le nombre nécessaire de sauts pour communiquer avec le collecteur. A la fin de cette phase seulement les meilleurs chemins sont enregistrés;
2. **Transmission de paquets de notification** : chaque nSud capteur envoie selon sa table de routage et l'événement capté, une notification de l'information qu'il a. Pour cela, il utilise le chemin le plus rapide et le moins

coûteux en terme d'énergie ;

3. **Propagation des paquets de souscription** : dans cette étape, après une souscription, par le collecteur, des données à transmettre, chaque nSud achemine cette dernière jusqu'au nSud capteur concerné ;
4. **Mécanisme de recouvrement de route** : le recouvrement est effectué après détection de pannes (voir figure suivante). Un nSud envoie son paquet puis attend un acquittement ACK. S'il est reçu, le message a été bien transmis ; sinon une panne est détectée au niveau du chemin de routage. On effectue donc une recherche "SEARCH" pour la sélection d'un autre nSud destination tout en minimisant le coût du nouveau chemin. Si aucun nSud n'est trouvé (tous les voisins sont détruits) le nSud devient isolé et doit donc augmenter son rayon de transmission radio pour atteindre les nSuds voisins lointains.

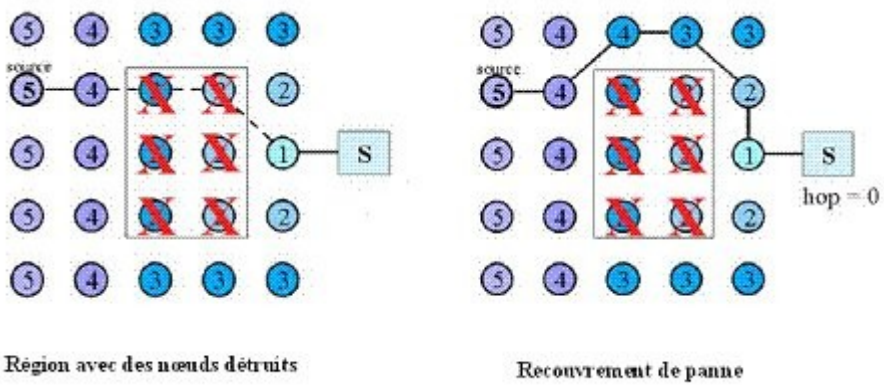


Image 30 : Recouvrement de route

ii Protocole EAR

Aperçus

Une solution hybride pour la tolérance aux pannes est proposée dans le protocole EAR. Pour son concept préventif, EAR offre une meilleure conservation d'énergie et définit plusieurs chemins de routage afin de garantir une fiabilité du transport et d'augmenter la durée de vie du réseau. En outre, un mécanisme de recouvrement de pannes est implémenté. Le protocole EAR supporte des réseaux de capteurs à collecteurs multiples (plusieurs nSuds puits). Chaque nSud capteur génère un paquet RPT (Report) contenant des informations pour les intérêts et préférences de l'utilisateur. Les paquets RPT peuvent être envoyés vers n'importe quel collecteur. Cependant, pour chaque nSud intermédiaire le protocole de routage choisit le meilleur chemin qui réduit la consommation d'énergie et la latence.

Méthode : Phase d'initialisation

Cette phase permet la construction de l'arbre de routage contenant tous les chemins possibles pour la dissémination des données. Chaque collecteur diffuse un message ADV (Advertisement) demandant des paquets RPT. Seuls les nSuds voisins du collecteur reçoivent le message ADV puis enregistrent le chemin dans leur table de routage ; sans propager le message ADV vers les autres nSuds, comme le montrent les étapes a) et b) de la figure suivante. Les autres nSuds capteurs envoient une demande RREQ (Route Request) cherchant un chemin vers le collecteur (étape c). Si un nSud, ayant déjà une route stockée dans sa table, reçoit RREQ, il envoie un paquet RREP (Route Reply) à son nSud voisin concerné par la demande (étapes d ; e). Le processus d'initialisation se termine quand chaque nSud reçoit une réponse RREP suite à sa requête RREQ ; puis enregistre le

chemin dans sa table de routage;

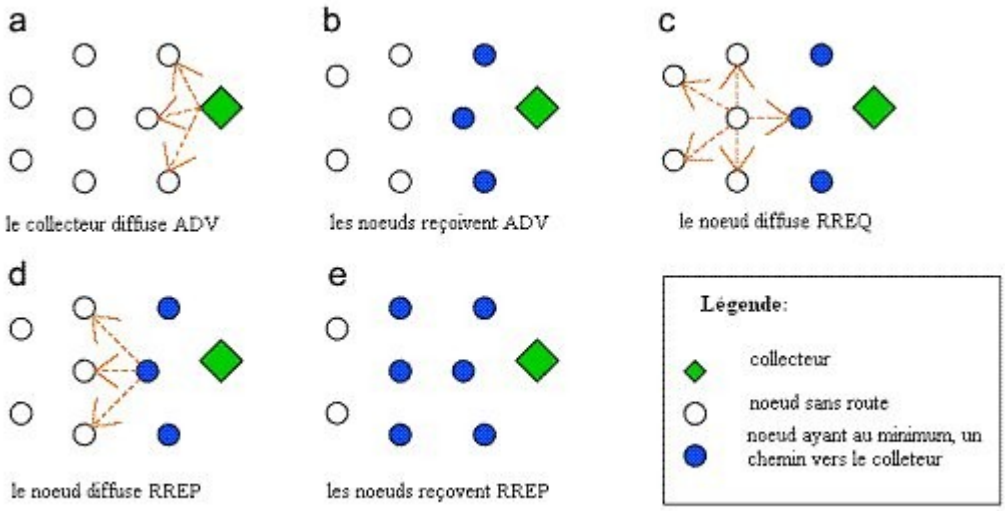


Image 31 : EAR

Méthode : Gestion de route

Les micro-capteurs, avec leur mémoire de taille réduite, ne peuvent garder tous les chemins possibles dans leurs tables de routage. Pour cela, et afin d'assurer une bonne tolérance aux pannes, on doit garder que les meilleurs chemins. Le protocole EAR définit donc deux métriques pour la sélection des meilleures routes à mémoriser. La première métrique est le nombre de sauts dans une route. Ceci permet de choisir le chemin le plus court. Cependant, la qualité des liens RF n'est pas prise en considération ; dans ce cas, le plus court chemin n'assure pas forcément la fiabilité de transmission. D'où on utilise le concept de « routes bannies ». En effet, si un chemin échoue à transmettre N paquets consécutifs, il sera mis dans une « liste noire » l'écartant ainsi d'une future utilisation. La deuxième métrique, appelée Score de route, est définie comme suit : RS = PE x WE + PT x WT Où:

PE: niveau de l'énergie du nSud du prochain saut ;

WE: poids assigné à PE ¬ [0-1] ;

PT: taux de succès dans la transmission ;

WT: poids assigné à PT ¬ [0-1] tel que : (WT + WE = 1);

L'historique enregistré sur l'état du routage dans tout le réseau permet, par conséquence, d'adapter la sélection des chemins en choisissant toujours les meilleurs chemins en termes de fiabilité de liens et de conservation d'énergie. Ceci garantit une bonne tolérance aux pannes en évitant la sélection des mauvaises routes.

Méthode : Dissémination de données

Après les deux premières étapes, chaque nSud aura au moins un chemin vers le collecteur. Les capteurs commencent donc à générer des paquets RPT, et le routage des données utilise la métrique « score de route » pour définir le meilleur chemin à emprunter. En cas où ce dernier présente une panne au niveau d'un ou plusieurs de ses nSuds, un mécanisme de recouvrement de route est exécuté, afin d'élire un second chemin fiable pour transmettre les données depuis le capteur vers le collecteur. Par ailleurs, au moment de sa durée d'inactivité, chaque capteur est mis en veille afin d'épargner davantage son énergie et augmenter ainsi la durée de vie

de tout le réseau.

iii VTRP (Variable Transmission Range Protocol)

Aperçus

VTRP est une solution de variation du rayon de transmission pour une meilleure propagation de données. Il permet de remédier au problème d'obstacles en les évitant par l'augmentation du rayon de transmission. Ce dernier augmente la probabilité d'atteindre des nSuds actifs quand le rayon actuel utilisé ne couvre aucun nSud à cause de pannes ou d'inactivité des nSuds voisins ou encore dans le cas des réseaux à faible densité. En outre, VTRP offre une meilleure longévité du réseau en évitant l'utilisation fréquente des nSuds critiques (les voisins proches du collecteur) ceci permet d'alléger leur fonction de routage ; conserve leur batterie et augmente ainsi la durée de vie de tout le réseau.

Fondamental

VTRP utilise des rayons de transmissions variés pour la propagation de données ; i.e. il permet d'augmenter les rayons de transmissions de différentes manières. Soit k nSuds avec k informations. Le problème posé dans cette étude est donc « comment acheminer toutes les k informations au collecteur d'une manière fiable et efficace ».

Méthode : Phase de recherche

Soit p' un nSud qui a reçu une information E de p. dans la phase de recherche, p' utilise une diffusion périodique de message afin de découvrir le noeud p le plus proche du collecteur (meilleur chemin de p' vers le collecteur en passant par p). Cependant, Une détection de panne est possible si aucun noeud p n'est trouvé. Trois différentes raisons sont possibles pour un tel échec : soit le noeud p est mis en veille et ne peut donc répondre à p'; soit il est en panne (détruit, batterie épuisée...etc.) ou bien à cause d'un obstacle qui empêche la communication entre les deux nSuds ;

Méthode : Phase de transmission directe

En cas où la phase de recherche réussit, p' envoie l'information à p et envoie un message « succès » à p ;

Méthode : Phase de variation du rayon de transmission

Si la phase de recherche échoue (aucun noeud p n'est détecté) p' passe à la phase de variation de son rayon de transmission qui représente le cas de recouvrement après pannes. En effet, chaque nSud maintient un compteur local ³ initialisé à 0. A chaque échec de l'étape de recherche, le compteur est incrémenté, et le rayon de transmission R est modifié selon la valeur de ³. Quatre différentes fonctions sont définies selon la vitesse de variation du rayon de transmission : linéaire, multiplicative, exponentielle et aléatoire ;

1. Progrès constant : VTRP est convenable dans ce cas aux réseaux où un large nombre de nSuds est compromis ;
2. Progrès multiplicatif : VTRPm définit un rayon de transmission qui est augmenté d'une manière radicale. Ce changement offre une meilleure probabilité pour trouver des nSuds actifs. En revanche, il requiert une consommation d'énergie plus importante ;
3. Progrès exponentiel : VTRPp est une variante qui augmente le rayon d'une vitesse encore plus rapide ;
4. Progrès aléatoire : quand la densité du réseau n'est pas connue au préalable, on utilise l'approche aléatoire VTRPr pour éviter un mauvais

comportement du réseau suite à un mauvais choix.

b) Clustering et tolérance aux pannes

Les algorithmes proposés dans cette catégorie viennent au secours du problème d'auto-organisation du réseau. Les protocoles du clustering divisent le réseau en un ensemble de clusters ayant chacun un nSud hôte (cluster head) qui récupère les données depuis tous les capteurs de son cluster puis les achemine vers le collecteur. Cette solution permet de mieux gérer le trafic de réseau et d'alléger la quantité d'information qui circule, en effectuant des traitements au sein du cluster avant de propager les données vers le reste du réseau pour les transmettre au collecteur.

i Protocole CPEQ

Aperçus

En plus de tous les mécanismes de tolérance aux pannes qu'implémente PEQ, la variante CPEQ (Cluster-based PEQ) ajoute un module de clustering pour offrir une meilleure gestion de routage. En effet, les nSuds ayant le plus d'énergie résiduelle sont sélectionnés comme nSuds agrégateurs (appelés aussi cluster head ou hub). Un nSud agrégateur établit son cluster, et les nSuds appartenant à ce dernier envoient leurs données à l'agrégateur qui effectue d'éventuel traitement sur les données brutes puis les achemine vers le collecteur. Chaque nSud du réseau peut devenir agrégateur pendant une certaine période de temps selon son niveau de batterie. Le but principal de CPEQ est de distribuer d'une manière uniforme la dissipation d'énergie entre les nSuds, et de réduire la latence et le trafic de données dans le réseau. Le protocole CPEQ est réalisé en cinq étapes

Méthode : Configuration initiale

Cette phase est basée sur l'algorithme PEQ ; où chaque nSud commence par un mécanisme d'inondation (diffusion) pour configurer tout le réseau et connaître par la suite le nombre de sauts nécessaires pour atteindre le collecteur. En outre, CPEQ introduit un champ additionnel contenant le pourcentage des nSuds qui deviendront agrégateurs ;

Méthode : Sélection d'agrégateur

C'est la phase d'élection des clusters-heads (appelés ici agrégateurs). Après la configuration initiale, chaque nSud peut devenir agrégateur avec un pourcentage donné. En effet, chaque nSud génère un nombre aléatoire entre 0 et 1. Si ce nombre est inférieur à une probabilité p (probabilité pour devenir agrégateur), le nSud demande à tous ses voisins immédiats leur niveau de batterie en envoyant un paquet REQ_EN (Request Energy). Chaque voisin répond par un message REP_EN (Reply Energy) contenant son ID et la quantité d'énergie. Le nSud choisit le voisin ayant le maximum d'énergie et diffuse un SET_AGR (Set Aggregator) pour informer tous les nSuds du nouvel agrégateur. Les trois étapes de cette phase sont illustrées dans la figure suivante ;

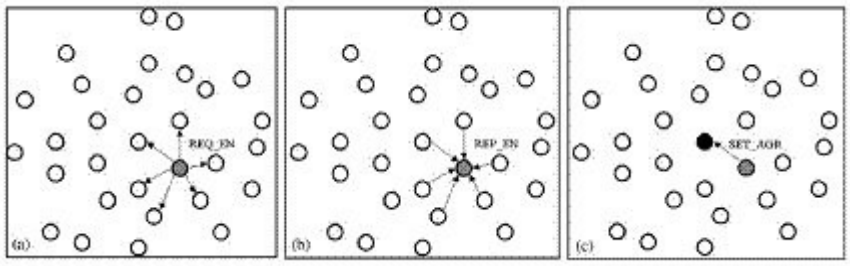


Image 32 : Configuration initiale

Méthode : Configuration de clusters

Cette phase divise le réseau en un ensemble de clusters. Le nouveau nSud agrégateur sélectionné doit avertir ses voisins de son rôle d'agrégation. De cette manière chaque agrégateur construit son cluster de nSuds. La configuration de clusters est réalisée à l'aide de messages AGR_NTF (Aggregator Notification) avec un champ TTL pour limiter la propagation du paquet sur les nSuds se trouvant à une distance inférieure ou égale au TTL. Chaque fois qu'un nSud reçoit ce message, il enregistre l'ID du nSud émetteur dans sa table de routage pour déterminer le chemin vers l'agrégateur. Si un nSud reçoit plusieurs messages AGR_NTF ; il choisit l'agrégateur avec le moindre nombre de sauts. La figure suivante illustre la configuration de clusters avec un TTL=2 ;

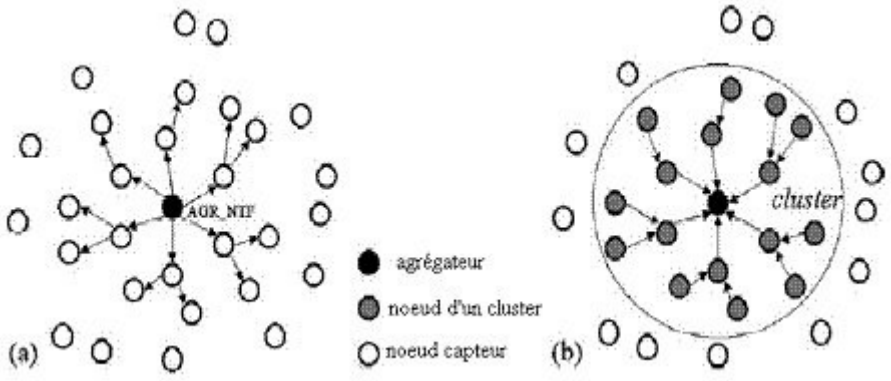


Image 33 : Configuration de clusters

Méthode : Transmission de données à l'agrégateur

L'algorithme de routage des données est le même implémenté dans le protocole PEQ. Chaque nSud utilise sa table de routage pour envoyer la donnée vers son agrégateur. Dans CPEQ l'agrégateur peut être considéré comme un nSud puits. Le mécanisme de recouvrement de chemin est aussi hérité du protocole PEQ;

Méthode : Transmission de données au collecteur

Après réception des données depuis les nSuds capteurs de son cluster, l'agrégateur doit acheminer ces données au collecteur. CPEQ utilise une communication multi sauts entre l'agrégateur et le collecteur tel que montre la figure suivant.

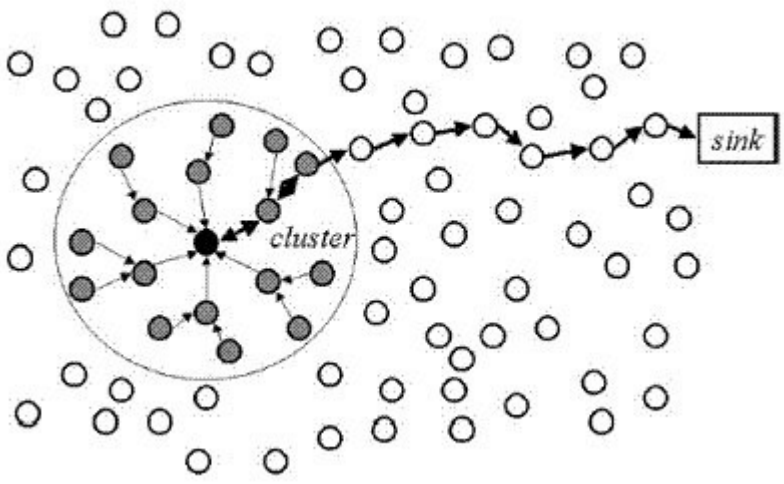


Image 34 : Transmission des données au collecteur

ii Algorithme K-CDS

Définition : k-CDS

Soit G = (V, E) le graphe qui représente notre réseau de capteurs sans fil ; où V est l'ensemble des nSuds capteurs, E est l'ensemble des liens sans fil.

