REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mouloud MAMMERI de Tizi-Ouzou
Faculté du Génie Electrique et d'Informatique
Département d'Informatique

Mémoire Master II En Informatique
Spécialité : Systèmes Informatiques

Les Impacts Des Algorithmes De Compression De Données Sur la Consommation Energétique Dans Les Réseaux De Capteurs Sans Fil

Réalisé par :
Mlle ***MAHIOUT Lilia***

Proposé par :
Mr ***DAOUI M.***

Promotion 2010/2011

Remerciements

Je tiens à remercier Mr. Daoui, pour son encadrement, pour la confiance totale qu'il m'a témoignée, et surtout pour sa disponibilité et sa grande générosité durant toutes ces années.

Je remercie les membres du jury, Mme BELKADI, Mr HAMEG et Mr TAHANOUT, d'avoir accepté de juger mon travail.

Je remercie également toute ma promotion, toutes les personnes qui m'ont marquée.

Ma dette de reconnaissance est très lourde envers ma chère sœur Maya MAHIOUT, pour son soutient, son investissement à m'offrir tout ce dont j'ai besoin afin de réussir.

Mes remerciements les plus sincères, et ma fierté la plus grande vont envers mon cher frère Said MAHIOUT, pour son soutien constant, sa grande confiance en moi.

Je livre ma profonde gratitude à mes chers parents.

Je remercie mes chers amis « MAIZ Zina », « KERRICHE Rabah », et « MOKRANI Lhadi », d'avoir cru en moi, et de m'avoir toujours soutenu.

A mes chers grands parents

A tous les enfants malades, à ceux qui leurs quotidiens ce limitent à la lutte contre la maladie….à Hanane

A tout ceux qui n'ont jamais connu la tendresse d'un parent….à tous les orphelins

Résumé

Les récentes avancées dans les divers domaines liés à la micro-électronique, à l'informatique et aux réseaux sans fil ont donné naissance à de nouvelles thématiques de recherche. Les réseaux de capteurs issus de ces nouveaux progrès technologiques constituent un axe de recherche très fertile. En effet, la capacité réduite des nœuds en terme de calcul, de mémoire et d'énergie génère de nombreuses problématiques intéressantes.

Le but de ce travail est de faire une étude sur les impacts de la compression de données sur la consommation énergétique dans les réseaux de capteurs.

L'économie d'énergie étant un fil conducteur de notre travail et le module de transmission la principale source de consommation d'énergie, nous implémentons et simulons le comportement de deux algorithmes de compression de données RLE et K-RLE, sous le système d'exploitation TinyOS-1.x, et le langage de programmation NesC. Cela dans le but d'optimiser, l'économie de l'énergie des nœuds capteurs, et l'utilisation du canal de transmission.

Mots-clés : réseaux de capteurs, économie de l'énergie, compression de données

Abstract

Recent advances in various areas related to micro-electronics, computer science and wireless networks have resulted in the development of new research topics. Sensor networks are one of them. The particularity of this new research direction is the reduced performances of nodes in terms of computation, memory and energy.

The objective of this work is to illustrate the compression data impacts to power consumption in wireless sensor networks.

Less power consumption is the wire driver of our work. And the transceiver (the must power consumption source of motes). We implement and simulate the behavior of data compression algorithms RLE and K-RLE,

Using TinyOS-1.x (operating system), and the NesC programming language. In order to optimize the power consumption of mote (sensor) and the using of transmission canal.

Keywords: Sensor Networks, energy saving, data compression.

Tables des matières

Chapitre III: Techniques de compression de données dans les réseaux de capteurs

Chapitre IV: Plateforme de Simulation et Simulation

Partie I : Plateforme de simulation

Tables de figures

Tables

Acronymes

WLAN Wireless Local Area Network

WPAN Wireless Personal Area Networks

ISM Industrial, Scientific, Medical

SCO Synchronous Connection-Oriented

ACL Asynchronous Connection-Less

RDC Réseaux de capteurs

CISC Complex Instruction Set Computer

RISC Reduced Instruction-Set Computer

FSK Frequency-Shift Keying

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance.

TDMA Time Division Multiple Access

MAC Medium Access Control

DARPA Defense Advanced Research Agency

MANET Mobile Ad Hoc Networks

IETF Internet Engineering Task Force

GPS Global Positioning System

RSSI Received Signal Strength Indication

TDOA Time Difference Of Arrival

AOA Angle Of Arrival

CAP Contention Access Period

CFP Contention Free Period

Introduction générale

Les avancées technologiques et techniques opérées dans le domaine des réseaux sans fil, de la microfabrication et de l'intégration des microprocesseurs ont fait naître une nouvelle génération de réseaux de capteurs à grande échelle adaptés à une gamme d'applications très variée. Imaginons un ensemble de petits appareils électroniques, autonomes, équipés de capteurs et capables de communiquer entre eux sans fil. Ensemble, ils forment un réseau de capteurs sans fil capable de superviser une région ou un phénomène d'intérêt, de fournir des informations utiles par la combinaison des mesures prises par les différents capteurs et de les communiquées ensuite via le support sans fil.

Cette nouvelle technologie promet de révolutionner notre façon de vivre, de travailler et d'interagir avec l'environnement physique qui nous entoure. Des capteurs communicants sans fil et dotés de capacités de calcul facilitent une série d'applications irréalisables ou trop chères il y a quelques années. Aujourd'hui, des capteurs minuscules et bon marché peuvent être littéralement éparpillés sur des routes, des structures, des murs ou des machines, créant ainsi une sorte de « seconde peau numérique » capable de détecter une variété de phénomènes physiques. De nombreux domaines d'application sont alors envisagés tels que la détection et la surveillance des désastres, le contrôle de l'environnement et la cartographie de la biodivesité, le bâtiment intelligent, l'agriculture de précision, la surveillance et la maintenance préventive des machines, la médecine et la santé, la logistique et les transports intelligents.

Les réseaux de capteurs sans fil sont souvent caractérisés par un déploiement dense et à grande échelle dans des environnements limités en terme de ressources. Les limites imposées sont la limitation des capacités de traitement, de stockage et surtout d'énergie car ils sont généralement alimentés par des piles. Recharger les batteries dans un réseau de capteurs est parfois impossible en raison de l'emplacement des nœuds, mais le plus souvent pour la simple raison que cette opération est pratiquement ou économiquement infaisable. Il est donc largement reconnu que la limitation énergétique est une question incontournable dans la conception des réseaux de capteurs sans fil en raison des contraintes strictes qu'elle impose sur l'exploitation du réseau. En fait, la consommation d'énergie des capteurs joue un rôle important dans la durée de vie du réseau qui est devenue le critère de performance prédominant dans ce domaine. Si nous voulons que le réseau fonctionne de manière satisfaisante aussi longtemps que possible, ces contraintes d'énergie nous obligent à faire des compromis entre différentes activités aussi bien au niveau du nœud qu'au niveau du réseau.

Plusieurs travaux de recherche sont apparus avec un objectif : optimiser la consommation énergétique des nœuds à travers l'utilisation de techniques de conservation innovantes afin d'améliorer les performances du réseau, notamment la maximisation de sa durée de vie. De façon générale, économiser l'énergie revient finalement à trouver le meilleur compromis entre les différentes activités consommatrices en énergie.

La principale source de consommation étant la transmission de données, nous allons nous focaliser sur une stratégie permettant d'optimiser l'exploitation du canal de communication à travers la compression de données. Cette stratégie permettant de réduire le volume de données à un double avantages, d’une part elle réduit l’utilisation de l’unité de transmission très

gourmande en énergie, et d'une autre part, elle permet de conserver la bande passante réduisant ainsi les collisions. Il faut noter que ce choix d'étude fait que les applications prises en compte sont de type observation c'est-à-dire centrées sur les données et non sur les évènements. Nous ferons l'étude et l'implémentation de deux algorithmes de compression de données RLE et K-RLE dans le cadre de cette thèse.

Cette thèse est structurée de la manière suivante :

Le premier chapitre présente le domaine des réseaux de capteurs sans fil. Il définit l'élément principal qui constitue un réseau de capteurs : le capteur et les différentes unités qui le composent. Ce chapitre Définie aussi des différentes notions et concepts gravitant autour de cette thématique, ensuite les formes de dissipations de l'énergie dans les réseaux de capteurs serons exposés.

Nous poursuivons notre état de l'art dans le deuxième chapitre en définissant la durée de vie des réseaux de capteurs et en synthétisant les différentes techniques et mécanismes de conservation d'énergie que nous avons recensés dans la littérature.

Le troisième chapitre présente de nouvelles techniques de compression de données. Cette technique a pour principal objectif d'économiser l'énergie mais se révèle être aussi, dans certains cas, une aubaine pour les applications contraintes par le temps. La littérature sur les algorithmes de compression pour réseau de capteurs reste encore pauvre, mais le travail qui à été fait dans [5] s'est traduit par la proposition de nouvelles techniques de compression de données qui sont plus efficaces que l'algorithme de référence dans le domaine. Ces propositions sont efficaces en terme de compression, d'économie d'énergie mais aussi garantissent un temps de transfert plus rapide dans certains cas.

Le quatrième et dernier chapitre permettra d'exposer les résultats de notre implémentation et de tests de simulation des algorithmes de compressions de données RLE et K-RLE, précédé par une présentation des outils nécessaires pour la réalisation à savoir le système d'exploitation TinyOS, le langage de programmation NesC, et les simulateurs TOSSIM et PowerTOSSIM.

Enfin, ce manuscrit sera clôturé par une conclusion et des perspectives.

Chapitre I: Introduction aux Réseaux de Capteurs sans fil

Sommaire

1. Introduction :

Le monde des réseaux et télécommunication est un univers en perpétuelle évolution qui se manifeste par l'enchaînement de dispositifs électroniques sur le marché. En effet, les progrès technologiques actuels ont favorisé l'accroissement des performances de processus, des mémoires et des technologies sans fils, entrainent ainsi le développement de nouveaux systèmes informatiques et électroniques. Ce développement est accéléré par l'intérêt pour des dispositifs miniaturisés, performant et autonomes par le grand public. On assiste alors à l'engouement par de grosses entreprises informatiques pour la course à la miniaturisation de dispositifs électroniques embarqués tel que le témoigne la mise sur le marché de l'IPhone d'Apple. Nous sommes passés d'un ordinateur personnel ayant une architecture 8bits à un téléphone multifonctions doté d'une architecture 32 bits. Ainsi, les dispositifs embarqués d'aujourd'hui sont plus puissants que les ordinateurs d'hier.

Avec ces différents progrès remarquables et la course à la miniaturisation nous assistons à l'émergence des systèmes embarqués combinant divers composants électroniques tels que des capteurs, des modules de communications sans fil, voire des modules de géo-localisation capables d'observer leurs environnements se positionner mais aussi transmettre des informations.

Par contre, la contrainte induite par l'autonomie des dispositifs embarqués incite à utiliser des plateformes ayant des performances de plus en plus réduites pour leur consommation malgré les puissances de calcul et tout cela dans un système dit collaboratif qui ne se limite plus à une seule ressource restreinte mais utilise les capacités de plusieurs autres. Nous assistons alors au développement de nouveaux types de réseaux distribués de microcomposants peu coûteux, capable de collecter des informations, les traiter puis les transmettre à l'aide d'une communication sans fil à une station de base : *les Réseaux de Capteurs sans Fil*

Les réseaux de capteurs sans fil sont l'une des technologies visant à résoudre les problèmes de cette nouvelle ère de l'informatique embarquée et omniprésente. Nous allons retracer dans le présent chapitre le fonctionnement des réseaux de capteurs en nous focalisant sur les mécanismes et les principes proposés pour économiser de l'énergie.

La mise en œuvre de simples possibilités de traitement, de stockage, de détection, et de communication dans les dispositifs à petite échelle, à faible coût et leur intégration dans ce qu'on appelle des réseaux de capteurs sans fil ouvrent la porte à une multitude de nouvelles applications. Les réseaux de capteurs constituent une catégorie de réseaux sans fil comportant d'un très grand nombre de nœuds. Ils sont également caractérisés entre autre par un déploiement très dense et à grande échelle dans des environnements souvent limités en terme de ressources. Ces nœuds déployés autour ou dans une zone à observer sont utilisés pour l'acquisition de données et leur transmission à une station de traitement appelée communément « Station de Base ». Les spécificités les plus frappantes de ces nœuds sont leurs capacités d'auto-organisation, de coopération, leur rapidité de déploiement, leur tolérance aux erreurs et leur faible coût.

En terme de domaines d'applications, les réseaux de capteurs ont connu un très grand succès, car ils détiennent un potentiel qui révolutionne de nombreux secteurs de notre économie et notre vie quotidienne, de la surveillance et la préservation de l'environnement, à la fabrication industrielle, en passant par l'automatisation dans les secteurs de transport et de la santé, la modernisation de la médecine, de l'agriculture, de la télématique et de la logistique.

Un concept intéressant que nous introduisons dès à présent est le concept de *« nœud-capteur »* par référence au terme anglais *« sensor node »* qui revient fréquemment dans la littérature. Toute fois nous gardons à travers ce manuscrit d'autres appellations.

En considérant, les quatre unités de base d'un nœud capteur, la principale source de consommation énergétique est l'unité de communication. Approximativement 80% de l'énergie consommée par un nœud, l'est pendant la transmission d'informations [25].

Aussi, cette unité est associée à une technologie sans fil donc le choix d'une technologie sans fil est très important et exige de trouver un bon compromis entre la consommation, le débit et la portée en fonction de l'application visée. Il existe différentes technologies sans fil mais les plus adaptées aux systèmes embarqués et plus spécialement aux réseaux de capteurs sont celles utilisées par les réseaux de proximité sans fil pour leur faible consommation. C'est pour cette raison que nous allons faire un état de l'art sur les réseaux personnels sans fil [26].

2. Etat de l'art sur les réseaux personnels sans fil :

Contrairement aux réseaux locaux notés WLAN tels que Wifi (IEEE 802.11x) et ses multiples standards (a, b g, n et y diffèrent par la portée de leur signal et leur taux de transfert) ou encore hiperLAN2 (HIgh Performance Radio LAN 2.0) dont l'objectif est de permettre une communication sans fil avec un rayon d'une centaine de mètres et qui consomment beaucoup d'énergie de l'ordre du Watt, les réseaux personnels sans fil ou réseaux sans fil de proximité, notés WPANs, concernent les réseaux à faible portée de l'ordre de quelques mètres voire dizaines de mètres. Ils sont utilisés pour connecter des dispositifs autonomes entre eux et donc qui peuvent s'adapter aux réseaux de capteurs.

2.1 IRDA

L'IRDA qui est un acronyme de *Infrared Data Association*, communément appelé *infrarouge*, est un moyen de communication qui utilise comme médium la lumière infrarouge pour transmettre des informations. Le standard initial, normalisé en 1994, offrait un débit de 115 Kbit/s qui a abouti en 1999 à une extension allant jusqu'à 16 Mbits/s. La principale caractéristique de l'IRDA qui est aussi son principal inconvénient est que deux périphériques utilisant cette technologie doivent être en ligne de vue pour pouvoir communiquer. En effet, les transmetteurs IRDA étant directifs avec un angle d'environ 15°, ils doivent être bien orientés pour communiquer. Cette contrainte de fonctionnement pour cette technologie qui a une faible consommation électrique fait qu'au niveau sécurité elle limite les possibilités

d'interception du signal. Au début des années 1990, l'IRDA était très utilisé pour les téléphones portables, les ordinateurs et d'autres périphériques mais avec l'apparition de Bluetooth, la donne a changé.