- **Définition 1** : un réseau G est k-connexe s'il n'aura aucune partition quand l'on omet i nSuds du graphe (i = 1, 2, ..., k-1). Une définition équivalente est donnée par le théorème de Menger ; telle que G est k-connexe si chaque deux nSuds sont connectés par k chemins disjoints.
- **Définition 2** : un sous ensemble V' □V est un k-DS (k-dominating Set) de G si chaque nSud de V qui n'appartient pas à V' a au moins k voisins dans V'. Le k-DS V' devient k-CDS si le sous graphe G[V'] est k-connexe.

Aperçus du protocole k-CDS

L'algorithme K-CDS utilise une approche préventive basée sur le clustering. Il propose une construction d'un ensemble k-connexe dominant k-CDS (k-Conncted k-Dominating Set) comme un backbone virtuel pour offrir une efficacité de routage aussi bien qu'une bonne tolérance aux pannes. Pour cela, quatre approches ont été introduites ; dont deux sont des algorithmes probabilistes, une est déterministe et la dernière est une hybridation des approches déterministes et probabilistes. Ces quatre approches permettent de considérer différents critères pour la construction des clusters.

Méthode : Protocole k-Gossip

C'est un protocole probabiliste où ; chaque nSud a une probabilité Pk pour devenir un nSud du backbone. Pk est déterminé selon la valeur de k, la surface de déploiement du réseau A, le nombre total des nSuds du réseau N, et le rayon de transmission R. La sélection des nSuds backbone est purement aléatoire ; et ne demande aucune information sur les nSuds voisins. Cependant, elle requiert des informations globales sur le réseau (les paramètres k, A, N et R) pour déterminer la valeur de probabilité Pk. le nombre des nSuds backbone prévus est donc N* Pk.

Méthode : Protocole k-Grid

C'est une approche probabiliste adaptée pour les distributions des nSuds non uniformes, où chaque nSud a au moins Bk voisins backbone. Le paramètre Bk est calculé selon les informations globales du réseau aussi bien que les informations

des voisins de chaque nSud. L'algorithme de k-Grid est donné comme suit :

k-Grid :

1) Chaque nœud calcule son degré de voisinage $\delta(v) = |N(v)|$ en échangeant des messages « *Hello* » avec tous les nœuds voisins ;
2) Chaque nœud devient backbone avec une probabilité $min\left\{1, \frac{B_k}{\delta(v)+1}\right\}$.

Image 35 : k-Grid

Méthode : Approche déterministe

Cette approche définit une condition de couverture telle que chaque nSud v est omis du backbone si pour chaque couple (u, w) de ses voisins il existe k chemins disjoints qui connectent u et w via plusieurs nSuds intermédiaires et avec une priorité supérieure à celle de v. La figure suivante montre un exemple où les nSuds u et w voisins du nSud v sont connectés par des chemins disjoints P1, P2, ..., Pk constitués de nSuds de forte priorité (représentés en gris).

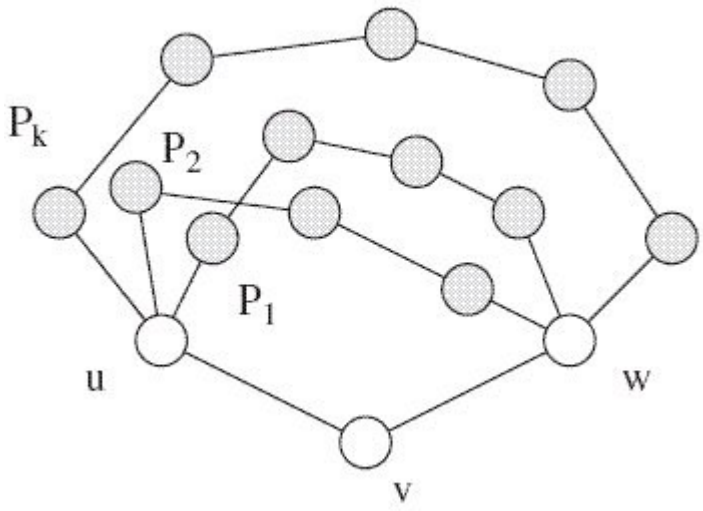


Image 36 : Approche déterministe

Si cette condition de k-couverture est appliquée sur un réseau G k-connexe, le backbone virtuel résultant forme un k-CDS pour G. cette approche permet donc, d'une manière déterministe, de construire un backbone virtuel formant un k-CDS pour le réseau G en supposant initialement que tous les nSuds appartiennent au backbone puis en appliquant la condition de k-couverture sur cet ensemble de nSuds.

Méthode : Algorithme CBKC

Color-Based k-CDS Construction est une approche hybride basée sur la technique de coloration de graphe et permet de construire un k-CDS avec une forte probabilité dans des réseaux relativement dense. Contrairement aux schémas

purement probabilistes, l'algorithme CBKC ne dépend pas des paramètres du réseau. Par ailleurs, il est plus facile à implémenter que l'algorithme déterministe discuté ci-dessus. L'idée de base de cette approche hybride est de partitionner aléatoirement le réseau en sous réseaux de différentes couleurs, puis appliquer l'algorithme CDS traditionnel pour chaque sous réseau. La première étape est probabiliste ; quand le réseau est suffisamment dense, les nSuds colorés de chaque partition forme un CDS pour le réseau original. La deuxième étape est déterministe ; chaque backbone coloré construit à l'intérieur d'un sous réseau par l'algorithme CDS est aussi bien un CDS pour le réseau entier. Tous ensemble, les backbones k-colorés forment un k-CDS pour le réseau. La construction CBKC est détaillée ci-dessous :

***Color-Based k-CDS Construction* CBKC**

1) Chaque nœud v sélectionne une couleur aléatoire c_v ($1 = c_v = k$). par conséquence, l'ensemble des nœuds V est divisé en k sous ensembles V_1, V_2, …,V_k où chaque sous ensemble V_c contient les nœuds de la couleur c ;
2) Pour chaque couleur c, un algorithme CDS local est appliqué pour construire un backbone virtuel V_c' ? V qui couvre le réseau original ;
3) Le k-CDS final est l'union $\bigcup_{c=1}^{k} V_c'$ de tous les backbones virtuels colorés.

Image 37 : CBKC

iii KAT Mobility

Aperçus

Dans KAT mobiliy (K-means And TSP-based mobility), en plus du clustering, le concept de mobilité est implémenté au niveau des nSuds collecteurs. Ces deux mécanismes, définissent une technique préventive hybride tolérante aux pannes qui offre une meilleure gestion d'énergie et augmente donc la durée de vie du réseau. Après réorganisation du réseau en clusters, la méthode proposée pilote le collecteur mobile pour se déplacer à travers les centres des clusters en prenant le chemin optimal. Le collecteur mobile récupère donc les données depuis les capteurs des clusters visités. Le principe de KAT mobility se résume en deux procédures: clustering, optimisation du chemin de routage. La figure suivante illustre le principe de KAT-Mobility.

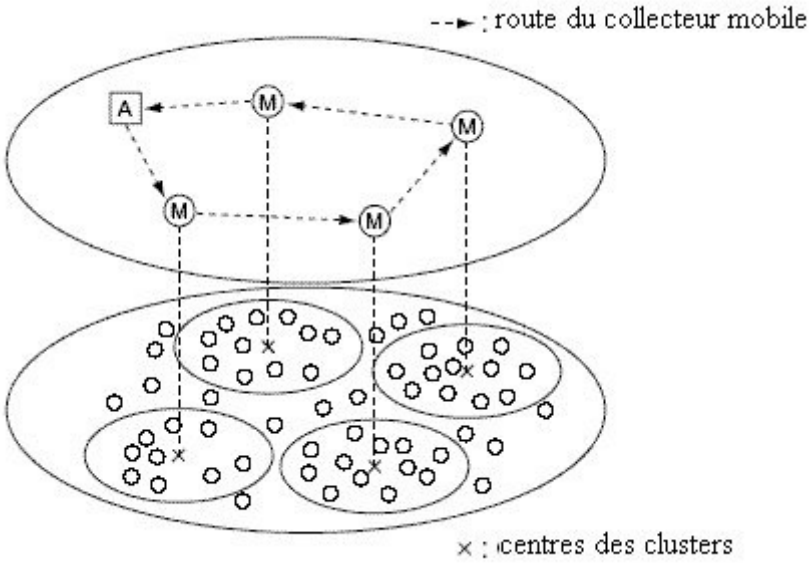


Image 38 : KAT-Mobility

Méthode : Algorithme de clustering

Cette procédure divise l'ensemble des N capteurs en k clusters C1, C2, ..., Ck (N k k). le coût du cluster est évalué par l'erreur approximative entre le collecteur et les nSuds ; soit d(x, yi) cette erreur, où x est un nSud capteur ; yi est un collecteur (i = 1, 2, ..., k). d(x, yi) est définie par la distance euclidienne entre le capteur et le collecteur. Le but est donc, d'affecter chaque capteur à un cluster Ci en minimisant l'erreur totale des clusters.

Méthode : Optimisation du chemin de routage

Trouver un chemin optimal pour le nSud mobile est identique au problème du voyageur de commerce TSP . Ainsi ; un collecteur représente le voyageur, et les centres des clusters définissent les villes. L'optimisation de la route du collecteur mobile pour visiter tous les nSuds centres des clusters une et une seule fois est équivalente à la recherche du plus court voyage d'un commerçant pour visiter chaque ville une seule fois.

Les résultats de simulation ont montré que KAT mobility peut fournir une meilleure conservation d'énergie aussi bien qu'une bonne tolérance aux pannes en cas de mal fonctionnement de certains nSuds.

c) Agrégation et tolérance aux pannes

Minimiser la consommation d'énergie revient à minimiser, entre autre, la quantité de données transmises dans le réseau. En effet, d'après les statistiques, 70% de l'énergie consommée dans un noeud capteur est due aux transmissions. L'agrégation combine les données provenant de plusieurs nSuds en une information significative ; en éliminant ainsi la redondance. Ceci résout le problème d'implosion dans le routage et allège ainsi la congestion du réseau.

i Classification des protocoles d'agrégation

On peut classer les différentes techniques d'agrégation de données dans les réseaux de capteurs en deux approches, comme illustré à la figue suivante :

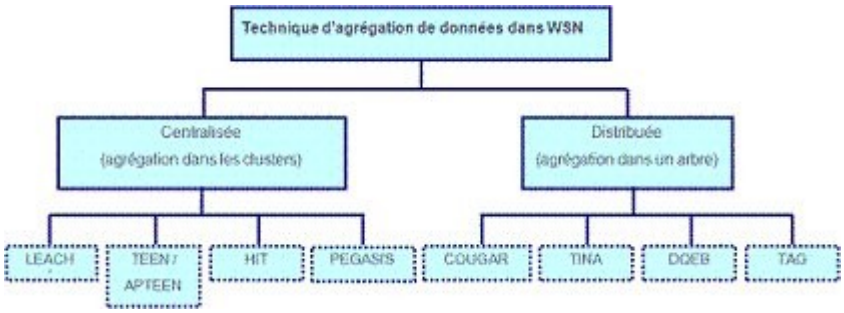


Image 39 : Protocoles d'agrégation dans les réseaux de capteurs sans fils

ii LEACH: Low-Energy Adaptive Clustering Hierarchy

Aperçus

LEACH (Low-Energy Adaptive Clustering Hierarchy) est un protocole de routage hiérarchique, employant un procédé de clustering qui divise le réseau en deux niveaux : les cluster-heads et les noeuds membres. Le protocole se déroule en rounds. Chaque round se compose de deux phases : construction et communication.

Méthode : Phase de construction

Le but de cette phase est la construction des clusters en choisissant les chefs et en établissant la politique d'accès au média au sein de chaque groupe. Cette phase commence par la prise de décision locale pour devenir cluster-head. Chaque noeud n choisit un nombre aléatoire, si ce nombre est inférieur à une valeur T (n), le noeud devient cluster-head. T (n) est définie comme suit :

$$T(n) = \begin{cases} \frac{P}{1-P\times\left(r \bmod \frac{1}{P}\right)} & \text{si } n \in G \\ 0 & \text{sinon} \end{cases}$$

Image 40 : T(n)

avec :

- P : pourcentage désiré de cluster-heads pendant un round.
- r : numéro du round.
- G : l'ensemble des noeuds qui n'ont pas été élu cluster-heads pendant les 1/P rounds précédents. Par la suite, chaque noeud qui s'est élu cluster-head émet un message de notification. Les noeuds membres récoltent les messages de notification, et décident leur appartenance à un cluster. La décision est basée sur l'amplitude du signal reçu : le cluster-head ayant le signal le plus fort est choisi (i.e. le plus proche). En cas d'égalité, un chef aléatoire est choisi. Chaque membre informe son chef de sa décision. Toutes les communications précédentes étant faite dans une topologie plate, la méthode CSMA doit être employée. Par la suite, les communications au sein d'un cluster peuvent être faites avec la méthode TDMA. Pour cela, chaque chef établie un schedule TDMA pour ses membres, en indiquant pour chaque noeud son slot d'émission. Ce schedule est envoyé aux membres.

Méthode : Phase de communication

En utilisant le schedule TDMA, les membres émettent leurs données captées pendant leurs propres slots. Cela leur permet d'éteindre leur interface de communication en dehors de leurs slots réservés, afin d'économiser leur énergie. Ces informations sont ensuite agrégées, pour être transmises au collecteur (sink).

Cette communication, entre un cluster-head et le collecteur, se fait d'une manière directe, i.e. : le cluster-head adapte son émetteur radio afin d'atteindre directement le collecteur.

iii TEEN (Threshold sensitive Energy Efficient sensor Network protocol)

Aperçus

En utilisant TDMA, le protocole LEACH est destiné aux applications time-driven. Dans ce type d'application, la donnée est propagée d'une manière périodique. Cependant, ce genre de protocole est inadapté pour les applications event-driven, où un comportement réactif est nécessaire pour le bon fonctionnement du système. TEEN (Threshold sensitive Energy Efficient sensor Network protocol) a été développé pour modeler LEACH afin de répondre aux exigences des applications event-driven.

Méthode

La majorité du comportement de TEEN est semblable au protocole LEACH. Cependant, quelques différences existent. Les chefs élus ne transmettent pas un schedule TDMA, mais émettent un message contenant les informations suivantes :

- Attributs : représentent la tâche demandée au capteur.
- Hard threshold (HT) : détermine la valeur critique après laquelle les membres doivent envoyer leur rapports de données.
- Soft threshold (ST) : spécifie le changement minimal obligeant le noeud à envoyer un nouveau rapport.

Donc, lorsqu'un noeud s'aperçoit que la valeur captée a dépassé HT, il doit émettre un rapport au chef. Il ne réémet un nouveau rapport que si la valeur change radicalement, i.e. : la différence dépasse ST. Ce mécanisme permet d'implémenter un comportement réactif, tout en limitant le nombre de messages utilisés.

Remarque

Etant donné la nature réactive de TEEN, l'emploi de TDMA est inadapté. Plusieurs alternatives peuvent être considérées : CDMA ou bien CSMA.

iv COUGAR

Dans Cougar, les données produites par le réseau de capteurs sont modélisées comme une table relationnelle. Dans cette table, chacun des attributs représente soit des informations sur le nSud capteur ou bien des données produites par ce nSud. L'approche Cougar fournit une agrégation partielle au niveau des nSuds. Chaque nSud maintient une liste d'attente contenant les nSuds fils qui doivent lui envoyer les paquets. Le nSud n'émet le paquet agrégé au prochain saut que s'il a reçu les paquets de tous les nSuds de la liste d'attente. Cependant, un nSud peut devenir inaccessible à cause du mouvement ou d'un problème de batterie. Pour cela, Cougar utilise un Timer afin d'éviter une attente indéfinie.

La sécurité des échanges dans les RCSF

VI

La sécurité dans les RCSF : taxonomie des menaces et des solutions	69
La gestion de clés dans les RCSF	71
Sécurité du routage dans les RCSF	80
Sécurité de l'agrégation dans les RCSF	88

Les propriétés des réseaux de capteurs sont à double tranchant. Certes ils permettent une grande facilité de production et de déploiement, mais rendent le système global de communication assez « fragile » à un certain nombre de défaillances.

Afin d'assurer un déploiement à large échelle de cette technologie, il est nécessaire de pallier ces problèmes de sécurité aux différent niveaux d'une architecture RCSF.

A. La sécurité dans les RCSF : taxonomie des menaces et des solutions

Vue globale des problèmes de sécurité et solutions dans les RCSF

Les principaux problèmes de sécurité dans les RCSF émergent à partir des propriétés qui les rendent efficaces et attrayants, qui sont:

- Limitation de ressources: l'énergie est peut-être la contrainte la plus forte aux capacités d'un noeud capteur. La réserve d'énergie de chaque nSud doit être conservé pour prolonger sa durée de vie et ainsi que celle de l'ensemble du réseau. Dans la plupart du temps, l'information transmise est redondante vus que les capteurs sont généralement géographiquement co-localisés. La plupart de cette énergie peut donc être économisée par agrégation de données. Cela exige une attention particulière à détecter l'injection de fausses données ou la modification défectueuse de données, lors des opérations d'agrégation au niveau des noeuds intermédiaires.
- La communication sans fils multi-sauts: en plus de fournir un déploiement simple, la communication sans fil a l'avantage d'offrir l'accès à des endroits difficilement accessibles tels que des terrains désastreux et hostiles. Malheureusement, la portée de la communication radio des "motes" est limité en raison de considérations énergétiques. La communication multi-sauts est donc indispensable pour la diffusion des données dans un RCSF. Cela introduit de nombreuses failles de sécurité à deux niveaux différents: attaque de la construction et maintenance des routes, et attque des données

utiles par injection, la modification ou la suppression de paquets. En outre, la communication sans fil introduit d'autres vulnérabilités à la couche liaison en ouvrant la porte à des attaques de brouillage et de style déni de service par épuisement des batteries.