2.2 IEEE 802.15.1/Bluetooth

Ce standard de communication créé en 1994 par Ericsson, a été défini à la base pour remplacer les câbles et combler les lacunes de l'IRDA qui était populaire à l'époque. C'est une technologie radio courte distance, faible consommation, basée sur des puces électroniques peu coûteuses et destinées à simplifier les connexions entre appareils électroniques. En 1998, d'autres entreprises rejoignent Ericsson pour former le Bluetooth *Special Interest Group* (SIG). Cependant, Bluetooth n'a pas seulement pour but de faire coopérer des périphériques de constructeurs différents en définissant un système de communication par ondes hertziennes sur la bande ISM de 2,4 GHz mais définit réellement une pile logicielle complète, contrairement à WiFi qui n'offre que le niveau 1 et le niveau 2 de la pile OSI. Elle permet aux périphériques de se découvrir et de communiquer entre eux sans savoir quels services ils offrent à la base. On distingue 3 classes de modules radio Bluetooth sur le marché ayant des puissances différentes et donc des portées différentes :

Classe	Puissance	Portée
1	100mW (20dBm)	100 mètres
2	2,5mW (4dBm)	15 à 20 mètres
3	1mW (0dBm)	1 mètre

La plupart des fabricants d'appareils utilisent des modules de classe 2 pour une consommation énergétique moindre, de l'ordre de 320 mW en fonctionnement et 60 mW hors association avec d'autre périphériques et un rayon de communication compatible avec les applications visées en domotique par exemple.

Les fonctionnalités des différentes couches de la pile Bluetooth (Fig. 1.1) sont les suivantes :

- L'interface de contrôle *HCI (Host Control interface)* propose une méthode d'accès uniforme aux fonctionnalités de la couche *Baseband* indépendamment de l'interface d'accès du matériel Bluetooth.

- Le protocole *L2CAP (Logical Link Control and Adaptation Protocol)* assure la transmission asynchrone des paquets. Il utilise le multiplexage, la segmentation et le réassemblage.

- Le protocole de découverte de service *SDP (Service Discovery Protocol)* permet à un appareil Bluetooth de rechercher d'autres appareils et d'identifier les services disponibles.

- Le protocole *RFCOMM* sert à transmettre des données aux couches de haut niveau. C'est un service basé sur les spécifications RS-232, qui émule des liaisons séries et peut notamment servir à faire passer une communication IP par Bluetooth.

- Le protocole *OBEX (Object Exchange)* est un protocole d'échange développé à la base pour l'IRDA permettant de transférer des objets.

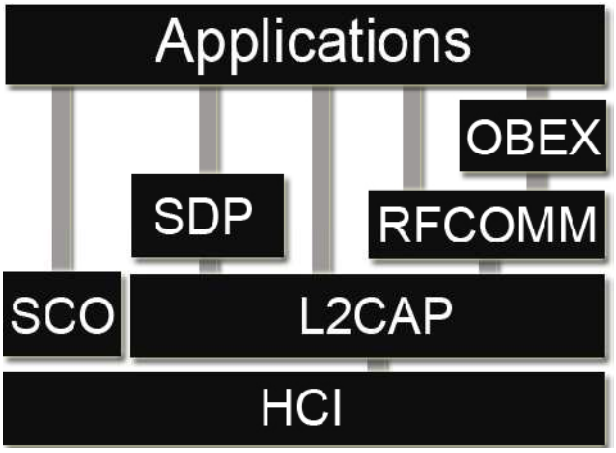


Fig. 1.1 Pile Bluetooth

Une spécificité de Bluetooth est le type de topologies qu'il offre. Les périphériques Bluetooth qui utilisent un même canal de communication forment un *Piconet*, constitué d'un maître et jusqu'à 255 esclaves mais seulement 7 esclaves peuvent être actifs, les autres sont "parqués" (Fig. 1.2).

Deux types de liens peuvent être établis avec un maître et un ou plusieurs esclaves : des liens synchrones orientés connexion (SCO), souvent utilisés pour le transport de la voix, et des liens asynchrones sans connexion (ACL).

L'interconnexion de plusieurs *Piconets*, appelé *scatternet* (Fig. 1.2), se fait au moyen d'un nœud qui est soit esclave dans un *Piconet* et maître dans l'autre *Piconet* soit esclave dans les deux *Piconets* car un nœud ne peut être maître dans différents *Piconets* simultanément. La formation des *scatternets* constitue un axe de recherche important dans le domaine des réseaux sans fil qui utilise la technologie Bluetooth. En effet, la scalabilité d'un réseau Bluetooth est un vrai handicap lorsque l'on veut étendre cette technologie à des réseaux denses qui s'éloignent d'une utilisation domestique.

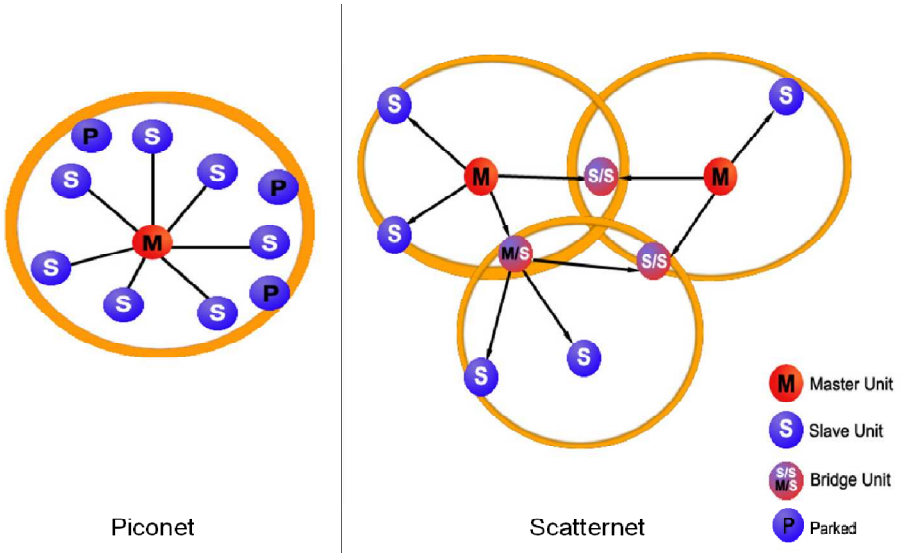


Fig. 1.2 Représentation des différents modèles réseau de Bluetooth

La bande de fréquence de 2.4GHz qu'utilise Bluetooth, libérée dans la plupart des pays du monde, est encombrée par plusieurs autres protocoles (WiFi, ZigBee, ...) et polluée par les fours à micro-ondes.

C'est pour cette raison que pour éviter les interférences avec d'autres modules radios, bluetooth utilise la technique *FHSS (Frequency Hopping Spread Spectrum)* qui est un étalement de spectre par saut de fréquence. Cette technique consiste à découper la bande de fréquence en 79 canaux d'une largeur de 1 MHz et change aléatoirement et fréquemment de fréquence avec une période de 625 µs, ce qui permet 1600 sauts par seconde.

Par contre, la mobilité, ou encore la transmission à travers des éléments non métalliques sont garanties par Bluetooth au détriment de la sécurité contrairement à l'IRDA.