- Couplage étroit avec l'environnement: la plupart des applications de RCSF exigent un déploiement étroit des noeuds à l'intérieur ou à proximité des phénomènes à surveiller. Cette proximité physique avec l'environnement conduit à de fréquentes compromission intentionnelle ou accidentelle des noeuds. Comme le succès des applications RCSF dépend également de leur faible coût, les nSuds ne peuvent pas se permettre une protection physique inviolable. Par conséquent, un adversaire "bien équipé" peut extraire des informations cryptographiques des noeuds capteurs. Comme la mission d'un RCSF est généralement sans surveillance, le potentiel d'attaquer les noeuds et de récupérer leur contenu est important. Ainsi, les clefs cryptographiques et informations sensibles devraient être gérés d'une manière qui augmente la résistance à la capture des noeuds.

La figure suivante résume les problèmes de sécurité émergeant des caractéristiques d'un RCSF et les solutions à entreprendre :

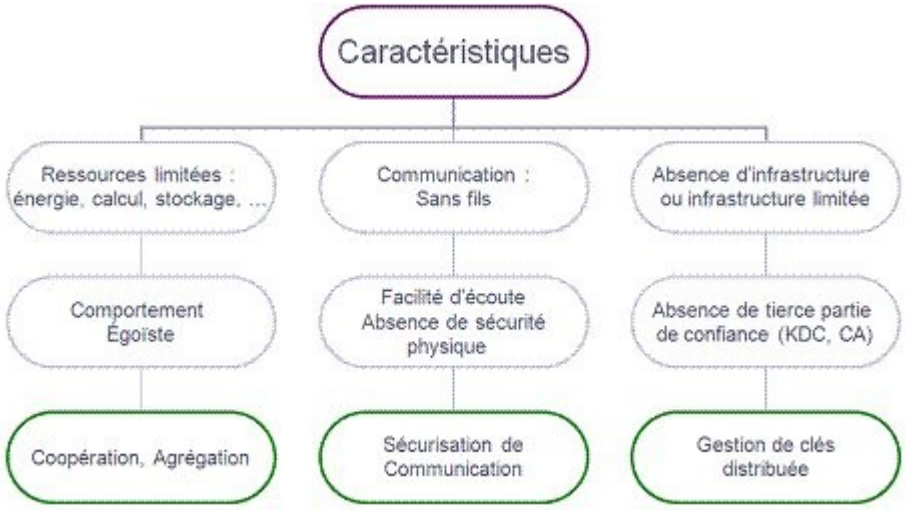


Image 41 : Sécurité dans les RCSF : propriétés, challenges et solutions

Bloques fonctionnels de la sécurité dans les RCSF

Comme illustré à la figure suivante, on distingue quatre blocks fonctionnels des solutions de sécurité dans les RCSF : la gestion de clés, la sécurité du routage, la sécurité de l'agrégation de données, et la sécurité de l'accès au canal.

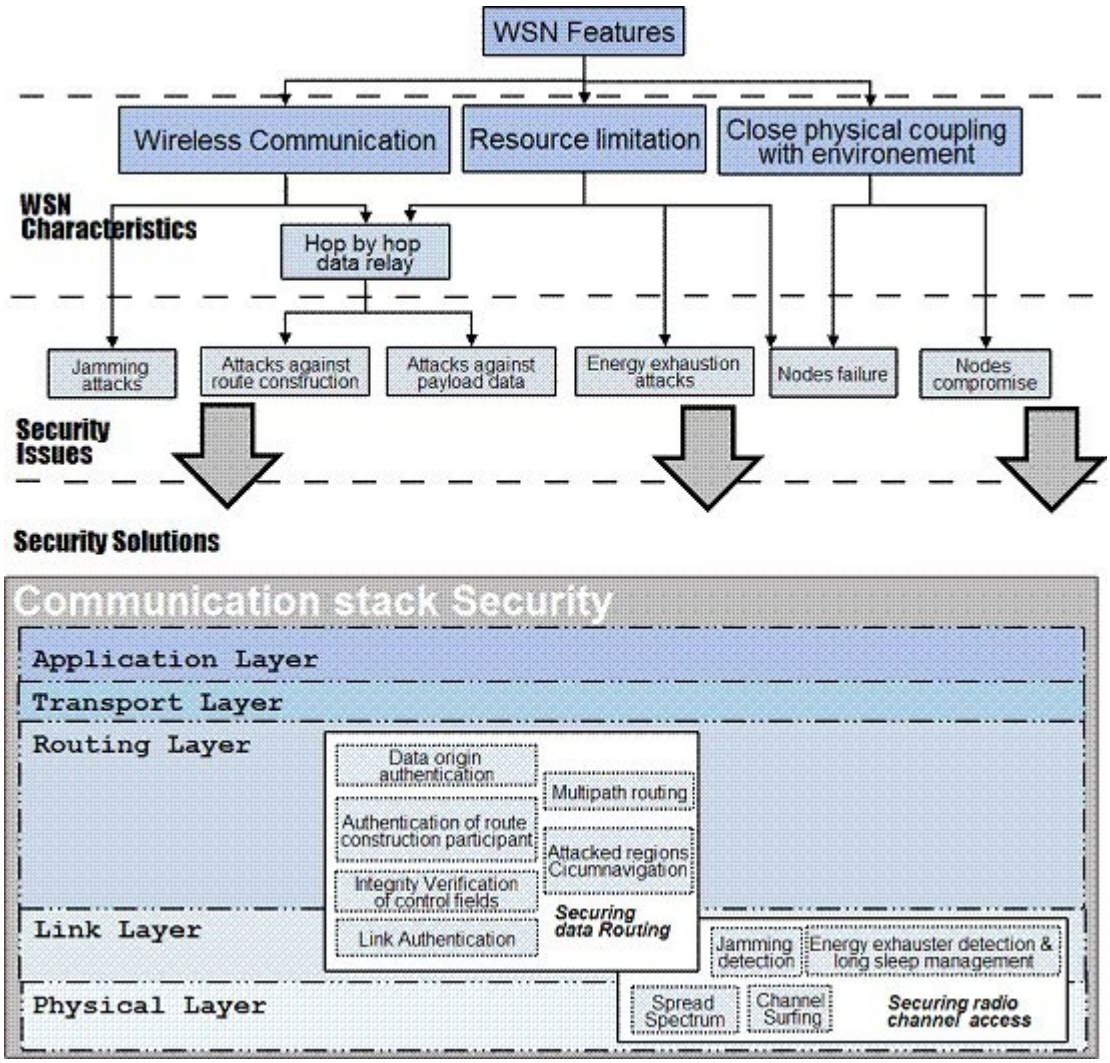


Image 42 : Taxonomie des challenges et solutions de sécurité dans les RCSF

B. La gestion de clés dans les RCSF

La gestion de clés fournit des mécanismes éfficaces, sécurisés et stables de gestion de clés utiliséees dans les opérations cryptographiques. Par conséquence, la gestion de clés est un service primordial pour la sécuritée de n'importe quel système basé sur la communication. Sous les contraintes des RCSF, la conception d'un système de gestion de clés est un grand défis. Sélectionner une solution cryptographique appropriée pour les RCSF est un autre défis

1. La fonction de gestion de clés dans les RCSF

Définition : Gestion de clés

La gestion des clés est un des aspects les plus difficiles de la configuration d'un système cryptographique de sécurité. Pour qu'un tel système fonctionne est 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 sure. 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.

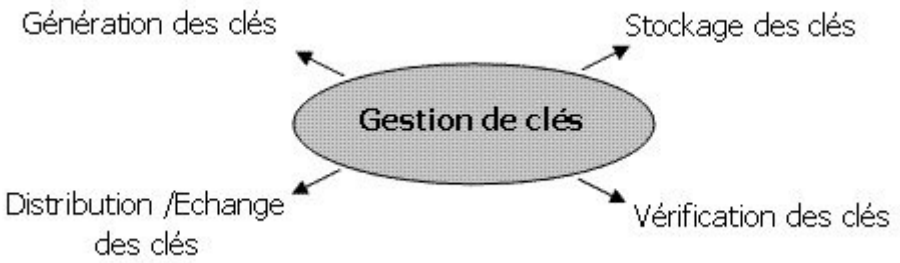


Image 43 : Fonctions de la gestion de clés

Pourquoi la gestion de clés dans les RCSF ?

Après leur déploiement, les capteurs ont besoin d'établir des clés cryptographiques avec leurs voisins pour assurer des services de sécurité:

- Sécuriser le routage
- Sécuriser l'agrégation
- Coopération (authentification), etc.

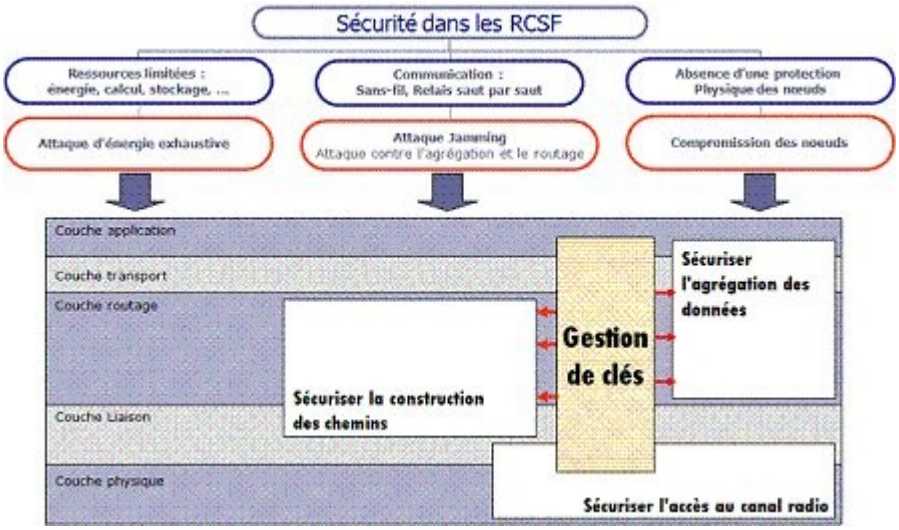


Image 44 : Positionnement de la gestion de clé dans un RCSF sécurisé

Contraintes de conception

La figure suivante résume les contraintes découlant des propriétés des RCSF, à prendre en compte dans la conception d'une solution de gestion de clés pour les RCSF.

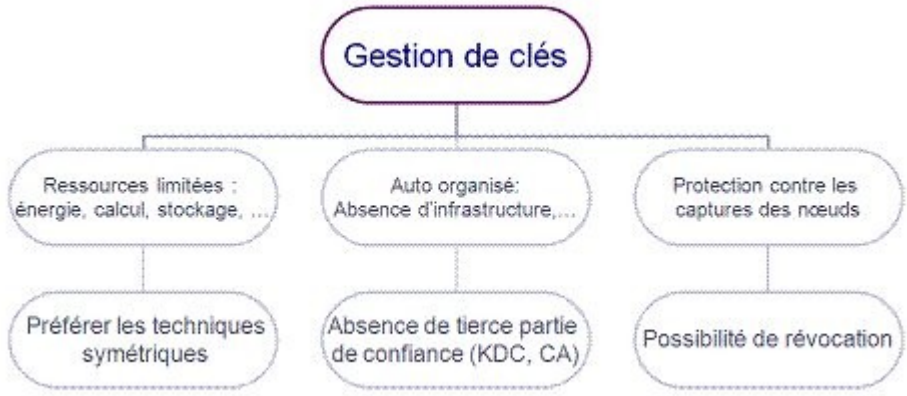


Image 45 : Contraintes de conception de solutions de gestion de clés

Remarque : Systèmes asymétriques ou symétriques ?

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 calcule des algorithmes à clé publiques, la consommation d'énergie due à la transmission des certificats, et le stockage des clés connues pour être plus grandes que les clés symétriques. Employer des mécanismes de clés symétriques pour l'établissement de la confiance réduit considérablement la consommation d'énergie des nœuds capteurs et l'espace de stockage réservé pour accueillir ces clés. Cependant, l'échange de clés dans les systèmes à clés symétriques est beaucoup plus compliqué. Habituellement, une seule clé symétrique est utilisée entre deux parties communicantes, sur une seule session ou sur une période limitée.

Taxonomie

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 RCSF, 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 RCSF sont basés sur la cryptographie symétrique. Le problème majeur avec la cryptographie symétrique est de pouvoir trouver une méthode qui facilite l'établissement des clés entre les noeuds. La solution commune est d'utiliser une méthode de pré-distribution, dans laquelle les clés sont chargées dans les noeuds capteurs avant le déploiement. La figure suivante illustre une taxonomie des solutions de gestion de clés basée sur la pré-distribution. Dans cette taxonomie, les protocoles sont classés selon la façon avec laquelle les noeuds voisins partagent des clés communes (probabiliste ou déterministe), et selon la topologie du réseau (hiérarchique ou plate).

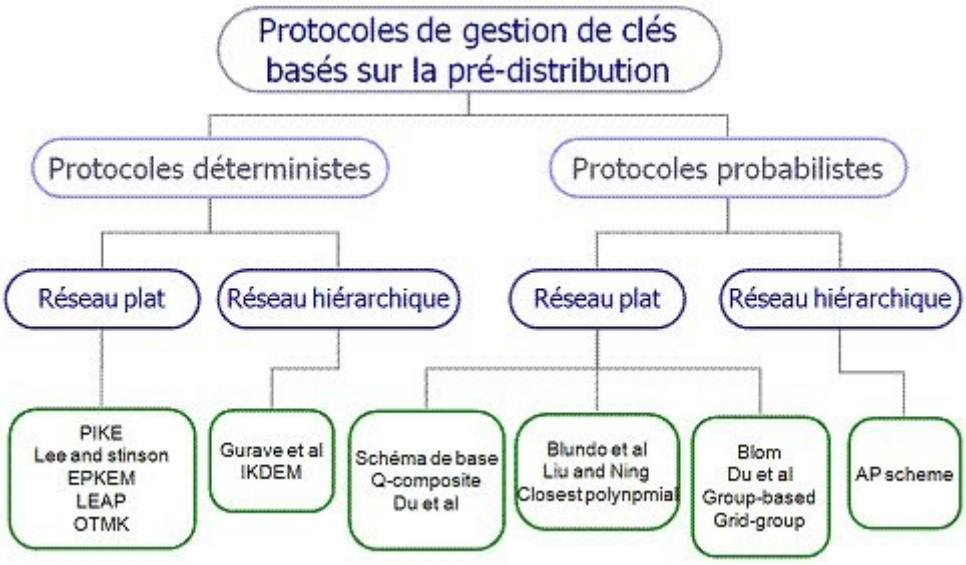


Image 46 : Taxonomie de pré-distribution de clés pour les RCSF

2. Schéma aléatoire de pré-distribution de clés de L.ESCHENAUER et D.GLIGOR

Aperçus

Eschenauer et Gligor ont proposé un schéma de gestion de clé basé sur la probabilité de partager une clé entre les nœuds d'un graphe aléatoire. Il fournit des techniques pour la pré- distribution de clé, la découverte de la clé partagée, l'établissement de chemin de clé, et la révocation de clé.

L'idée maîtresse de ce schéma, est de distribuer aléatoirement un certain nombre de clés, issues d'un ensemble fini à chaque nœud du réseau avant son déploiement. Deux nœuds quelconques seront en mesure de s'échanger des messages sécurisés s'ils possèdent une clé commune.

Méthode : Phase de pré-distribution de clés

Un grand ensemble S de clés est générée (2^{17}-2^{20} Clés). Pour chaque nœud, m clés sont choisies au hasard de l'ensemble S (S= {(k_{id1}, key_1), (k_{id2}, key_2),....}). Ces m clés sont stockées dans la mémoire du nœud et forment le trousseau de clés du nœud. Le nombre de clés |S| de l'ensemble est choisi de telle manière que deux sous-ensembles aléatoires de S de taille m auront une certaine probabilité p d'avoir au moins une clé en commun, par exemple pour une probabilité p=0.5 on a besoin d'un sous ensemble de taille m=75 clés de l'ensemble S de taille |S|=10,000 clés.

Méthode : Phase de découverte de clés partagées

Les nœuds découvrent leurs voisins et plus particulièrement ceux avec qui ils sont en mesure de communiquer de façon sécurisée car ils possèdent une clé identique dans leur trousseau de clés respectif. Le protocole peut être de diffuser la liste des identités k_{idi} des clés possédées. La clé partagée devient la clé de session du lien entre les deux nœuds. La figure suivante illustre cette phase :

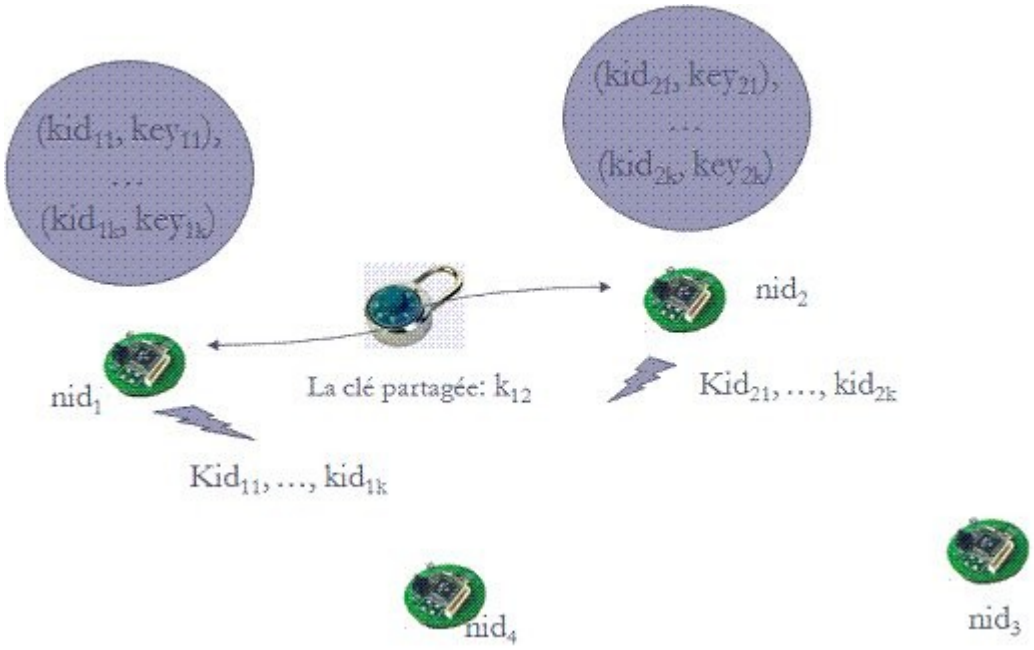


Image 47 : Découverte des clés partagées

Méthode : Phase d'établissement de chemin de clé

Après la phase de découverte de clés partagées, le réseau devient un graphe connecté formé de quelques liens sécurisés. Les nœuds peuvent alors utiliser les liens existants pour mettre en place des clés partagées avec leurs voisins qui ne partageaient pas de clé en commun avec eux. La figure suivante illustre cette phase :

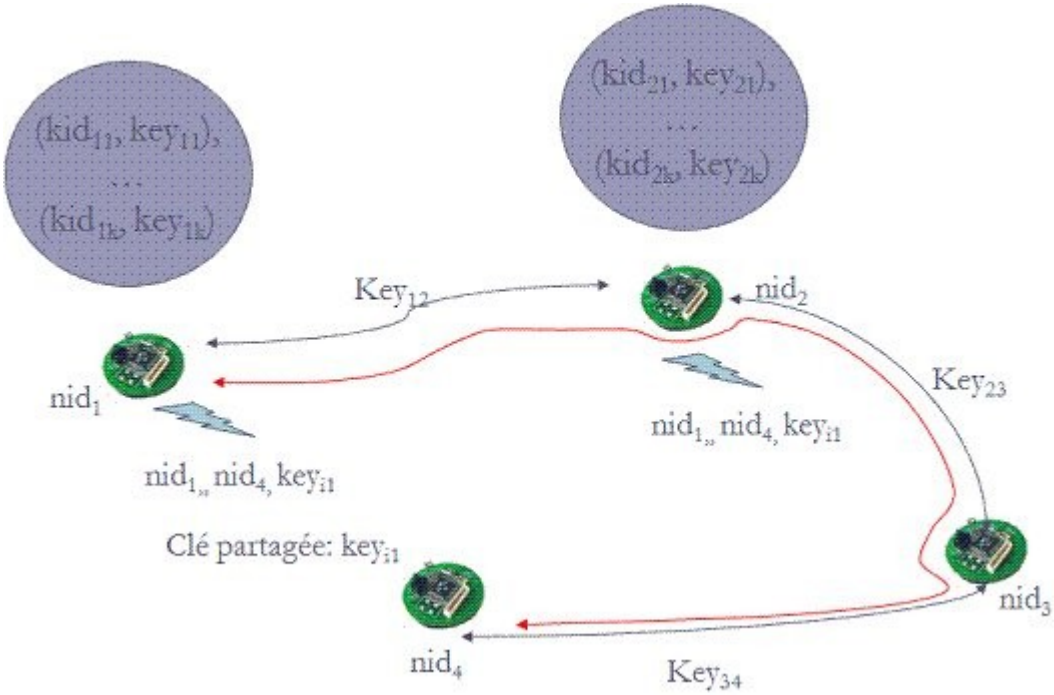


Image 48 : Etablissement de chemins sécurisés

Méthode : La révocation de clés

La révocation d'un nœud compromis se fait par l'élimination de leur trousseau de clés. Pour cela, un nœud contrôleur (qui a une grande connectivité et peut être

mobile) annonce un message simple de révocation contenant une liste signée de k identificateurs des clés (k_{idi}) pour que ces clés soient retirées des trousseaux de clés des autres noeuds. La liste des identités est signée par une clé de signature K_e générée par le nœud contrôleur et envoyée en unicast à chaque nœud i en la chiffrant avec la clé K_{ci} (la clé K_{ci} est partagée entre le contrôleur et le ième nœud pendant la phase de pré- distribution de clés). Quelques liens seront disparus à cause de la suppression de clés du noeud compromis ce qui nécessite une reconfiguration de ces liens (par la découverte de clés partagées ou l'établissement de chemin de clé). La figure suivante illustre cette phase :

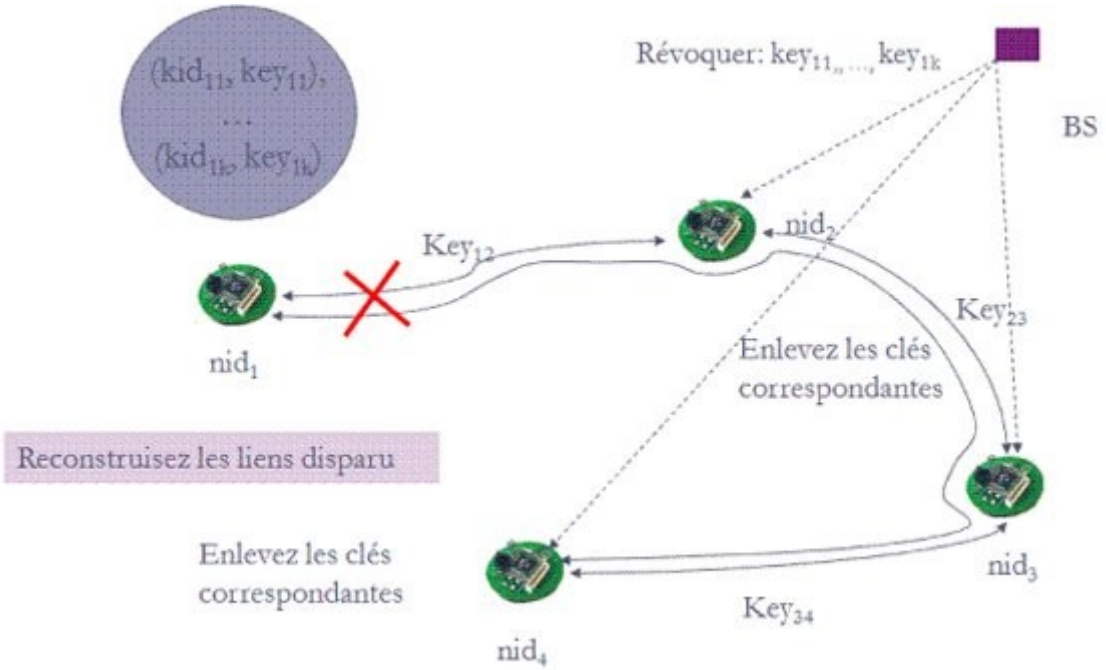


Image 49 : Révocation de clés

Complément : Schéma q-composite de H.CHAN, A.PERRIG et D.SONG

Ce schéma est identique à celui de Eschenaur et Gligor sauf qu'au lieu d'exiger le partage d'une clé commune pour sécuriser un lien, une paire de nœud doit partager q clés avec q > 1 pour établir un lien sécurisé. La nouvelle clé utilisée pour la communication entre ces deux nœuds est le hash de toutes les clés partagées, par exemple pour deux nœuds quelconque qui partage q' clés ($q' \geq q$) la clé utilisée pour la communication est $K = hash (k_1 || k_2 || ... || k_{q'})$. Plus le nombre de clé partagées augmente plus la résilience contre la capture du nœud augmente. Autrement, lorsque le nombre, exigé, de clés partagées augmente, il devient plus difficile à un attaquant avec un ensemble donné de clés de casser un lien. Cependant, pour préserver une probabilité donnée p que deux nœuds partageant des clés suffisantes pour établir un lien sécurisé, il est nécessaire de réduire la taille de l'ensemble de clés S. Ceci permet à un attaquant de gagner un plus grand échantillon de S en cassant peu de nœuds.

La figure suivante illustre un exemple de partage de clés avec q=2.

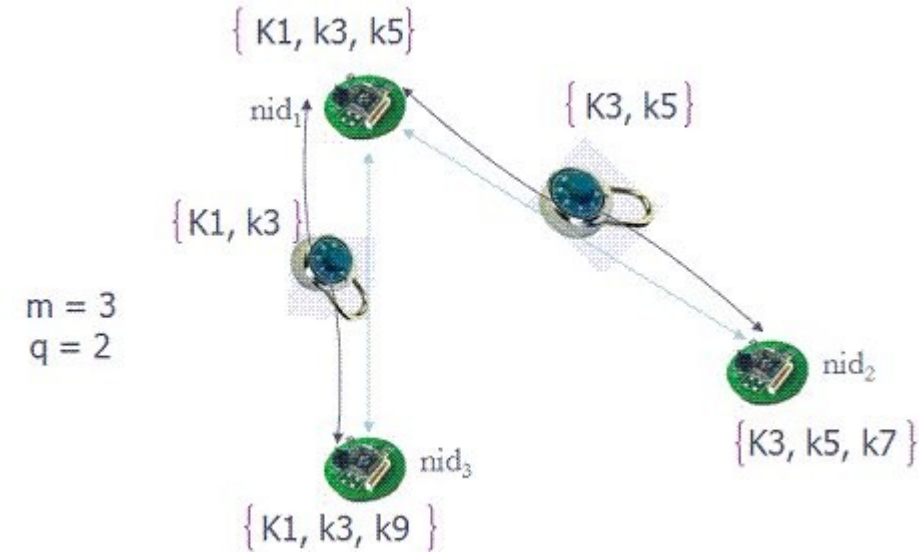


Image 50 : Schéma q-composite

3. LEAP

Aperçus

LEAP est un protocole déterministe de gestion de clés pour les réseaux de capteurs sans-fils. Le mécanisme de gestion de clés fourni par LEAP support le traitement interne "in-network processing" tout en limitant l'impact de sécurité d'un noeud compromis sur son voisinage immédiat dans le réseau. LEAP support l'établissement de quatre type de clés pour chaque noeud : cllé individuelle, clé par paire, clé de groupe et clé globale.

Fondamental : Hypothèse de fonctionnement

LEAP est basé sur une clé initiale transitoire K_{IN} chargée dans chacun des noeuds du réseau. Les auteurs de LEAP supposent que pour compromettre un noeud, l'adversaire nécessite un temps minimal T_{min} : c'est le temps de brancher un câble série et le temps de copier le contenu de la mémoire du noeud compromis. LEAP exploite ce temps (de confiance) pour permettre à deux noeuds voisins d'établir, d'une manière sécurisée, une clé symétrique de session à partir de la clé initiale transitoire K_{IN}. Après T_{min}, la clé K_{IN} est supprimée de la mémoire du noeud.

Méthode : Chargement de la clé initiale

Le contrôleur (SB) génère une clé initiale K_{IN} et charge chaque noeud avec cette clé. Chaque noeud u dérive une clé principale (Maser Key) $K_u=f_{KIN}(u)$, f_k étant une fonction pseudo-aléatoire.

Méthode : Découverte des voisins

Immédiatement après son déploiement, le noeud u essaye de découvrir ses voisins en diffusant un message HELLO qui contient son id. Aussi, il initie un timer qui sera déclenché après le temps T_{min}. Le noeud u attend un ACK de chacun de ses voisins v qui contient l'identificateur de v. l'ACK est authentifié en utilisant la clé principale K_v, qui est dérivée comme suit : $K_v=f_{KIN}(v)$. Comme le noeud u a la clé K_{IN}, il pourra aussi dériver K_v, ainsi il pourra vérifier l'authenticité du ACK reçus :

u ==> *, u

v ==> u, v | MAC(K_v, u|v)

Méthode : Etablissement de la clé par-paire

Le noeud u calcule sa clé par paire K_{uv} avec v, comme suit :

$K_{uv}=f_{Kv}(u)$.

Le noeud v peut de même calculer K_{uv} de la même manière.

K_{uv} sert comme clé entre u et v.

Méthode : Effacement des clés

Lorsque le timer expire après T_{min}, le noeud u efface K_{IN} et toutes les clés principales K_v de ses voisins. Il est à noter que le noeud u n'efface pas sa clé principale K_u.

Remarque : Sécurite de LEAP

A la fin de ces quatre étapes, le noeud u aura établi une clé par paire partagée avec chacun de ses voisins. Cette clé sera utilisée pour sécuriser les données échangées entre eux. de plus, aucun noeud dans le réseau ne possède la clé K_{IN}. Un adversaire peut écouter clandestinement tout le trafic dans cette phase, mais sans la clé KIN il ne peut injecter des informations incorrectes ou déchiffrer les messages. Un adversaire compromettant un noeud après T_{min}, obtient seulement les clés du noeud compromis. Quand un noeud compromis est détecté, ses voisins suppriment simplement les clés qui ont été partagée avec lui.

C. Sécurité du routage dans les RCSF

La couche de routage est le module responsable d'acheminer correctement une donnée d'un point du réseau vers un autre. Pour assurer ce rôle, la couche de routage est composée de deux blocs fonctionnels : la construction de routes et le relais des données. Le premier composant permet de construire un backbone connectant les noeuds aux destinations désirées via un ensemble de chemins. Le deuxième composant quant à lui utilise ce backbone afin d'acheminer les données captées vers les utilisateurs finaux. Un adversaire désirant attaquer le réseaux peut alors s'en prendre à l'un des deux composant qu'il faudra protéger.

1. Le routage dans les RCSF : menaces et solutions

Attaques sur les protocoles de routage dans les RCSF

Vus les contraintes des RCSF, la plus part des protocoles de routage sont assez simple, et par conséquent assez vulnérables aux attaques. Un noued malicieux peut opérer sur deux niveaux :

- Les données échangées entre les noeuds
- La topologie du réseau créée par le protocole

Ces attaques peuvent être classées en deux catégories : actives et passives.

a) Attaques actives

Attaque de "jamming"

Vu la sensibilité du média sans fil au bruit, un noeud peut provoquer un déni de service en émettant des signaux à une certaine fréquence. Cette attaque peut être très dangereuse car elle peut être menée par une personne non authentifiée et

étrangère au réseau.

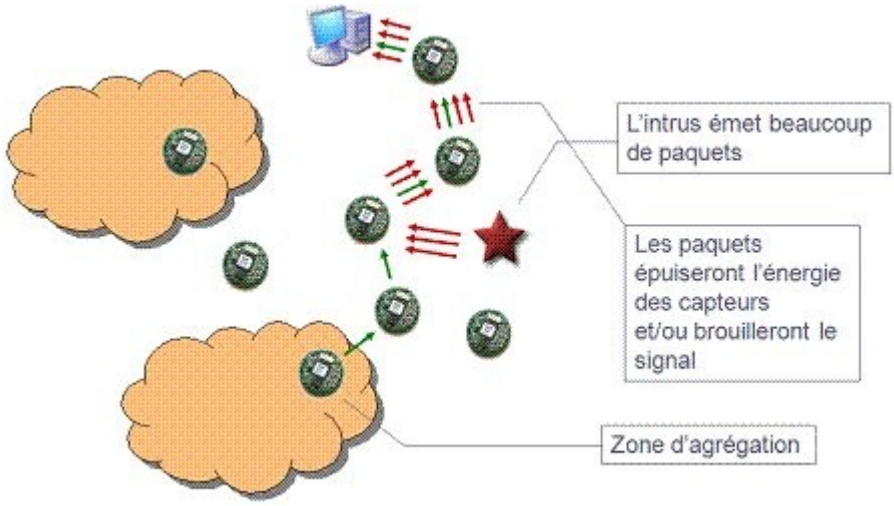


Image 51 : Attaque de "jamming"

Sink hole

Dans une attaque sinkhole, le noeud essaye d'attirer vers lui le plus de chemins possibles permettant le contrôle sur la plus part des données circulant dans le réseau. pour ce faire, l'attaquant doit apparaître aux autres comme étant très attractif, en présentant des routes optimales.

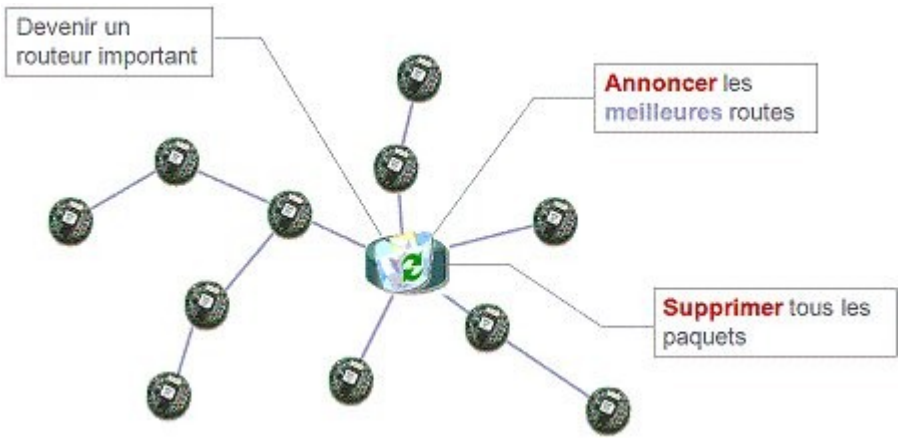


Image 52 : Attaque sinkhole

Attaque 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 peut se faire de deux manières: – Multi-sauts: l'encapsulation multi-sauts permet de cacher les noeuds se trouvant entre les deux attaquants. Donc, les chemins passant par le noeud malicieux apparaissent plus courts. Cela facilite la création de sinkholes avec des protocoles qui utilisent le nombre de sauts comme métrique de choix de chemins. – Communication directe: les routes passant par les attaquants sont plus rapides, car ils sont à un saut. Donc, cette technique peut être employée contre les protocoles qui se basent sur la latence des routes ou ceux qui utilisent la première route découverte.

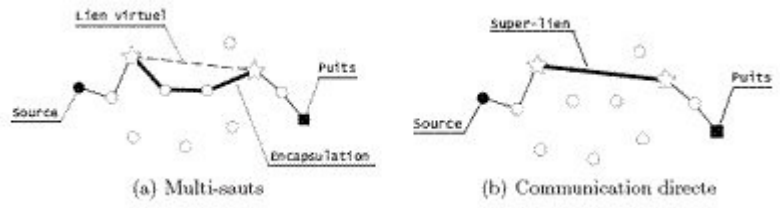


Image 53 : Attaque Wormhole

Routing table poisoning

Certaines optimisations ont été développées afin d'augmenter la connaissance des chemins. Lorsqu'un noeud entend (en mode promiscuous) une information de routage, il met à jour sa table de routage locale en conséquence. Un noeud malicieux peut émettre un nombre important de fausses informations, remplissant ainsi les tables de routage des noeuds. Comme ces tables possèdent des tailles limitées, cela va engendrer un débordement, et les tables ne contiendront que de fausses routes.