2.3 Wibree - Ultra Low Power (ULP) Bluetooth

Le succès de Bluetooth fait qu'une nouvelle norme conçue par Nokia a été annoncée en Octobre 2006. Cette norme s'appelle Wibree [26] devenue Ultra Low Power Bluetooth car le Bluetooth SIG l'a acceptée en accord avec Nokia comme variante de la norme Bluetooth. Wibree est une technologie sans fil qui offre un débit d'environ 1 Mbps dans un rayon d'une dizaine de mètres et ne se positionne pas comme un concurrent de Bluetooth mais plutôt comme un complément. Elle opère dans la bande de 2,4 GHz et son principal avantage est sa consommation qui promet d'être 10 fois moins gourmande en énergie que Bluetooth. Cette norme constituera un bon compromis entre le débit et la consommation énergétique. On pourra ainsi l'adapter à des dispositifs de petite taille comme des montres par exemple.

2.4 HOME RF

HomeRF est une spécification de réseau sans fil conçue comme son nom l'indique pour les réseaux domestiques mais qui n'a pas connu un grand succès. Cette technologie permettrait aussi entre autre de soutenir la technologie de transport de la voix en mode numérique sur les réseaux sans fil appelée DECT qui est un acronyme de *Digital Enhanced Cordless Telephone*. Elle a été imaginée par un groupe de sociétés appelé HomeRF Working Group dont les principaux acteurs étaient Compaq, HP, IBM, Intel et Microsoft. Cependant, la disponibilité de la norme WiFi pour des réseaux domestiques, a entrainé l'éloignement de deux gros sponsors, Intel et Microsoft, ce qui fait que cette norme était en perte de vitesse. Malgré son accès au médium intéressant, *Shared Wireless Access Protocol*, qui est un mélange de CSMA/CA pour les données asynchrones et TDMA pour les données isochrones, elle est abandonnée aujourd'hui.

2.5 IEEE 802.15.3/UWB

L'UWB, qui est un acronyme de *Ultra Wide Band* ou encore *Ultra large bande passante* en français, est une technologie sans fil conçue pour faire communiquer des périphériques grand public sur une courte distance avec un haut débit, tout en considérant une des principales contraintes des dispositifs de domotique qui est l'autonomie. En effet, cette technologie, qui consomme peu d'énergie, est basée sur une technique d'étalement de spectre permettant de transmettre des données sur un très large spectre en un temps très court. Ce profil fait qu'elle est bien adaptée pour le transport de données nécessitant un haut débit tel que le multimédia. La promotion de cette technologie est assurée par un groupement d'industriels appelé *WiMedia Alliance*. Cependant, le coût des composants et la concurrence d'autres technologies comme l'USB sans fil ou encore Bluetooth fait qu'elle est en perte de vitesse. Cela se traduit par le départ d'un des promoteurs, Intel en novembre 2008 et l'arrêt en Mars 2009 de WiMedia. Néanmoins, il faut noter que les briques de l'UWB se retrouveront dans les spécifications des normes de l'USB sans fil et Bluetooth 3.0. En effet, la norme Bluetooth 3.0. encore appelée *Next-Gen Bluetooth* ou *Bluetooth UWB* aura sa couche matérielle basée sur l'UWB permettant d'atteindre un débit de 480 Mbits/s.

Face à ces différentes technologies, une s'est démarquée considérablement par rapport aux autres pour les réseaux de capteurs : IEEE 802.15.4/ZigBee.

2.6 IEEE 802.15.4/ZigBee

Contrairement à plusieurs applications et à plusieurs technologies sans fil comme celles qui précèdent où le débit est une priorité, l'émergence d'un nouveau type d'applications contraintes par la consommation énergétique pour une meilleure autonomie des dispositifs utilisés dans ce type de réseau a donné lieu à un nouveau standard à faible débit et à consommation réduite : IEEE 802.15.4.

Ce nouveau standard, qui fait partie du groupe des *Low-Rate Wireless Personal Area Networks (LR-WPANs)* c'est-à-dire aux *réseaux personnels sans fils à bas débit*, a été conçu pour connecter les dispositifs sans batterie ou contraint par leur batterie limitée en énergie. Son objectif est donc de consommer peu d'énergie de sorte qu'une batterie puisse tenir pendant très longtemps.

Ce standard spécifie les couches basses, *MAC* et *physique*, pour les LR-WPANs. Comme l'IEEE ne définit que la couche MAC et la couche physique, un groupe d'entreprises appelé la ZigBee Alliance [28] a spécifié les couches hautes, c'est-à-dire la couche réseau et la couche application pour ce standard allant du routage à l'application, ce qui a donné naissance au protocole *ZigBee* (Fig. 1.3). Un faible débit, une faible portée et une faible consommation énergétique sont les principales caractéristiques d'un RdC classique.

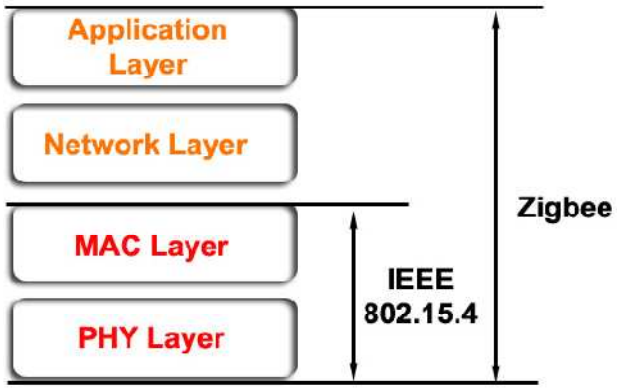


Fig. 1.3 La pile du protocole IEEE 802.15.4/ZigBee

D'une manière générale, nous pouvons conclure que les protocoles de communication changent régulièrement avec une durée de vie inférieure à 10 ans. Cela montre que ce domaine n'est pas encore assez mature et rend le travail du chercheur, dépendant des industriels du domaine, difficile.

Après s'être principalement focalisé sur deux des quatre principales unités d'un capteur qui sont : l'unité calcul et son système d'exploitation puis les technologies sans fil susceptibles d'être utilisées par l'unité de transmission (ou *transceiver*), nous allons observer la Fig. 1.4.

Cette figure montre que pour une unité de calcul précise, c'est à dire pour un micro-processeur donné, l'unité de communication correspondante, c'est à dire le transceiver ayant en moyenne le même débit, consomme plus d'énergie. Cette image montre aussi les technologies sans fil susceptibles d'être adaptées pour les réseaux de capteurs de par leur consommation énergétique telles que : ZigBee (IEEE 802.15.4), Bluetooth (IEEE 802.15.1) ou encore Wibree voir même WiFi (IEEE 802.11x) et ses multiples standards.

ZigBee et Bluetooth sont plus souvent utilisés à cause de leur faible consommation mais leur débit et la portée de leur signal est limitée. Aussi, l'architecture complexe de Bluetooth et

ses communications point à point font que Zigbee, facilitant les communications « un-vers-tous » pour le passage à l'échelle, connaît un plus grand succès dans les réseaux de capteurs.

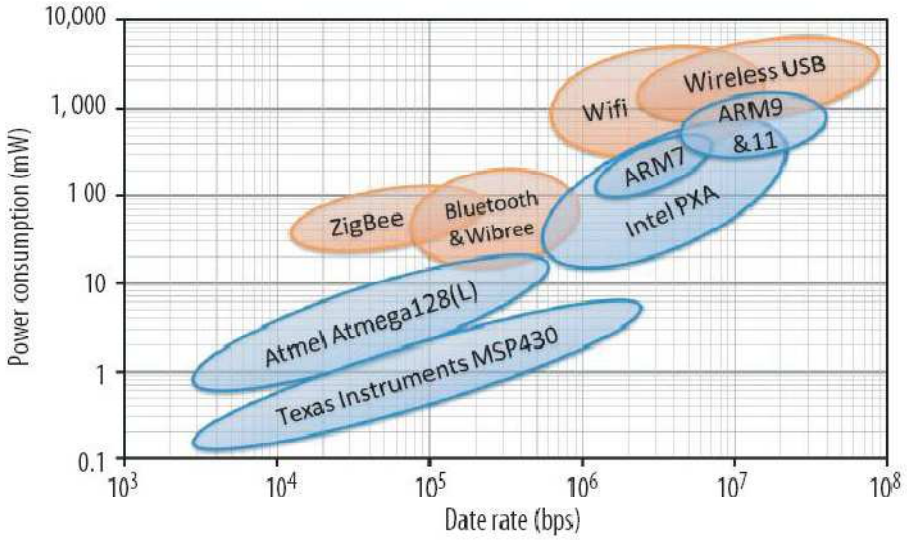


Fig. 1.4 Comparaison énergétique entre transceiver et micro-processeurs [5]

C'est la mise en réseau de plusieurs de ces capteurs dit *intelligents* et leur collaboration à l'aide de leur module de communication qui forment un RdC. Les capteurs deviennent ainsi des entités capables d'opérer en toute autonomie, dans tout environnement même hostile à l'homme, afin de collecter, traiter et envoyer des données relatives à un endroit du globe bien précis vers un lieu d'analyse.

3. Les réseaux *ad hoc*

Les réseaux de capteurs constituant une spécialisation des réseaux *ad hoc*, nous allons brièvement présenter ce type d'architecture réseau.

Originaire du latin, le mot *ad hoc* signifie *qui se satisfait de lui même*. Un réseau *ad hoc* se définit alors comme un système formé de nœuds ne disposant d'aucune infrastructure, communicant par ondes radio, où chaque nœud offre un service relais permettant de réémettre un message dont il n'est pas destinataire à un nœud qui est hors de portée radio de l'émetteur du message [39]. Cette technique de réémission très utilisée dans les réseaux *ad hoc* dépourvus d'infrastructure porte le nom de *multisaut* ou *multihop*.

A la base, les réseaux ad hoc sont une application militaire née dans les années 1970 avec le Projet PRNet de la DARPA qui correspond à l'agence pour les projets de recherche avancée de défense des Etats-Unis. Ces réseaux illustrent proprement le concept de lieu de combat où il n'y a pas d'infrastructure préexistante et dans lequel le *multisaut* permet de pallier le problème de dispositif n'étant pas à portée radio. D'autres projets militaires sponsorisés par DARPA suivront tel que SURAN (SURvivable Radio Network) en 1983, prolongement de

PRNet, ayant pour but de combler les lacunes rencontrées expérimentalement par PRnet avec un champ d'application un peu plus large que le champ de bataille ou encore GLoMo (Global Mobile Information System) en 1994 avec pour mission l'étude des possibilités d'adaptation des concepts d'internet à des terminaux mobiles.

Cette technologie finira par aussi susciter un intérêt pour les applications civiles qui se traduira par la création du groupe MANET par l'IETF en 1995. Les caractéristiques de ces réseaux font que la manière d'acheminer les informations est très importante. On distingue alors différents types de protocoles de routage illustrés sur la Fig. 1.5. Cette figure n'est qu'un support mais d'autres familles et références peuvent bien sûr être ajoutées.

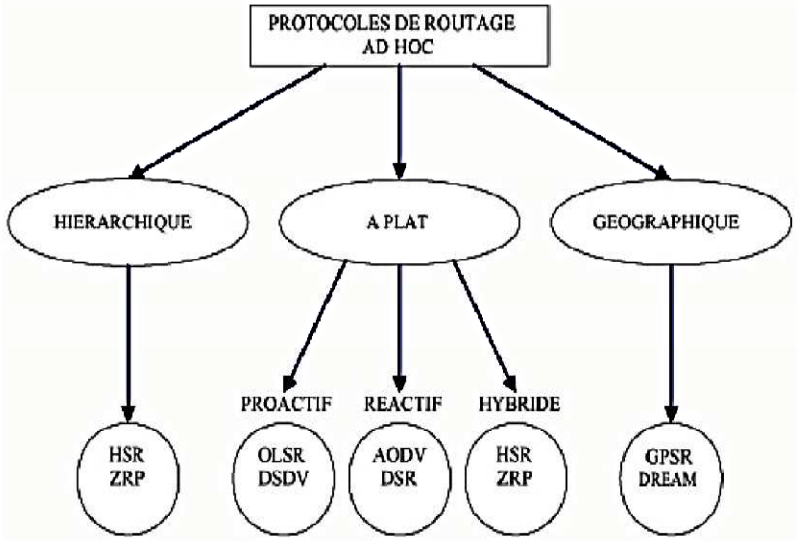


Fig. 1.5 Classifications des protocoles dans les réseaux ad hoc

Les principales caractéristiques dans ces environnements sont :

- une topologie dynamique,
- une sécurité des communications limitée,
- des contraintes d'énergie,
- une bande passante limitée,
- l'absence d'infrastructure.

Sur la Fig. 1.5, les principales familles de protocoles de routage pour les réseaux ad hoc sont : *hiérarchiques*, *géographiques* et *les protocoles à plat* que nous allons respectivement présenter dans les prochains paragraphes.

3.1 Les protocoles hiérarchiques

Le point clé dans un protocole de routage hiérarchique est la différenciation entre les nœuds du réseau. Cette distinction peut se faire au niveau *physique* ou au niveau *logique*.

1. *Au niveau physique* : lorsqu'il y a une différenciation au niveau matériel, les nœuds sont hétérogènes et diffèrent de par leurs capacités et fonctionnalités. De ce fait, certains nœuds disposent de caractéristiques supplémentaires et peuvent réaliser des tâches plus complexes que d'autres nœuds. Cependant, il existe un coût supplémentaire au niveau du prix et de la consommation énergétique pour les dispositifs qui ont plus de fonctionnalités.
2. *Au niveau logique* : dans un réseau homogène les nœuds peuvent également être hiérarchisés selon leurs fonctionnalités. Dans un découpage en cluster par exemple, permettant de partitionner le réseau et donc de le structurer, certains nœuds appelés *clusterheads* ou encore tête de clusters ont pour rôle d'organiser leurs clusters respectifs. Les *clusterheads* qui sont des nœuds de niveau supérieur dans la hiérarchisation peuvent communiquer exclusivement entre eux et forment un espace appelé *backbone*.

Le principe de base d'un protocole de routage hiérarchique est de créer une topologie virtuelle dans un réseau dense, puis de l'exploiter pour router les paquets. Cette architecture facilite l'échange de paquets lors du passage à l'échelle de ce type de réseau car il y a des échanges intra-zones et inter-zones organisés.

3.2 Les protocoles géographiques

Dans les protocoles qui utilisent le concept de routage géographique, les nœuds sont supposés connaître leur position :

- Soit en utilisant un module externe comme le GPS,
- Soit en utilisant des mesures du signal (RSSI, TDOA, AOA).

Une fois les positions connues par les nœuds, ils déterminent le chemin vers une destination en fonction de ses coordonnées [5]. Le protocole LAR par exemple exploite la connaissance des coordonnées du nœud émetteur et du nœud récepteur pour optimiser la procédure d'inondation. Il est aussi important de noter que l'ajout d'un module de géolocalisation, comme le GPS par exemple, est coûteux, ce qui poussent certaines solutions à se servir juste de quelques nœuds repères, appelés *ancres*, qui connaissent leurs positions pour déterminer la position des autres nœuds en utilisant des techniques de triangularisation.