Sybil attack

Dans certains algorithmes, la fiabilité du routage est implémentée par l'instauration d'une redondance de chemins. Un attaquant peut altérer ce genre de systèmes en "endossant" plusieurs identités, ce qui permet de créer plusieurs routes passant par le noeud malicieux, qui ne sont en réalité qu'un seul chemin.

Hello flooding

La faible portée de capteurs et la présence d'attaquant de classe laptop ont permis l'introduction d'une nouvelle attaque : hello flooding. Cette attaque se base sur le fait que la plus part des liens entre l'attaquant laptop et les capteurs sont unidirectionnels. Donc, un attaquant peut diffuser l'information d'une route optimale vers tous les noeuds du réseau en émettant avec un signal puissant, et tous les noeuds mettrons à jour leurs tables locales. Lorsqu'un noeud veut communiquer, il ne pourra pas utiliser cette route car le prochain saut, qui est l'attaquant, est hors portée.

b) Attaques passives

Lack of cooperation ou Selective Forwarding

Tous les protocoles de routage supposent que les noeuds sont "honnêtes" et vont relayer normalement les paquets qui transitent par eux. Cependant, un attaquant peut violer cette règle en supprimant la totalité ou une partie de ces paquets. De plus, si l'attaquant a auparavent utilisé une attaque sinkhole, il devient un routeur important dans le réseau. Donc, en abandonnant son rôle de routeur, les performances du systèmes seront gravement dégradées.

Eavesdropping

Comme le média sans fil est un média ouvert, un noeud peut entendre toutes les communications de ses voisins. Cela peut divulguer d'importantes informations, comme la localisation d'un noeud important. La combinaison avec une attaque sinkhole aggrave d'avantage l'impact de cette attaque.

c) Types de solutions

Nous distinguons trois niveaux de solutions aux attaques sur le routage de données dans les RCSF :

1. La prévention contre les attaques actives: dans cette catégorie on utilise généralement des mécanismes cryptographiques pour protéger la signalisation qui sert à la construction de routes. C'est généralement des mécanismes d'authentification et de contrôle d'intégrité qui sont utilisés pour empêcher un noeud malicieux d'injecter, de modifier et/ou de supprimer une information qui servira pour la découverte, la construction ou la maintenance d'une route.
2. La détection de comportements suspects: dans cette catégorie on tente de déceler des comportements qui témoignent d'une attaque passive (manque de coopération, refus de relais de paquets, etc.)
3. La tolérance : dans cette catégorie, on introduit des mécanismes de tolérance de défaillance de noeuds à cause d'attaques ou de pannes. Le routage multi-chemin en est un exemple typique.

La figure suivante résume les catégories de solutions à préconiser pour faire face à différents types d'attaques.

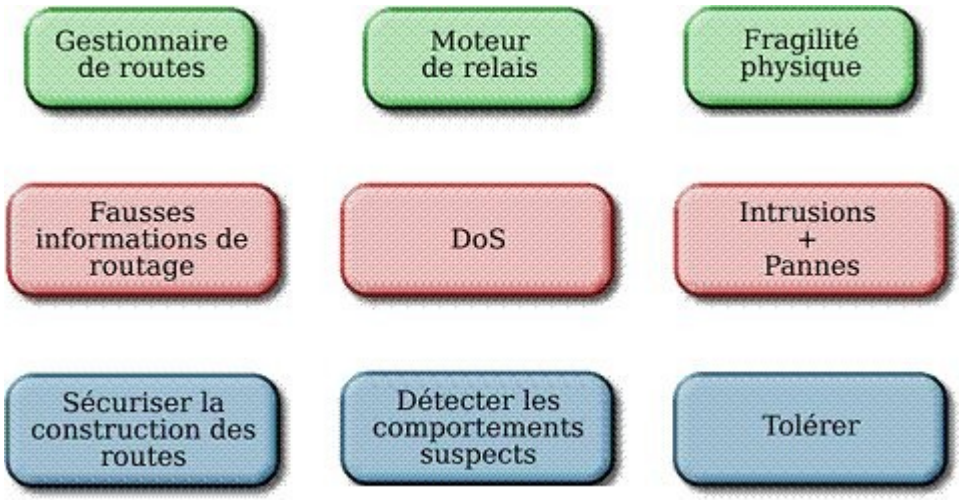


Image 54 : Catégories de solutions contre les attaques sur le routage

2. INSENS (Intrusion-tolerant routing for wireless sensor networks)

Aperçus

L'idée du protocole est de permettre à la SB de tracer une cartographie correcte du réseau qui permettra d'établir les tables de routage pour chaque capteur. Ces tables seront transmises par la suite aux noeuds concernés de façon sécurisée. Le protocole vise deux objectifs :

- **Opérer correctement en présence d'intrus**: Pour cela, INSENS est un protocole tolérant aux intrusions, qui assure le routage même en présence d'intrus. Il construit plusieurs chemins indépendants pour chaque couple de communiquants. Les chemins indépendants ne partagent qu'un nombre réduit de noeuds/liens (idéalement, ils partagent seulement la source et la destination). Avec cette propriété, un intrus ne pourra altérer que les communications traversant un seul chemin.
- **Scalabilité et économie d'énergie**: Le calcul des chemins indépendants et l'établissement des tables de routage représentent une tâche assez lourde. INSENS effectue ces calculs dans les SB (station de base). Les résultats sont ensuite transmis vers chaque noeud d'une manière sécurisée.

Le protocole se déroule selon les phases suivantes.

Méthode : Initiation authentifiée de la construction de l'arbre

La SB doit d'abord dresser un arbre couvrant tout le réseau, dont elle est la racine. Cet arbre permettra d'acheminer les messages de contrôle entre les capteurs et la SB. Afin d'éviter le spoofing de SB, INSENS emploie un mécanisme de broadcast authentifié. Pour ce faire, la SB génère une chaîne de hachage à sens unique $(n_i)_{0\leq i\leq k}$ comme suit :

$n_{i+1}=h(n_i)$, $0<i<k$ où n_0 est choisi aléatoirement, et n_k est connu par tous les noeuds, et h est une fonction de hashage à sens unique.

Périodiquement, la SB génère une requête pour re-construire les tables de routage des capteurs. Le format du message est le suivant :

type|ows|size|path|$MACR_x$

Le champ ows (One-Way Sequence number) désigne la valeur courante de la chaîne de hachage (i.e. pour la requête i, $ows=n_i$). Chaque noeud maintient localement la dernière valeur reçue de ows, désignée par ows_{fresh}. Lorsqu'un noeud x reçoit une requête, il vérifie sa fraîcheur et son authenticité par la relation suivante:

Trouver j, tel que $j>0$ et $ows=h^j(ows_{fresh})$

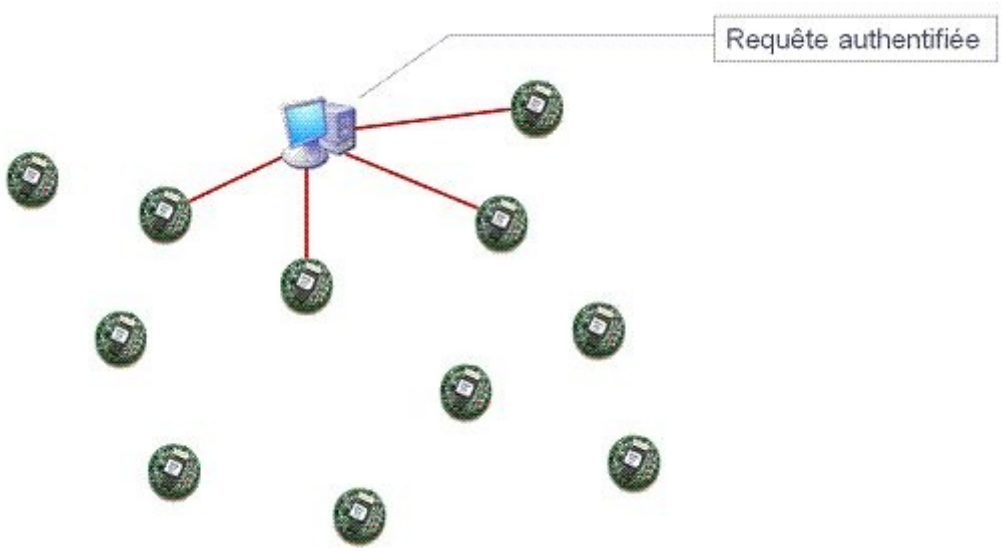


Image 55 : Requête authentifiée de construction de l'arbre

Méthode : Construction de l'arbre par relayage de la requête

Un noeud x qui reçoit le message de requête précédent, ajoute son identificateur au champ path et calcule le champ $MACR_x$:

$MACR_x = MAC(k_x, size|path|ows|type)$

où k_x représente la clé sectrète partagée entre le noeud x et la SB. Le noeud doit aussi choisir son "upstream" vers la SB, qui représente le premier voisin qui a émis une requête valide, et sauvegarde son MACR, qui sera désigné par $MACR_{upstream}$.

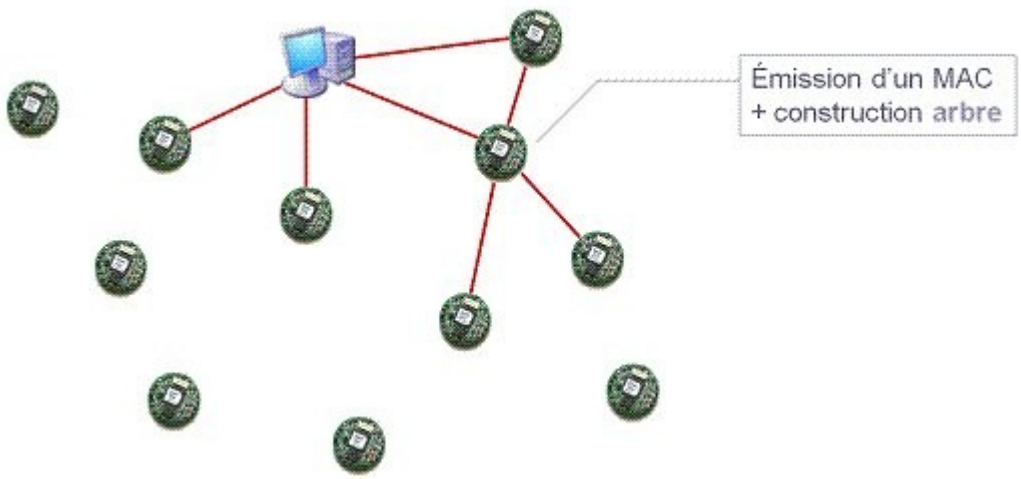


Image 56 : Construction de l'arbre

Méthode : Route feedback

Après l'émission de la requête, le noeud attend un certain temps pour envoyer un feedback à la SB. Cette période lui permet de récolter les informations sur son voisinage, qui vont permettre à la SB d'établir une vue globale du réseau d'une manière sécurisée. Un message feedback contient les informations suivantes:

$type|ows|path\_info|nbr\_info|MACR_{upstream}|MACF_x$

Le champ "path_info" contient la liste des noeuds entre le noeud x et la SB, l'identification de x et son MACR:

$ID_x|size|path|MACR_x$

Le champ "nbr_info" contient la liste des identificateurs des voisins ainsi que leurs MACR :

$size|ID_a|MACR_a|ID_b|MACR_b|\ . \ . \ .$

Le champ $MACF_x$ est calculé comme suit : $MACF_x=MAC(k_x, path\_info|nbr\_info|ows|type)$

Pour renforcer la sécurité, le routage du message s'effectue grâce au champ $MACR_{upstream}$. Ainsi, lorsqu'un noeud reçoit un message feedback, il compare la valeur du $MACR_{upstream}$ reçue avec sa valeur liée au ows reçu, et relaie éventuellement le message.

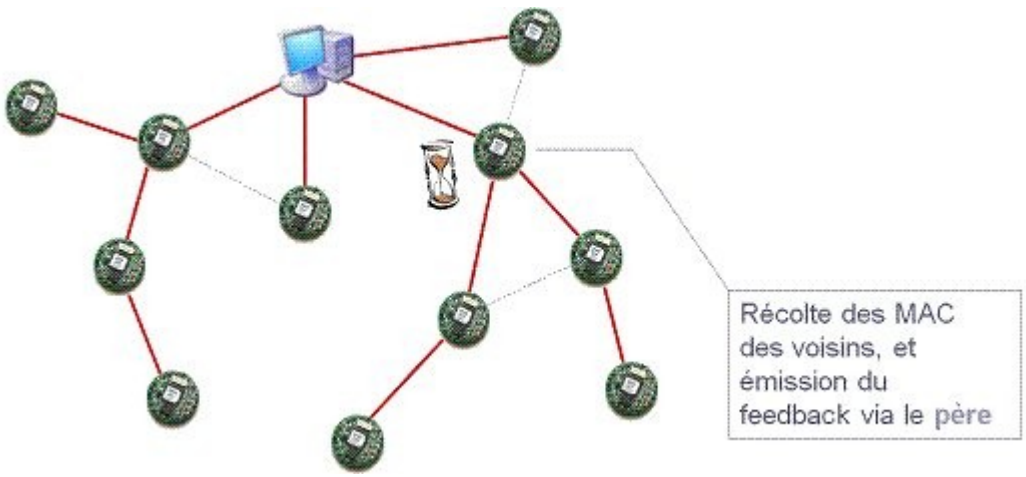


Image 57 : Route feedback

Méthode : Construction des tables de routage

Après l'émission de la requête, la SB attend une certaine période avant d'entamer la construction des tables de routage. Pendant cette durée, elle traite les messages de feedback reçus afin de dresser le graphe du réseau. Pour vérifier l'information du feedback d'un noeud x, la SB recalcule le $MACF_x$. Elle doit aussi comparer les valeurs des MACR de chaque voisin de i avec la valeur $MACR_i$ reçue dans le feedback du noeud i. Ainsi, les informations falsifiées sont rejetées, et un graphe correct peut être établi. La SB peut ensuite rechercher les chemins indépendants et construire en conséquence les tables de relais pour chaque noeud. Les chemins indépendants sont choisis de telle sorte qu'ils regroupent le moins de noeuds possible en commun. L'algorithme utilisé emploie une méthode simple qui cherche d'abord le chemin le plus court entre chaque couple de communiquants. Par la suite, l'algorithme essaye de trouver un autre chemin dans le sous-graphe ne contenant pas les noeuds du premier chemin, leurs voisins et les voisins des voisins. Si cela est impossible, le procédé est réitéré en rajoutant l'ensemble des voisins des voisins, puis en cas d'échec on rajoute l'ensemble des voisins. Pour chaque capteur, la SB calcule une table de relais qui contient une entrée pour chaque chemin passant par le capteur. Cette table est encapsulée dans le message suivant:

type|ows|size|routingTable|MAC

où le MAC est calculé comme suit :

$MAC=MAC(k_x,type|ows|size|table)$

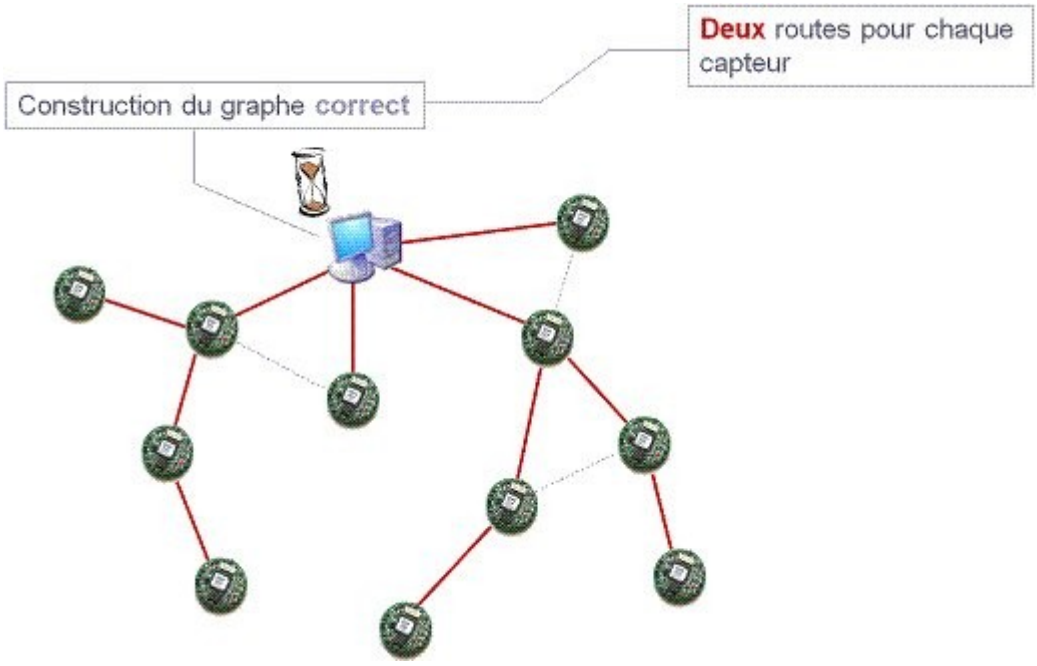


Image 58 : Construction et distribution des tables de routage

3. SecRoute

Aperçus

Le protocole SecRoute est un protocole de routage hiérarchique sécurisé. Le réseau est organisé en clusters ayant chacun un chef. Le noeud collecteur est supposé connaître cette organisation du réseau, et doit maintenir localement une table contenant une clé secrète de chaque capteur. Cette clé est supposée pré-chargée dans chaque noeud. De plus, chaque cluster doit posséder une clé permettant de

sécuriser les échanges intra-cluster. Cette clé doit être connue par le cluster-head et tous les noeuds du groupe. Le protocole SecRoute ne spécifie pas l'algorithme de construction de clusters, et suppose que les clusters ainsi que leurs clés sont établis par un autre protocole, comme LEAP.