3.3 Les protocoles à plats

Les protocoles à plat sont les plus répandus car ils peuvent être utilisés par les autres familles de protocoles comme les protocoles hiérarchiques et géographiques. Dans ces

protocoles, tous les nœuds ont le même rôle et sont homogènes en terme de ressources. On distingue trois catégories de routage pour ces protocoles :

1. le routage proactif où les routes sont établies à l'avance,
2. le routage réactif où les routes sont recherchées à la demande,
3. le routage hybride qui constitue un mélange des deux précédents.

La liste des protocoles n'est pas exhaustive car ce n'est pas notre sujet mais nous avons donné juste un aperçu de ce qui existe avec les principaux protocoles de chacune des catégories.

4. Architecture d'un nœud de capteur :

Les capteurs sans fil sont conçus comme de véritables systèmes embarqués, doté de moyens de traitement et de communisation de l'information, en plus de leurs fonctions initiale de relever des mesures. Ils représentent une révolution technologique des instruments de mesure, issus de la convergence des systèmes électroniques miniaturisés et des systèmes de communication sans fil.

Un nœud capteur est composé de plusieurs éléments ou modules correspondant chacun à une tâche particulière d'acquisition, de traitement, ou de transmission de données. Il comprend également une source d'énergie comme illustré dans la Fig 1.7 :

L'unité d'acquisition de données : Contient le ou les capteurs embarqués sur le nœud. Le principale fonctionnement des détecteurs est le même : il s'agit de répondre à une variation de condition de l'environnement par une variation de certains caractéristiques électriques. Les variations de tensions sont ensuite converties par un *convertisseur Analogique/Numérique (ADC)* afin de pouvoir être traitées par l'unité de traitement.

L'unité de traitement de données : Elle est généralement dotée d'un microcontrôleur dédié et de la mémoire.

Les microcontrôleurs utilisés dans les réseaux de capteurs sans a faibles consommation d'énergie. Leurs fréquences sans assez faibles, moins de 10 MHz pour une consommation d'ordre 1 mW. Une autre caractéristique est la taille de la mémoire qui est d'ordre de 10 Ko de RAM pour les données, et de 10 Ko de ROM pour les programmes [2]. Cette mémoire consomme la majeure partie de l'énergie allouée au microcontrôleur, c'est pourquoi on lui adjoint souvent de la mémoire flash moins coûteuse en énergie.

Outre le traitement des données, le microcontrôleur commande également toutes les autres unités notamment le système de transmission.

L'unité de transmission de données : Elle est le plus souvent constituée d'un transcepteur radio qui fournit au capteur la capacité de communique avec les autres au sein d'un réseau. Elle met en œuvre des protocoles dépendant de la technologie utilisée (par exemple 802.11, 802.15.1, 802.15.4, etc. pour les technologies sans fil) [1].

Les composants utilisés pour réaliser la transmission sont des composants classiques. Ainsi on trouve les mêmes problèmes que dans tous les réseaux sans fil : la quantité d'énergie nécessaire à la transmission augmente avec la distance. Pour les réseaux sans fil classiques (*LA N, GSM*) la consommation d'énergie est de l'ordre de plusieurs centaines de milliwatts, et on se repose sur une infrastructure alors que pour les réseaux de capteurs, le système de transmission consomme environ 20 mW et possède une portée de quelques dizaines de mètres. Pour augmente ces distances tout en préservant l'énergie, le réseau utilise un routage *multi-sauts* [18].

La source d'énergie : Pour les réseaux de capteur sans fil autonomes, l'alimentation est une composante cruciale. Il y'a essentiellement deux aspects : premièrement, stocker l'énergie et de la fournir sous la forme requise ; deuxièmement, tenter de reconstituer l'énergie consommée par un réapprovisionnement grâce à une source externe au nœud-capteur telles les cellules solaires. Le stockage de l'énergie se fait traditionnellement en utilisant ses piles [2].

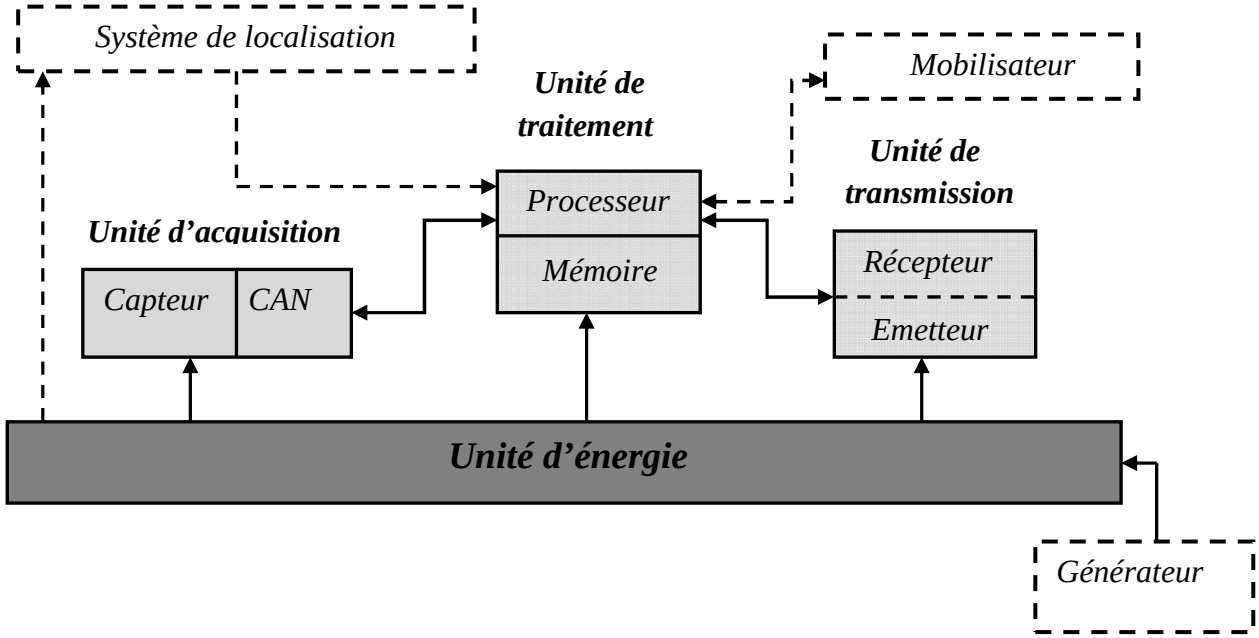


Fig 1.6 : Anatomie générale d'un nœud de capteur

En fonction des applications pour lesquelles ils sont conçus, les réseaux de capteurs sans fil pourraient également avoir d'autres modules, comme une Unité de Localisation, afin d'identifier leurs position géographique, par exemple en utilisant un récepteur GPS ou une technique triangulation. Certaines applications pourraient aussi avoir besoin de capteur équipés d'un *Mobilisateur* pour qu'ils puissent se déplacé.

Enfin s'il est nécessaire qu'un nœud soit maintenu en activité pendant une très longue période de temps, un *Générateur de Puissance*, tel que cellules solaire, serait utile afin de tenir le nœud alimenté électriquement sans avoir à changer ses batteries [1].

Caractéristiques d'un nœud capteur sans fil:

En analysant la gamme des réseaux de capteurs disponibles sur le marché et les prototypes présentés dans la littérature, il est évident que la principale caractéristique d'un nœud de capteur sans fil est sa *petite taille*. Depuis que les premiers nœuds de capteurs sans fil sont apparus, la tendance est la *miniaturisation*. Une deuxième caractéristique, évidente mais essentielle, est l'*autonomie* (pas seulement du point de vue de leur source d'énergie, mais aussi de leur fonctionnement). Ces deux premières particularités induisent plusieurs autres caractéristiques à considérer, en particulier la vitesse de calcul et la vitesse de transmission. Des performances élevées en termes de vitesse de traitement et de transmission impliquent une consommation d'énergie élevée. De manière générale, il est souhaitable que la durée de vie de la batterie de nœud soit la plus grande possible, donc les différentes unités qui composent un nœud sont généralement très limitées en termes de ressources et de performance pour que leur consommation d'énergie soit extrêmement faible [1].

La Fig.1.7 résume la consommation d'énergie dans les différentes unités d'un capteur :

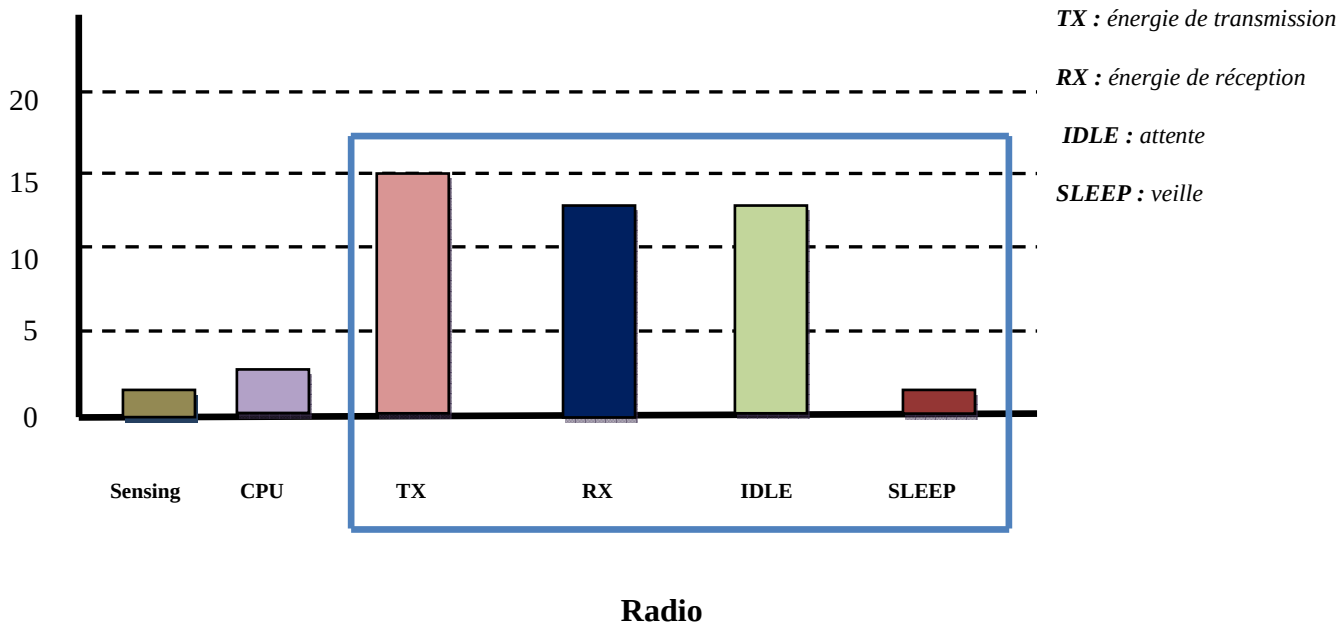
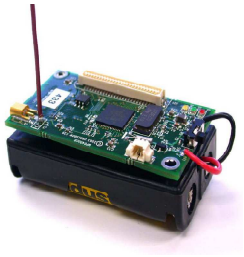


Fig.1.7 La consommation d'énergie dans les différentes unités d'un capteur

5. Technologies des nœuds capteurs :

Le développement rapide dans les domaines de l'informatique, de l'électronique et des technologies sans fil a abouti à l'émergence de petits dispositifs distribués capables de surveiller de manière autonomes nos environnements pour nous prévenir des éventuelles catastrophes : les réseaux de capteurs. Ces mêmes progrès scientifiques qui mettent à disposition différentes plateformes de réseau de capteurs (Fig.1.8), procurent une vision pragmatique à la recherche dans les RdCs.

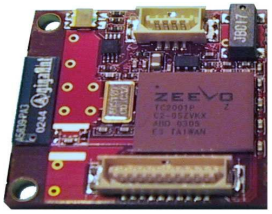
(a) Spec node



(b) Mica2

(c) MicaZ

(d) Telos



(e) imote

(f) BTnode

(g) Stargate

Fig. 1.8 Les plateformes de réseaux de capteurs

En utilisant ces plateformes, les chercheurs peuvent valider et exploiter des résultats obtenus dans des conditions réelles et ne pas se contenter de résultats fournis par des simulateurs. La Fig. 1.8 illustre bien la disponibilité du matériel, la multiplicité des plateformes et la rapidité de renouvellement de celles-ci où quasiment chaque année une nouvelle plateforme est mise sur le marché. Il existe encore beaucoup d'autres plateformes, mais pour ne pas surcharger l'image elles n'ont pas été toutes représentées. Cependant, une

liste exhaustive est disponible sur le site The Sensor Network Museum [30]. Les applications utilisant les réseaux de capteurs étant de plus en plus fréquentes et complexes, le nombre de plateformes aussi se multiplie vu le nombre croissant d'applications visées. Celles-ci différent de par leurs capacités mémoire, leurs unités de traitement, leurs modules de transmission ou encore leurs tailles.

Sur la Fig.1.8, *(a) Spec node* qui est une puce conçue à l'Université de Californie, à Berkeley, pour des opérations de coût ultra-faible et de faible puissance. Ce petit bijou technologique mesure approximativement 2mm x 2.5mm. Il se compose d'un microcontrôleur 8 bit Atmel AVR RISC, une mémoire (3K) et une communication radio basée sur la modulation par déplacement de fréquence (FSK).

Les motes *Mica2 (b), MicaZ (c)* et *Telos (d*) ce sont des plateformes Crossbow [31], qui sont les plus populaires et permettent des captures génériques.

Les motes *Mica2*, utilisant des piles de types AA pour fonctionner, sont équipées de plusieurs capteurs : lumière, température, pression barométrique, accéléromètre, et disposent d'un cœur de processeur Atmel AVR. Celles-ci diffèrent des motes *micaZ* au niveau du module de transmission qui est respectivement pour la mica2 et MicaZ un Chipcon CC1000 et un Chipcon CC2420. La principale différence entre ces deux tranceivers est le fait que le Chipcon CC2420 est conforme au standard IEEE 802.15.4/ZigBee tandis que le Chipcon CC1000 utilise les bandes de fréquence 315/433/868 et 915 MHz dont certaines sont interdites dans certains pays du monde tels que le Japon (qui interdit l'utilisation entre 433-915 MHz) ou encore l'Europe (qui interdit la bande de 915 MHz). A celà s'ajoute la différence de portée qui est plus longue par exemple à 433 MHz mais aussi le débit qui est respectivement pour le Chipcon CC1000 et le Chipcon CC2420 de 38,4 Kbps et de 250 Kbps.

La famille des *Telos*, c'est à dire des nœuds de capteurs avec USB, diffère principalement de celles qui précèdent au niveau du microcontrôleur. Les nœuds de type telos tournent sur des microcontrôleurs fabriqués par Texas Instruments appelé TI MSP430 [32], conçus pour des applications embarquées à basse consommation. C'est un dispositif qui consomme très peu et compatible avec la norme IEEE 802.15.4 pour la communication sans fil. Comme la mote MicaZ son transceiver est de type Chipcon CC2420.

Imote (e) développé par Intel Research et *BTnode (f)* élaborée à l'ETH de Zurich. Ces capteurs dotés d'architectures 32 bits et à haut débit qui utilisent la technologie sans fil Bluetooth comme moyen de communication. En effet, ils utilisent Bluetooth qui a un débit plus élevé et donc peuvent transmettre des informations plus volumineuses ou encore le même type information mais plus rapidement.