Remarque : Propriétés

Le protocole possède les propriétés suivantes:

- Les paquets de routage ne sont pas volumineux, car ils ne contiennent qu'une information partielle sur le chemin parcouru.
- Le protocole utilise une architecture à deux niveaux, dans laquelle les chefs agrègent les données des membres puis les transmettent au noeud collecteur.
- Le protocole emploie seulement des méthodes de chiffrement symétrique.
- Pour des raisons de sécurité, le protocole remplace les unicasts par des broadcasts locaux ciblés. En effet, en évitant les unicasts, un message émis est reçu par tous les voisins. Donc, cela permet de vérifier, lors du relais, l'intégrité du message émis par le prochain saut.
- Chaque capteur stocke une table de routage ayant le format suivant :

Source	*Pre*	*Next*
ID_{Soure}	ID_{pre_2}, ID_{pre_1}	D_{next_1}, D_{next_2}
⋮	⋮	⋮

Image 59 : Format de la table de routage dans SecRoute

La table est organisée suivant l'adresse des sources. Le champ Pre (respectivement Next) indique les deux prochains sauts vers la SB (respectivement source) sur le chemin entre la source et la SB.

Méthode : Découverte des chemins

Le noeud source initie une découverte des chemins en émettant vers ses voisins directs un paquet de requête RREQ contenant les informations suivantes:

$ID_{source}|ID_{sink}|ID_{RREQ}|N_{source}$ $MAC(K_{source}, ID_{source}|ID_{sink}|ID_{RREQ}|N_{source})$ avec :

- ID_{source}, ID_{sink}, ID_{RREQ}: les identificateurs de la source, du collecteur et de la requête.
- N_{source} : un nonce généré par la source.

Lorsqu'un noeud reçoit une requête, elle n'est acceptée qu'avec l'unicité de son identificateur ID_{REQQ}. Il met à jour par la suite sa table de routage en utilisant l'information des deux sauts précédents vers la source. Avant de relayer la requête, le noeud remplace les valeurs de ID_{pre} et ID_{this} par, respectivement, ID_{this} et son identificateur :

$ID_{this}|ID_{pre}|ID_{source}|ID_{sink}|ID_{RREQ}|N_{source}$ $MAC\ (K_{source}, ID_{source}|ID_{sink}|ID_{RREQ}|N_{source})$

Méthode : Relais de la réponse

Lorsque le collecteur reçoit la première requête, il vérifie le MAC construit par la source en utilisant la clé relative à son identificateur ID_{source}, sauvegardée dans la table locale. Si le MAC est correct, le collecteur met à jour sa table de routage en utilisant les champs ID_{pre} et ID_{this}. Il génère ensuite une réponse RREP ayant le format suivant :

$ID_{pre}|ID_{this}|ID_{next}|ID_{sink}|ID_{RREQ}|N_{sink}$ $MAC\ (K_{source}, ID_{source}|ID_{sink}|ID_{RREQ}|N_{sink})$

La requête est émise en broadast local, ciblé à l'aide du champs ID_{next}. Lorsque le

capteur voisin ayant l'identificateur ID_{next} reçoit cette réponse, il met à jour sa table de routage en conséquence, puis remplace les champs ID_{pre} et ID_{this} par ID_{this} et son identificateur. Il doit aussi modifier le champ ID_{next} par l'identificateur connus lors de la phase de découverte de chemins. Si le noeud ne reçoit pas la réponse émise par ID_{next} après un certain temps, il ignore toutes les requêtes émises pendant la prochaine phase de découverte. Le noeud de relais doit aussi vérifier que la réponse émise par le prochain saut est valide, en s'assurant qu'elle est bien destinée au noeud à deux sauts contenu dans le chemin vers la source (i.e. le champ ID_{pre2} de la table de routage). Lorsque la source reçoit la première réponse, elle vérifie le MAC généré par le collecteur et met à jour sa table de routage en ajoutant ID_{this} et ID_{pre} comme prochains sauts vers le collecteur.

Méthode : Relais des données

Le relais des données est effectué en deux étapes. Les noeuds membres émettent leurs données vers le chef du groupe, qui va par la suite envoyer le résumé vers le collecteur. L'établissement du chemin entre le chef et le collecteur s'effectue grâce au procédé décrit précédemment, i.e. : le chef représente la source de son groupe. Pour les communications intra-groupe, le protocole utilise la clé de cluster CK établie lors de sa création. Chaque donnée D d'un capteur est émise vers le clusterhead ID, encapsulée comme suit :

$ID|\{D\}CK|MAC(CK, ID|\{D\}CK)$

Le chef agrège les données des membres après vérification des MAC de chaque paquet, puis émet le résumé vers le collecteur. Pour cela, le chef opère comme étant la source de la donnée. Si aucune route vers le collecteur n'est connue, le procédé de découverte de chemins doit être effectué. En utilisant le chemin construit, le chef émet la donnée dans le paquet suivant :

$ID_{this}|ID_{next}|ID_{source}|ID_{sink}|Q_{ID}|\{D\}K_{source}$ MAC $(K_{source}, ID_{source}|ID_{sink}|Q_{ID}|\{D\}K_{source})$

Un noeud intermédiaire avec le même identificateur que ID_{next} relaie le paquet en remplaçant ID_{this} et ID_{next}. Si un noeud de relais ne reçoit pas le paquet émis par le prochain saut, une maintenance de la route doit être effectuée. Pour cela, il doit émettre vers la source un message d'erreur permettant d'enclencher à nouveau la procédure de découverte de routes. De plus, le prochain saut est ajouté à la liste noire (black list). Cette liste contient les identificateurs dont le noeud doit ignorer leurs paquets de réponse. Lorsque le paquet atteint le collecteur, il vérifie le MAC en utilisant la clé de la source et peut donc décrypter la donnée et l'utiliser.

D. Sécurité de l'agrégation dans les RCSF

1. Attaques sur l'agrégation de données dans les RCSF

Cohabitation entre sécurité et agrégation

L'agrégation de données est nécessaire dans les RCSF pour minimiser les transmissions redondantes et donc économiser de l'énergie.

Pour réaliser une opération d'agrégation, un noeud intermédiaire doit avoir accès aux données transmises par ses paires pour calculer l'information utile en utilisant une fonction d'agrégation comme : la moyenne, le maximul, le minimum etc.

Un noeud malicieux peut alors attaquer ce schéma en injectant de fausses données dans le réseau ou en falsifiant le résultat d'une opération d'agrégation. Dans ce cas, le noeud malicieux réussira à falsifier l'information captée dans tout une zone.

La figure suivante montre le risque qui peut être encouru par une mauvaise

opération d'agrégation.

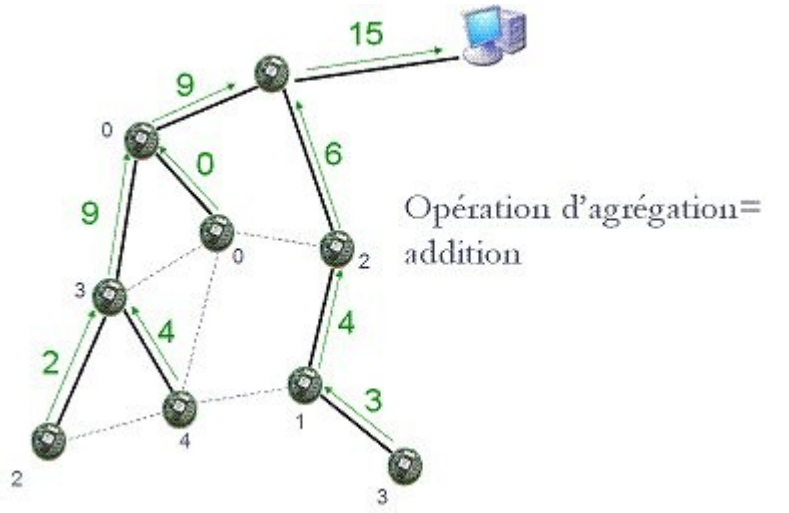


Image 60 : Fonctionnement correcte de l'agrégation

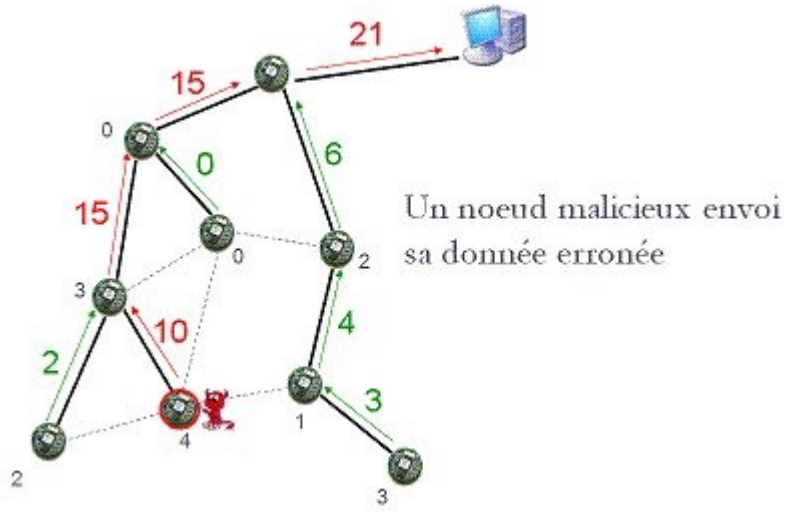


Image 61 : Un malicieux injecte une fausse donnée

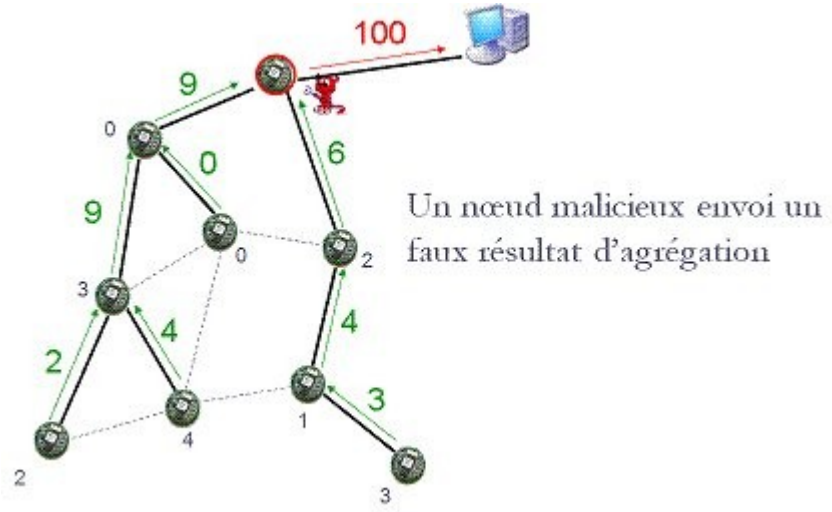


Image 62 : Un malicieux falsifie le résultat d'une agrégation

Toute la difficulté est comment permettre aux noeuds de relais d'avoir accès aux résultats intermédiaires pour calculer l'information utile par agrégation, et rendre

cette opération exempt de risque de falsification, suppression ou modification ?

Taxonomie des solutions d'agrégation sécurisée dans les RCSF

Il existe deux grandes catégories de solutions selon le mécanisme cryptographique utilisé :

- Solutions basées sur le cryptage de bout en bout : dans cette catégorie on utilise des mécanismes cryptographiques qui sécurisent l'information captée de bout en bout tout en permettant aux noeuds intermédiaires de réaliser les opérations d'agrégation. Dans cette catégorie, la vérification de l'information ne se fait généralement qu'au niveau du collecteur, ce qui engendre une forte contamination de la fausse information.
- Solutions basées sur le cryptage de proche en proche : dans ce cas, la véracité de la l'information est vérifiée de proche en proche et son rejet peut se faire à n'importe quel niveau de l'arbre couvrant le RCSF.

La figure suivante résume cette classification :

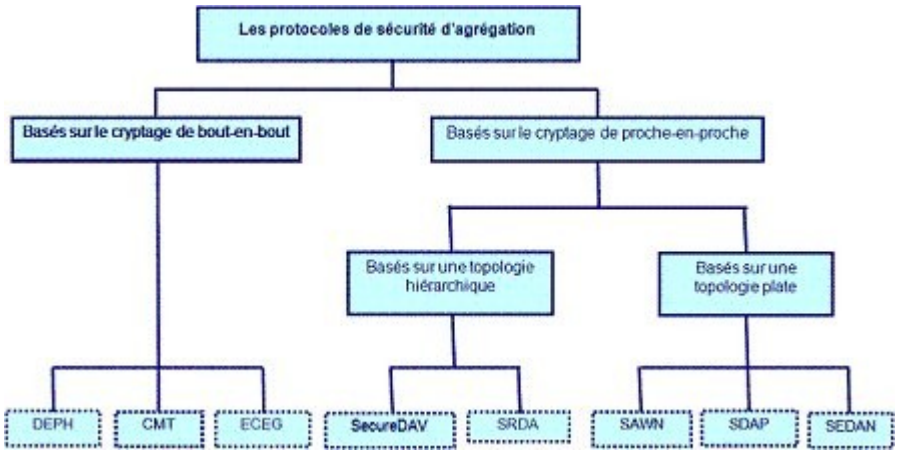


Image 63 : Classification des solutions d'agrégation sécurisée

2. SAWN (Secure Aggregation for Wireless Networks)

Aperçus

SAWN suppose que deux noeuds consécutifs ne peuvent pas être compromis simultanément.

Il se base sur la vérification à deux sauts : un noeud vérifie si l'agrégation des données de ses petits fils, réalisée par son fils, est correcte.

La vérification de l'agrégation se fait d'une manière différée dans le temps en utilisant le protocole µTESLA pour l'authentification des clés utilisées dans l'authentification des données et de leurs agrégations.

Dans ce qui suit nous supposons que les noeuds ont le moyen de vérifier l'authenticité des clés partagées entre les noeuds et la SB, lorsque cette dernière révèle les clés pour la vérification.

Chaque nœud feuille transmet sa lecture à son père. Les messages incluent la lecture des données du nœud, son id, ainsi qu'un MAC calculé grâce à la clé K_{Ai}. Cette dernière est partagée entre le nœud A et la station de base, mais n'est pas encore connue par les autres capteurs. Le nœud père stocke le message ainsi que son MAC jusqu'à la révélation de clé K_{Ai} par la station de base. A cet instant, il vérifiera le MAC et envoi une alarme en cas de différence. L'agrégation des lectures est exécutée dans chaque étape intermédiaire. Les nœuds attendent pendant un temps indiqué pour recevoir des messages de leurs fils et retransmettent ensuite

les messages et les MACs qu'ils reçoivent directement de leurs fils immédiats. Les nœuds agrègent les données qu'ils reçoivent de leurs petits-fils (via leurs fils) et transmettent le MAC de la valeur d'agrégation. Après l'arrivée de tous les messages à la station de base, cette dernière révèle les clés temporaires des nœuds. Une fois que la clé (K_{Ai}) est révélée, les nœuds passent à la clé temporaire suivante (K_{Ai+1}).

Exemple : Agrégation sécurisée dans SAWN

Considérons l'arbre d'agrégation illustré par la figure suivante. L'exemple illustre le i-ème tour où l'en utilise les clés K_{xi} pour authentifier les messages transmis:

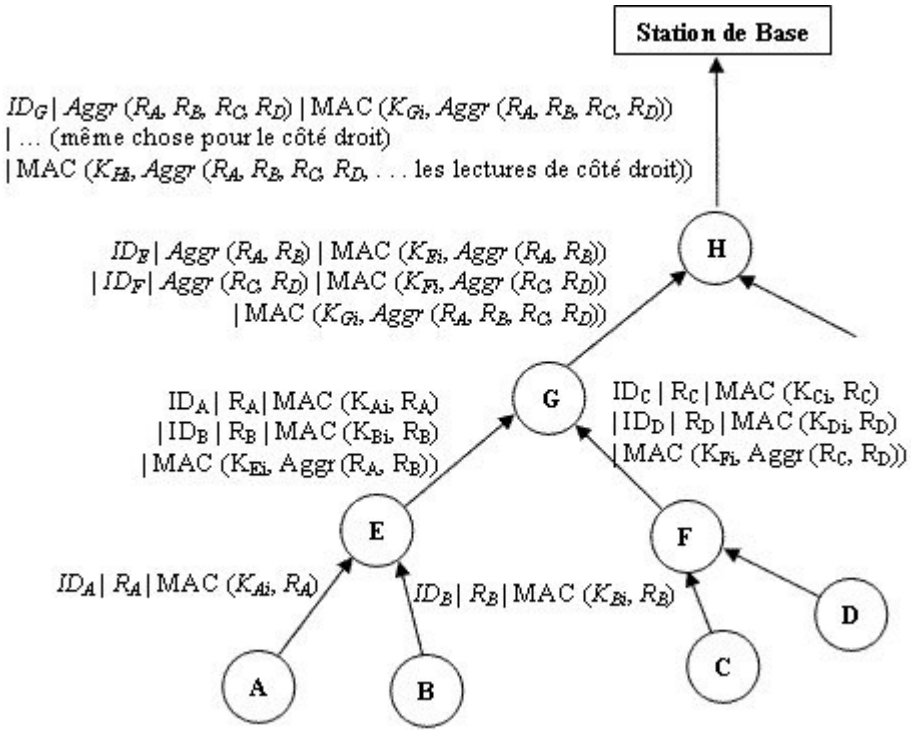


Image 64 : Exemple d'arbre d'agrégation sécurisé avec SAWN

Transmission de données

Les nœuds A, B, C et D envoient des données à la station de base via l'arbre d'agrégation construit avec un protocole de routage.

- Les nœuds feuilles envoient des données à leur père. Les messages incluent des MACs calculés avec la clé d'authentification courante:

  A==>E : $R_A \mid ID_A \mid MAC(K_{Ai},R_A)$

  Chaque clé est utilisée pour authentifier un seul message ce qui empêchera l'attaque de rejeu.
- Les nœuds intermédiaires reçoivent les messages de leurs fils. Le nœud père ne peut pas encore vérifier le MAC car la clé du fils ne sera révélée que pendant la phase de vérification. Pour le moment, le père stocke le message et le MAC. Le nœud intermédiaire attend les paquets des fils, et envoie ensuite un message à son père contenant les lectures des fils, leurs MACs, ainsi que le MAC calculé sur la valeur d'agrégation:

  E==>G : $R_A \mid ID_A \mid MAC(K_{Ai},R_A) \mid R_B \mid ID_B \mid MAC(K_{Bi},R_B) \mid MAC(K_{Ei},Aggr(R_A,R_B))$

  Il n'y a aucun besoin de transmettre la valeur d'agrégation calculée, puisque G peut calculer $Aggr(R_A,R_B)$ depuis les valeurs R_A et R_B. Ce n'est pas aussi nécessaire de transmettre l'ID_E à G, parce que G connaît la topologie du réseau donc peut déterminer le nœud qui envoie le message.

- Le nœud G reçoit les messages des noeuds E et F. Pour chacun d'eux, G calcule les valeurs de l'agrégation des lectures de ses petits-fils c'est-à-dire (A, B, C et D). Il transmet alors les valeurs agrégées de ses petits-fils, l'ID de ses fils et leurs valeurs de MAC. G calcule aussi et transmet le MAC de la valeur d'agrégation suivante :

  $Aggr(R_A,R_B,R_C,R_D)=Aggr(Aggr(R_A,R_B),Aggr(R_C,R_D))$.

  Puisque la fonction de l'agrégation est connue à tous les nœuds, le MAC calculé par E authentifiera la valeur calculée par G. Les lectures des capteurs et les valeurs de MAC reçues à partir de E et F sont stockées pour vérification postérieure.

  $G==>H : ID_E \mid Aggr(R_A,R_B) \mid MAC(K_{Ei}, Aggr(R_A,R_B)) \mid ID_F \mid Aggr(R_C,R_D) \mid MAC(K_{Fi},Aggr(R_C,R_D)) \mid MAC(K_{Gi},Aggr(R_A,R_B,R_C,R_D))$

  De la même façon, le nœud H reçoit des messages de G et d'une autre branche, et transmet à son tour le message agrégé à la station de base. Noter que la longueur du message n'augmente pas si le réseau était plus profond.
- La station de base reçoit le message de H. Elle peut calculer la valeur de l'agrégation finale, $Aggr(R_A, R_B, R_C, R_D, ...)$ en utilisant $Aggr(R_A,R_B,R_C,R_D)$ et les autres valeurs de ses nœuds fils.

Validation des données

Le but du protocole SAWN est d'authentifier toutes les lectures qui ont participé à la valeur d'agrégation, sans pour autant recevoir toutes ces lectures. Pour valider les données (lecture des noeuds et valeurs d'agrégation), la station de base révèle la clé courante K_{xi} qu'elle partage avec chaque noeud x du réseau, en émettant un seul message contenant toutes ces clés. Ce message de révélation des clés est authentifié par un MAC en utilisant une clé authentifiée grâce à µTESLA.

Si un nœud détecte un message erroné dans l'étape de validation de données, il envoie un message d'alarme. Une alarme est émise par un parent quand il détecte que le MAC d'agrégation d'un fils est contradictoire avec les données des petits-fils, ou bien quand les MAC des données eux-mêmes sont erronés.

3. Protocoles basés sur le cryptage de bout en bout

Fondamental : Concept

Les protococles de cette catégorie utilisent une clé partagée entre chaque noeud et le noeud collecteur pour garantir l'intégrité des données transmises dans le réseau. Comme les contenus des données sont cryptés, les noeuds utilisent un type de cryptographie particulier appelé "Privacy Homomorphism (PH) pour pouvoir exécuter l'agrégation

Définition : Privacy Homomorphism

Un algorithme est PH si et seulement si en ayant E(x) et E(y) on peut calculer E(x ¤y) sans décrypter x et y. Ainsi il vérifie la propriété suivante :

$E_{K1}(x_1)$ ¤ $E_{K2}(x_2)=E_{K1+K2}(x_1$ ¤ $x_2)$, où K_i sont les clés et x_i sont les données.

Remarque

Le point unique de vérification dans ce type de protocoles est le noeud collecteur. Ce dernier ayant toutes les clés utilisées pour crypter les données dans le réseau.

Méthode : Protocole CMT

Castelluccia, Myldetun et Tsudik ont proposé le protocole CMT basé sur l'hypothèse

que chaque noeud utilise une clé symétrique partagée entre ce noeud et le noeud collecteur. L'idée de ce protocole est que chaque noeud fait l'addition modulaire entre sa clé stockée et sa sonnée. Pendant la phase d'acheminement des données, l'agrégation se fait sur ces données qui sont déjà cryptées. L'algorithme suivant montre les différentes étapes de ce protocole :

L'algorithme de CMT

Paramètre :

Sélection d'un grand nombre entier M.

Cryptage :

Le message $m \in [0, M-1]$.

Aléatoirement générer une clé $k \in [0, M-1]$.

$C = (m+k) mod M$.

Décryptage :

$m = (c-k) mod M$.

Agrégation :

$c_{12} = (c_1 + c_2) mod M$.

Image 65 : Algorithme CMT

La taille du paquet dans ce protocole dépend de la taille de M, et une seule addition modulaire suffit pour l'agrégation et le cryptage. Ainsi, ce protocole ne consomme pas beaucoup d'énergie.

Méthode : Protocole Elliptic Curve ElGamel

Ce protocole utilise un algorithme cryptographique à courbe élliptique ElGamel (ECEG) qui est une approche asymétrique qui ne consomme pas autant d'énergie que les systèmes asymétriques classiques comme RSA. L'algorithme suivant illustre l'agrégation et la vérification dans ce protocole :

L'algorithme de ECEG

Paramètre :

Une clé privé x.

Une clé publique(G, H), G et H des points dans **ECEG**, $H = xG$.

Cryptage :

$C = [c_1, c_2] = [kG, kH + mG]$ = un point dans **ECEG**.

Décryptage :

$mG = (kH + mG) - x(kG)$.

Agrégation :

$C_{12} = C_1 + C_2 = [(c_{11} + c_{21}), (c_{12} + c_{22})]$.

Image 66 : Algorithme ECEG

Ateliers pratiques dans un environnement TinyOS

VII

Découverte de TinyOS 95
Développement d'un protocole de routage (many-to-one) pour RCSF 102

L'objectif de ces ateliers est de vous familiriser avec le développement d'applications pour réseaux de capteurs sans fils dans un environnement TinyOS. Par ailleurs, vous apprendrez à évaluer vos applications en les exécutant dans l'émulateur TOSSIM.

Le "NesC Reference Manual" (voir ci-bas) est un document succinct qui est très utile pour commencer à programmer avec NesC sous TinyOS

Vous trouverez également ci-bas un ensemble de tutoriaux pour découvrir les possibilités offertes par TinyOS et la syntaxe de NesC.

A. Découverte de TinyOS

Dans cet atalier vous allez découvrir TinyOS et le langage NesC pas à pas à travers une application. Dans un premier temps vous allez analyser le code de cette application. Puis vous exécuterez cette application et analyser le résultat obtenu. Enfin on vous demandera de modifier cette application en l'enrichissant avec de nouvelles fonctionnalités.

1. TinyOS : Concepts et définitions

TinyOS et NesC

Le système TinyOS, ses librairies, et ses applications sont écrites en nesC, un nouveau langage pour le développement d'applications orientées composants. Le langage nesC est principalement dédié aux systèmes embarqués comme les réseaux de capteurs. nesC a une syntaxe proche du langage C. mais supporte le modèle concurrent de TinyOS ainsi que des mécanismes pour la structuration, le nommage et l'assemblage de composants logiciels en des systèmes réseaux embarqués fiables. L'objectif principal est de permettre aux concepteurs d'applications de construire des composants qui peuvent être composés rapidement

en des systèmes complets, concurrent, tout en permettant une vérification profonde à la compilation.

TinyOS définie un nombre important de concepts qui sont exprimés dans nesC. Premièrement, les applications nesC sont construites à partir de composants ayant des interfaces bidirectionnelles bien définies. Deuxièmement, nesC définie un modèle de concurrence basé sur les notions de tâches, les "handlers" d'événements matériels, et la détection des conditions d'accès concurrent aux données au moment de la compilation.

Fondamental : Application nesC = ensemble de composants

Une application nesC consiste en un ou plusieurs composants assemblés pour former un exécutable.

Définition : Interface d'un composant

Un composant offre et utilise des interfaces. Ces interfaces sont l'unique point d'accès au composant et sont bidirectionnelles. Une interface définie un ensemble de fonctions appelées commandes (qui doivent êtes implémentées par le composant qui offre cette interface), et un autre ensemble de fonctions appelées événements (qui doivent êtres implémentées par le composant utilisant l'interface). Pour qu'un composant puisse appeler les commandes d'une interface il doit implémenter les événement définis dans l'interface. Un même composant pourrait offrir et utiliser plusieurs interfaces différentes ou plusieurs instances d'une même interface.

Définition : Modules et configurations

Il existe deux types de composants dans nesC: modules et configurations. Les modules définissent le code de l'application en implémentant une ou plusieurs interfaces. Les configurations sont utilisées pour assembler d'autres composants, en reliant les interfaces utilisées par des composants aux interfaces offertes par d'autres composants. ceci est appelé "wiring" (câblage). Toute application nesC est décrite par une configuration qui assemble les composants de l'application.

Remarque : L'extension de fichier ".nc"

nesC utilise l'extension de fichier ".nc" pour tous les fichiers sources -- interfaces, modules, et configurations.

Modèle de concurrence

TinyOS execute seulement un programme consistant d'un ensemble de composants requis par l'application. Il existe deux chemins d'exécution : les tâches, et les "handlers" d'événements matériels.

Définition : Tâche

Les tâches sont des fonctions dont l'exécution est différée. Une fois lancée, une tâche s'exécute jusqu'à sa fin et n'est pas et ne peut pas interrompre une autre tâche.

Définition : "Handler" d'événement matériel

Les "handlers" d'événements matériels sont exécutés en réponse à une interruption matérielle et s'exécutent aussi jusqu'à leurs fins, mais peuvent interrompre l'exécution d'une tâche ou d'un autre "handler".

Remarque

Les commandes et les événements qui s'exécutent dans un "handler" d'événement matériel doivent être déclarés avec le mot clé "async".

Complément

Comme les tâches et les "handlers" sont préemptibles par d'autres codes asynchrones, les programmes nesC son susceptibles à certaines conditions de concurrences d'accès aux données.

Pour éviter ces conditions d'accès concurrent, il faut soit accéder aux données partagées exclusivement à travers des tâches, ou en ayant tous les accès aux données partagées dans des instructions "atomic".

Le compilateur signale les accès potentiellement concurrents aux données partagées au programmeur. Il se peut que le compilateur signale des faux positifs Dans ce cas, une variable peut être déclarée avec le mot clé norace.

Attention : norace

Le mot clé norace doit être utilisé avec une extrême précaution.

2. Application Blink

Aperçus général

Le programme "Blink" se trouve à apps/Blink dans l'arbre de TinyOS. Cette application allume la LED rouge à une fréquence de 1HZ.

L'application Blink est composée de deux composants: un module, appelé "BlinkM.nc", et une configuration, appelée "Blink.nc". Blink.nc est utilisée pour relier le module BlinkM.nc à d'autres composants requis par l'application Blink.

Remarque

La raison pour laquelle on distingue les modules des configurations est de permettre au concepteurs d'un système de construire des applications rapidement et efficacement. Par exemple, un concepteur pourrait fournir uniquement une configuration qui relie un ensemble de modules qu'il ne développe pas lui même. De même, un autre développeur peut fournir une libraire d'un ensemle de modules qui peut être utilisés dans la construction d'autres applications.

Méthode : Convention de nommage

Parfois (et c'est le cas de BlinkM et Blink) vous aurez un module et une configuration qui vont ensemble. Quand c'est le cas, la convention utilisée dans l'arbre des codes sources de TinyOS est que Foo.nc représentes la configuration et FooM.nc représente le module correspondant.

a) La configuration Blink.nc

Le compilateur nesC, ncc, compile une application nesC à partir de sa configuration. Une application nesC typique est fournie avec un fichier Makefile standard qui permet de sélectionner la plateforme d'exécution de l'application (type de mote, pc, etc.) et invoque ncc, avec les options appropriées, sur la configuration de l'application. Examinons maintenant la configuration Blink.nc de notre application :

Blink.nc

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;
  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

Image 67 : Blink.nc Configuration

Dans les deux premières lignes, le mot clé "configuration" indique qu'il s'agit d'une configuration appelé "Blink".

Entre les deux accolades (vides ici) il est possible de spécifier des interfaces fournies et/ou utilisées par la configuration.

La configuration est implémentée entre les accolades suivants le mot clé "implementation".

La ligne "components" spécifies l'ensemble de composants référencés par cette configuration : Main, BlinkM, SingleTimer, LedsC. La suite de l'implémentation relie les interfaces utilisées par des composants aux interfaces fournies par les autres composants.

Fondamental : Le composant Main

Main est le composant exécuté en premier lieu dans une application TinyOS. Précisément, la commande Main.StdControl.init() est la première commande exécutée dans TinyOS suivie par la commande Main.StdControl.start(). Donc, une application TinyOS doit obligatoirement avoir un composant Main dans sa configuration. StdControl est une interface commune utilisée pour initialiser et lancer les composants TinyOS. examinons cette interface qui se trouve à tos/interfaces/StdControl.nc:

StdControl.nc

```
interface StdControl {
  command result_t init();
  command result_t start();
  command result_t stop();
}
```

Image 68 : L'interface StdControl.nc

Nous remarquons que StdControl définie trois commandes : init(), start(), et stop(). init() est appelé quand le composant est initialisé pour la première fois, et start() quand ison exécution est lancée la première fois. stop() est appelé quand le composant est arrêté, par exemple pour éteindre le composant qu'il contrôle.

Fondamental : Le "wiring"

nesC utilise des flèches pour déterminer la liaison entre interfaces. Il lire la flèche

vers la droite (->) "est liée à". Le côté gauche de la flèche relie une interface à son implémentation fournie par le côté droit de la flèche. En d'autres mots, le composant qui utilise une interface est du côté gauche de la flèche, et le composant qui fournie une implémentation de l'interface se trouve à droite de la flèche. La ligne

BlinkM.Timer -> SingleTimer.Timer;

est to utilisée pour relier l'interface Timer utilisée par BlinkM à l'interface Timer implémentée par SingleTimer. Le "wiring" peut aussi être implicite. Par exemple,

BlinkM.Leds -> LedsC;

est une abréviation pour

BlinkM.Leds -> LedsC.Leds;

Si aucun nom d'interface n'est donné au côté droit de la flèche, le compilateur nesC essaye par défaut de relier à la même interface du côté gauche de l'interface.

b) Le module BlinkM.nc

Considérons le module BlinkM.nc :

BlinkM.nc

```
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}
// Continued below...
```

Image 69 : Le module BlinkM.nc

La première partie du code signifie qu'il s'agit d'un module appelé BlinkM, et décalre les interfaces qu'il utilise et fournie. Le module BlinkM fournie l'interface StdControl. ceci veut dire que BlinkM implémente l'interface StdControl. Comme expliqué ci-dessus, ceci est nécessaire pour initialiser et lancer l'exécution du composant Blink. Le module BlinkM utilise deux interfaces : Leds et Timer. Ceci veut dire que BlinkM peut faire appel à toutes les commandes déclarées dans ces deux interfaces et doit aussi implémenter tous les événements déclarés dans ces interfaces qu'il utilise. L'interface Leds définie plusieurs commandes comme redOn(),redOff(), etc., qui allument et éteignent les différents LEDs (rouge, vert, jaune) du capteur.

Cependnat, Leds est juste une interface : l'implémentation est spécifiée dans la configuration Blink.nc.

Remarque : Timer

Considérons l'interface Timer.nc

Timer.nc

```
interface Timer {
  command result_t start(char type, uint32_t interval);
  command result_t stop();
  event result_t fired();
}
```

Image 70 : L'interface Timer.nc

On remarque que l'interface Timer définie les commandes start() et stop(), l'événement fired(). La commande start() est utilisée pour spécifier le type du timer et l'intervalle de temps au bout duquel le Timer expire. L'unité de l'argument interval est la milliseconde. Les types valides sont TIMER_REPEAT (le Timer se déclenche après chaque periode de "interval" de temps jusqu'à sont arrêt avec stop()), et TIMER_ONE_SHOT (le Timer se déclenche une seule fois après l'expiration du délai "interval"). Comment est ce qu'une application apprend l'expiration d'un Timer ? La réponse est lorsqu'elle reçois l'événement "expired()". L'interface Timer fournie un événement:

event result_t fired();

un événement est une fonction que l'implémentation de l'interface signale quand un certain événement apparaît. dans notre cas, l'événement "fired()" est signalé quand l'intervalle de temps spécifié en argument est passé. Ceci est un exemple d'une interface bidirectionnelle : une interface non seulement fournie des commandes qui peuvent être appelé par les composants utilisant l'interface, mais aussi signale des événements qui font appel à des handlers dans le composant qui utilisent l'interface. Un module qui utilise une interface doit implémenter les événement signalés par cette interface.

Considérons maintenant la suite du module BlinkL.nc :

BlinkM.nc, continued

```
implementation {
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }
  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000) ;
  }
  command result_t StdControl.stop() {
    return call Timer.stop();
  }
  event result_t Timer.fired()
  {
    call Leds.redToggle();
    return SUCCESS;
  }
}
```

Image 71 : Suite module BlinkM.nc

Comme on peut le voir facilement, le module BlinkM implémente les commandes StdControl.init(), StdControl.start(), et StdControl.stop(), puisqu'il fournie l'interface StdControl. Il implémente également l'événement Timer.fired(), qui est nécessaire puisque BlinkM doit implémenter les événement signalés par une interface qu'il utilise. La commande init() initialise simplement le composant Leds en appelant Leds.init(). La commande start() invoque Timer.start() pour créer le Timer répétitif (REPEAT TIMER) qui expire à chaque 1000 ms. stop() arrête le Timer. A chaque fois que l'événement Timer.fired() est signalé, la commande Leds.redToggle() permet d'allumer la diode rouge.

3. Compilation et exécution de Blink

Nous allons émuler l'exécution de l'application Blink avec TOSSIM.

Pour cela on compile Blink en tapant la commande suivante :

make pc

Comme nous sommes intéressé par afficher uniquement l'état des diodes on doit affecter la valeur led à la variable globale DBG en tapant la commande :

export DBG=led

Enfin pour lancer l'exécution de Blink sur un seul capteur, on tape la commande :

build/pc/main.exe 1

4. Exercice

Modifiez Blink de telle sorte que le capteur affiche, en utilisant les diodes, les trois bits de poids faible d'un compteur.

Recompiler et relancer l'exécution de Blink sur un capteur en utilisant TOSSIM. utilisez export DBG=led pour afficher l'état des diodes seulement.

B. Développement d'un protocole de routage (many-to-one) pour RCSF

Vous apprendrez progressivement comment développer et mettre au point un protocole de routage (many-to-one). Les capteurs pourront ainsi capter des informations environnementales puis les acheminer vers le la station de base pour analyse et prise de décision.

L'atelier esr composé de plusieurs étape. Après chaque étape vous devriez faire une analyse du résultat obtenu puis définir les mécanismes à introduire dans l'étape suivante afin de palier les insuffisances notées dans l'étape en cours.

1. Protocole de routage "many-to-one" simple

Description du protocole

Dans cet atelier, vous allez développer votre premier protocole de routage pour RCSF. Vous allez aussi découvrir comment créer une application TinyOS à partir de composants. Puis exécuter l'application dans TOSSIM, et analyser les résultats obtenus.

L'objectf de ce protocole est de créer un arbre qui couvre le maximum de capteurs, dont la racine se situe au puis (sink node).

Nous appellerons ce protocole HTC pour (Hierarchical Tree Construction).

Le nœud puis lance la procédure de création de l'arbre en envoyant par "broadcast" une requête de création.

Quand un noeud reçois cette requête il considère l'émetteur de la requête comme son père et le mémorise dans une variable locale ParentID.

Donc le message de requête va contenir au moins l'identifiant de l'émetteur de la requête.

Voici le fichier le fichier HTC.h définissant la structure du message de requête et son identifiant :

Syntaxe : HTC.h (Version 1)

```
#ifndef HTC_MESSAGES_H
#define HTC_MESSAGES_H
#define SINK_NODE 0
typedef struct TreeReqMsg {
uint16_t src; // This is the node ID
}TreeReqMsg;
enum { AM_TREQMSG = 50 // This is the packet ID of control messages (Tree Request Messages) that are sent to build the tree };
#endif
```

Composants de l'application

Comme expliqué ci-dessus, cette première version du protocole aura besoin d'envoyer et recevoir des messages. Pour cela, nous utiliserons le composant GenericComm de TinyOS qui implémente les interfaces SendMsg et ReceiveMsg permettant d'envoyer et recevoir des messages respectivement.

Toute fois, afin d'implémenter le protocole lui même (initier la procédure de

création de l'arbre par le puis, noter le noeud père, etc.) nous aurons besoin de développer notre propre composant qu'on appellera TreeBuilderM.nc. Ce composant utilisera les interfaces SendMsg et ReceiveMsg de GenericComm pour l'envoie et la réception des messages.

TreeBuilderM.nc implémente l'interface StdControl obligatoirement (imposé par TinyOS) pour son initialisation, son lancement et son arrêt.

Analysez le code suivant qui implémente le module TreeBuilder et répondez aux questions suivantes :

1/ Qui lance la construction de l'arbre ?

2/ A qui est envoyée la requête de construction de l'arbre ? qui va la recevoir ?

3/ Est ce que cette version du protocole permettra de réaliser une couverture maximale des noeuds du réseau ? Pourquoi ?

Syntaxe : TreeBuilderM.nc

```
module TreeBuilderM {
provides { interface StdControl; }
uses { interface ReceiveMsg as ReceiveTreeReqMsg; interface SendMsg as SendTreeReqMsg; }
}
implementation {
//Routing
uint16_t parentId;
//Communication
TOS_Msg buffer; // Each node has a buffer where it stores received messages.
//Functions and tasks
task void sendInitTreeReq(); // This task is launched by the SINK_NODE to initialize tree construction task
//Implementation of StdControl Interface
command result_t StdControl.init() {
parentId = TOS_BCAST_ADDR; //Unknown parent
return SUCCESS;
}
command result_t StdControl.start() {
if (TOS_LOCAL_ADDRESS == SINK_NODE) {
post sendInitTreeReq(); //Starting the creation of the tree
}
return SUCCESS;
}
task void sendInitTreeReq() {
struct TreeReqMsg* ptr;
ptr = (struct TreeReqMsg*)(buffer.data);
ptr->src = TOS_LOCAL_ADDRESS; // I will be a parent
call SendTreeReqMsg.send(TOS_BCAST_ADDR,sizeof(struct TreeReqMsg),&buffer);
}
command result_t StdControl.stop() {
return SUCCESS;
}
event result_t SendTreeReqMsg.sendDone(TOS_MsgPtr pmsg, result_t success ) {
```

```
return SUCCESS;
}
event TOS_MsgPtr ReceiveTreeReqMsg.receive(TOS_MsgPtr pmsg) {
struct TreeReqMsg * ptr;
ptr = (struct TreeReqMsg *)(pmsg->data);
parentId = ptr->src; // I received a tree construction request. I note the Id of my parent
dbg(DBG_USR2,"__ROUTE__:parent=%d\n",parentId);
return pmsg;
}
}
```

Construction de l'application

Pour construire l'application HTC.nc il faut maintenant relier les différents composants qui constituent cette application.

Analysez le code suivant :

Syntaxe : HTC.nc

```
includes HTC;
configuration HTC { }
implementation {
components Main, GenericComm, TreeBuilderM;
Main.StdControl -> TreeBuilderM;
Main.StdControl -> GenericComm.Control;
//Tree Builder
TreeBuilderM.ReceiveTreeReqMsg -> GenericComm.ReceiveMsg[AM_TREQMSG];
TreeBuilderM.SendTreeReqMsg -> GenericComm.SendMsg[AM_TREQMSG];
}
```

Compilation et exécution

Avant de compiler et exécuter l'application HTC, nous devons d'abord définir la topologie du réseau. Pour simplifier les choses, nous choisirons d'utiliser une grille de 7x7 noeuds.

Le fichier suivant définie une telle topologie :

Pour compiler l'application pour être émulé par TOSSIM, tapez

```
make pc
```

Pour lancer l'exécution, tapez :

```
export DBG=usr2
./build/pc/main.exe -t=10 -r=lossy -rf=./grid7x7 49
```

Méthode : Analyse et suite ...

Analysez le résultat obtenu et répondre aux questions suivantes :

1/ Combien de noeuds sont couverts par l'arbre construit ?

2/ Qu'elle est la particularité des noeuds couverts par l'arbre ?

3/ Que proposez vous pour améliorer la couverture du protocole ?

2. Construction de l'arbre de routage par relais de la requête de construction

Lessons de la version précédente

Vous avez remarqué dans la version précédente que l'arbre construit est réduit à un seul saut.

Pour que la requête de construction parvienne aux autres noeuds, il est nécessaire de la faire relayer à chaque fois qu'elle est reçus par un noeud.

Pour cela, reprenez la version précédente de TreeBuilderM et rajoutez une tâche :

```
task void relayTreeReq() {
struct TreeReqMsg* ptr;
ptr = (struct TreeReqMsg*)(buffer.data);
ptr->src = TOS_LOCAL_ADDRESS;
dbg(DBG_USR2,"__ROUTE__:mainParent=%d\n",parentId);
call SendTreeReqMsg.send(TOS_BCAST_ADDR,sizeof(struct TreeReqMsg),&buffer);
}
```

Analysez que fait cette tâche.

Modifier TreeBuilderM pour que cette tâche soit appelée par chaque noeud recevant une requête de construction de l'arbre de routage.

Méthode : Exécution et analyse

Relancer la compilation et l'exécution de cette nouvelle version.

Analyser le résultat obtenu.

Est ce que tous les noeuds couverts par l'arbre ont un chemin vers la racine (noeuds puis 0) ?

De quoi souffre cette version du protocole ?

Proposez une solution à ce problème.

3. Loop free routing protocol

Lessons de la version précédente

Vous avez remarqué dans la version précédente que certaines routes formaient des boucles. Un paquet qui sera injecté dans une boucle y restera jusqu'à l'infini.

ce problème est dû au fait qu'un noeud qui reçoit la requête de construction de l'arbre ne change pas son état. Ainsi, comme la requête est diffusée par inondation, ce noeud pourra recevoir de nouveau une requête initiée par lui même avec le mécanisme de relais.

Conseil : Elimination des boucles

Une solution simple à ce problème serait de marquer les noeuds visiter pour qu'il ignorent les nouvelles requêtes.

Reprenez l'ancienne version de HTC et mettez à jours TreeBuilderM.nc pour éliminer les boucles.

Relancer la compilation et l'exécution de nouveau.

Le problème persiste toujours ?

Quel est le taux de couverture des noeuds du réseau ?

Pourquoi la requête de construction de l'arbre ne parvient elle pas à tous les noeuds bien que le réseaux est connexe et la requête est diffusée par broadcast ?

Proposez une solution à ce problème.

4. Evitement des collisions dans la propagation de la requête de construction

Lessons de la version précédente

Dans la version précédente, la couverture du réseau est très faible.

Ceci signifie que la requête ne parvient pas à tous les noeuds du réseau.

Comme les noeuds propagent la requête de construction simultanément, des collisions sont provoquées et donc des requêtes sont perdues durant la propagation par "broadcast".

Pour éliminer ce phénomène, une solution consisterait à introduire un délai aléatoire avant le relayage de la requête par les noeuds qui reçoivent cette requête.

Conseil : Evitement des collisions par attente aléatoire

Pour mettre en oeuvre cette solution nous aurons besoins de deux nouveaux éléments dans notre application HTC :

1/ Attendre un délai avant de relayer la requête

2/ Ce délai doit être aléatoire

Pour tirer profit de la programmation par composant utilisée dans TinyOS nous allons utiliser deux composants de TinyOS qui permettent d'implémenter ces deux comportement :

1/ TimerC : ce composant implémente une interface qui permet d'attendre et lancer un événement après le délai d'attente

2/ RandomLSFR : permet de générer des nombres aléatoires.

Etudiez ces deux composants en prenant connaissance de leurs fonctionnalités dans la libraire de TinyOS

Rajouter ces composants dans la configuration de l'application HTC.nc

Rajouter les liaisons nécessaires entre ces composants et notre composant TreeBuilderM

Rajouter dans le module TreeBuilderM le fait que le composant nécessite les interfaces Timer et Random

Mettre à jours l'implémentation de TreeBuilderM de telle sorte que chaque noeud attende un délai aléatoire à la réception de la requête de construction avant de faire appel à la tâche relayTreeReq pour relayer la requête à d'autres noeuds.

Relancer la compilation et l'exécution de HTC

Quel est le nouveau taux de couverture du réseau par l'arbre ainsi construit

5. Tolérance aux pannes par reconstruction de l'arbre de routage

Des noeuds capteurs peuvent tomber en panne, à cause de l'épuisement de leur batterie, ou à cause d'une destruction physique. De nouveaux noeuds capteurs peuvent être déployés pour renforcer la densité du réseau. Pour tenir compte de ces phénomène et éliminer les noeuds en pannes de l'arbre et inclure les nouveaux noeuds dans l'arbre, il est nécessaires de relancer périodiquement la construction de l'arbre.

Attention

Pour éliminer le problème des boucles, on avait utilisé un booléen pour savoir si le noeud a été déjà visité ou pas. Si on veut relancer la construction périodiquement il faut introduire un mécanisme pour savoir s'il s'agit d'une ancienne requête ou d'une nouvelle requête de reconstruction qu'il ne faudrait pas ignorer.

Une solution simple serait de remplacer ce booléen par un numéro séquentiel de la requête de reconstruction de l'arbre.

Si le numéro de la requête reçus par un noeud est inférieur ou égal au numéro séquentiel actuel (qui doit être sauvegardé au niveau du noeud), alors ignrer cette requête pour éviter les boucles. Sinon, traiter la requête comme une nouvelle requête de reconstruction de l'arbre.

Conseil : Reconstruction périodique de l'arbre

Afin de relancer la construction de l'arbre périodiquement, le noeuds puis (sink) doit relancer la tâche sendInitTreeReq() périodiquement après un délai fixe.

Pour cela, nous aurons besoin d'utiliser un deuxième Timer.

Rajouter dans TreeBuilderM.nc le fait que TreeBuilderM utilise une deuxième interface Timer avec un autre alias (par exemple RoundTimer)

Rajouter dans HTC.nc la liaison entre TreeBuilderM et cette deuxième instance de TimerC

Mettre à jours TreeBuilderM pour relancer périodiquement la construction de l'arbre.

Recompiler puis relancer l'exécution de HTC.

Bibliographie

[1] akyildiz

Akyildiz I. F. Su W., Sankarasubramaniam Y. et Cayirci E. "Wireless sensor networks: a survey", Computer Networks. Vol. 38(4). - pp. 393-422, 2002

[2] boukerche1

Boukerche A. Werner Nelem Pazzia R., Borges Araujo R. "Fault-tolerant wireless sensor network routing protocols for the supervision of context-aware physical environments"; Journal of Parallel and Distributed Computing, Avr 2006. - 4 : Vol. 66. - pp. 586-599.

[3] boukerche2

Boukerche A., Chatzigiannakisb I., Nikoletseas S. "A new energy efficient and fault-tolerant protocol for data propagation in smart dust networks using varying transmission range, 2006. - Vol. 29. - pp. 477-489.

[4] castellucia

Claude Castelluccia, Einar Mykletun, and Gene Tsudik. "Efficient aggregation of encrypted data in wireless sensor networks". In MobiQuitous, pages 109-117. 2005.

[5] cougar

Yong Yao and J. E. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. Sigmod Record, Volume 31, Number 3, September 2002.

[6] DD

C. Intanagonwiwat and R. Govindan and D. Estrin, "Directed Diffusion : a Scalable and Robust Communication Paradigm for Sensor Networks", ACM MobiCom 2000, Boston, MA (2000).

[7] EEC-WSN

D. J. Malan, M. Welsh, and M. D. Smith, "A Public-Key Infrastructure for Key Distribution in TinyOS based on Elliptic Curve Cryptography", Proc. 1st IEEE Int'l.Conf. Sensor and Ad Hoc Communications and Networks, Santa Clara, CA, Oct. 2004.

[8] eschenauer

L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks", In Proceedings of the 9th ACM conference on Computer and communications security, November 2002.

[9] insens

J. Deng, R. Han, and S. Mishra, "Insens : intrusion-Tolerant Routing for Wireless Sensor Networks", Computer Communications 29 (2006), no. 2, 216–230.

[10] kCDS

Daia F., Wub J. "On constructing k-connected k-dominating set in wireless ad hoc and sensor networks"; J. Parallel Distrib. Comput., 2006. - Vol. 66. - pp. 947 – 958.

[11] leap

S. Zhu, S. Setia, and S. Jajodia, "Leap : efficient security mechanisms for large-scale distributed sensor networks", CCS 03 : Proceedings of the 10th ACM conference on Computer and communications security (New York, NY, USA), ACM Press, 2003, pp. 62–72.

[12] mcfa

F. Ye et al., "A Scalable Solution to Minimum Cost Forwarding in Large Sensor Networks", 10th Int. Conf. Comp. Commun. and Networks (2001), 304–309.

[13] nesc

David Gay, Philip Levis, david Culler, Eric Brewer ; "nesC 1.1 Language Reference Manual".

[14] pegasis

S. Lindsey and C. Raghavendra, "PEGASIS : Power-efficient gathering in sensor information systems," in Proceedings of IEEE Aerospace Conference, vol. 3, March 2002, pp. 1125-1130.

[15] ph

Josep Domingo-Ferrer. "A provably secure additive and multiplicative privacy homomorphism". In ISC '02 : Proceedings of the 5th International Conference on Information Security, 2002.

[16] q-composite

H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks", In IEEE Symposium on Security and Privacy, Berkeley, California, May 11-14 2003, pp. 197{213.

[17] route

Kemal Akkaya , Mohamed Younis "A survey on routing protocols for wireless sensor networks" Ad Hoc Networks 3,2005.

[18] RR

D. Braginsky and D. Estrin, "Rumor Routing Algorithm for Sensor Networks", 1st Workshop. Sensor Networks and Apps., Atlanta, GA (2002).

[19] sawn

L.Hu and D. Evans, "Secure aggregation for wireless networks",Workshop on Security and Assurance in Ad Hoc Networks, January 2003.

[20] secrout

J. Yin and S. Madria, "Secrout : A secure routing protocol for sensor networks", AINA 06 : Proceedings of the 20th International Conference on Advanced Information Networking and Applications (Washington, DC, USA), vol. 1, 2006, pp. 393–398.

[21] secroute

Chris Karlof, David Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures"; Ad Hoc Networks Vol.1 (2003) pp. 293–315.

[22] secureDAV

Mahimkar A et Rappaport, T.S, SecureDAV : A Secure Data Aggregation and Verication Protocol for Sensor Networks, 2004.

[23] sedan

M.Bagaa, N. Lasla, A. Ouadjaout and Y.Challal ; "SEDAN : Secure and Efficient protocol for Data Aggregation in wireless sensor Networks", 32nd IEEE Conference on Local Computer Networks (LCN 2007) pp. 1053-1060, Worksohop on Netwok Security.

[24] spin

A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "Spins : Security Protocols for Sensor Networks", Wirel. Netw. 8 (2002), no. 5, 521–534.

[25] surveyCST

Y.wang, G.Attebury and B.Ramamurty, "A survey of security issues in wireless sensor networks", in IEEE Communication Survey Tutorials, 2006.

[26] teen

A. Manjeshwar and D. P. Agrawal, "Teen : Arouting protocol for enhanced efficiency in wireless sensor networks", the 15th International Parallel & Distributed Processing Symposium (IPDPS-01), 2001, p. 189.

[27] tinysec

C. Karlof, N. Sastry, and D. Wagner, "Tinysec : A link layer security architecture for wireless sensor networks," in Second ACM Conference on Embedded Networked Sensor Systems (SensSys 2004), November 2004.

[28] tossim

Philip Levis and Nelson Lee, "TOSSIM: A Simulator for TinyOS Networks"

[29] crossbow

http://www.xbow.com/

[30] tinyos

http://www.tinyos.net

[31] zigbee

http://www.zigbee.org