Stargate (g) à haut débit, capable de capturer des informations, les agréger puis les transmettre et ainsi donc de former une passerelle avec d'autres réseaux tel que internet. Il est conçu par Intel et produit par Crossbow. La plateforme Stargate est basée sur un processeur

32 bit d'Intel, PXA-255 XScale à 400 MHz, qui dispose de 32 Mo de mémoire flash, 64 Moctets de mémoire SDRAM, et de connecteurs Crossbow pour les motes mica2 ou micaZ ainsi qu'une entrée PCMCIA pour avoir une interface Bluetooth ou IEEE 802.11.

Il faut noter que l'énergie consommée est proportionnelle au nombre de portes logiques (CISC > RISC, ou encore une architecture 32 bits > 16 bits), à la vitesse (plus il est lent et moins il y a de communication de portes et donc la consommation est réduite) et à la durée de fonctionnement pour exécuter une tâche donnée [5]. De là, nous comprenons aisément que la gestion d'énergie, qui est un point critique dans les réseaux de capteurs, a entrainé le développement de capteurs de très petites puissances limitées par rapport à la multitude d'applications de plus en plus complexes d'ou la naissance d'algorithmes adaptés à ces contraintes tels que les algorithmes de compression qui seront présentés dans le Chapitre 3 mais aussi de plateformes de puissance plus élevée pour réaliser des tâches complexes.

6. Caractéristiques des réseaux de capteurs :

1.1 Architecture d'un réseau de capteur :

Un réseau de capteur sans fil, est un système distribué de grande échelle mettant en communication un grand nombre d'entités autonomes « nœuds capteurs ». Ces nœuds sont reliés à une ou plusieurs passerelles (*sinks*) qui permettent l'interconnexion avec d'autre réseaux (Internet, satellites,…).

Dans un scénario d'application classique, plusieurs nœuds capteurs sont déployés dans une zone « sensor field », un certain environnement pour mesurer certains phénomènes physiques et de faire remonter les informations collectées à une station de base, *puits* ou *sinks* (une porte d'entrée vers le monde extérieur qui fait l'interface entre le réseau capteur et l'utilisateur des données). Dans le cas le plus simple, les nœuds seront dans le voisinage direct du puits (un réseau de types étoilé a un saut). Cependant, dans le cas d'un réseau à grande échelle, les capteurs ne sont pas tous dans le voisinage du puits et les messages seront acheminés du nœud source vers le puits en transitant par plusieurs nœuds, selon un mode de communication multi-saut, comme l'illustre la Fig.1.9 :

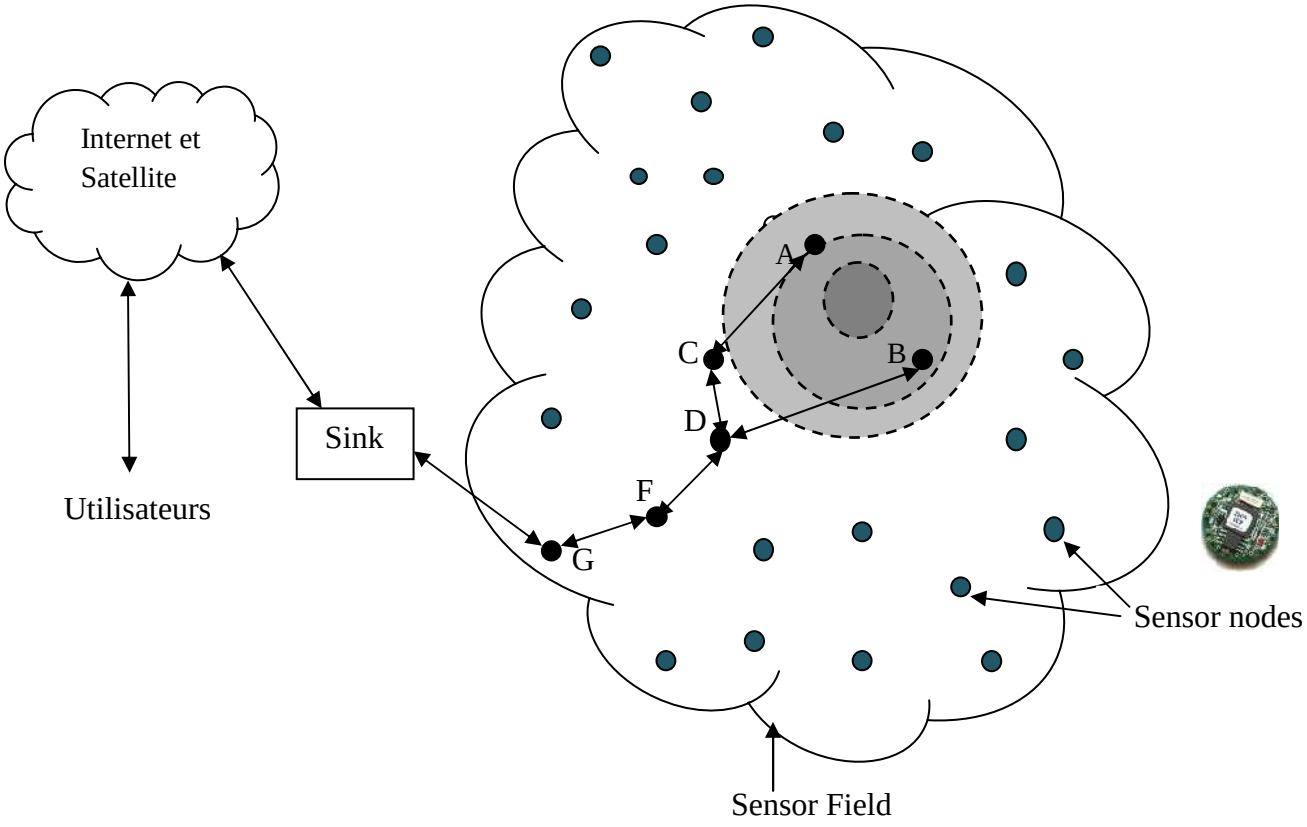


Fig1.9 : Architecture d'un Réseau de Capteur

Un réseau de capteur sans fil est un type particulier de *réseaux ad-hoc* qui sont utilisés pour l'interconnexion spontanée des systèmes informatiques [18]. Dans un *réseau ad-hoc,* les entités sont en mesure de s'organiser entre elles pour former le réseau sans l'aide d'une infrastructure fixe définie à l'avance, ni d'une intervention humaine. Les nœuds ont la capacité de jouer le rôle de routeurs.

Les principales différences entre les réseaux de capteurs sans fils et les *réseaux ad-hoc* traditionnels sont le problème de l'énergie et le facteur d'échelle.

1.2 Les différents facteurs de conception :

La conception des réseaux de capteurs est influencée par de nombreux facteurs comme la tolérance aux pannes, les coûts de production, la consommation d'énergie, l'environnement ou la topologie du réseau. Ces facteurs représentent la base de la conception de protocoles ou d'algorithmes pour les réseaux de capteurs.

6.2.1 Tolérance aux pannes :

Les nœuds peuvent être sujets à des pannes dues à leur fabrication (ce sont des produits de bon marché, il peut donc y avoir des capteurs défectueux) ou plus fréquemment à un manque

d'énergie. Les interactions externes (chocs, interférences) peuvent aussi être la cause des dysfonctionnements.

La propriété de tolérance aux pannes est définie par l'habilité du réseau à maintenir ses fonctionnalités sans interruptions provoquées par la panne les capteurs. Elle vise donc à minimiser l'influence de ces pannes sur la tâche globale du réseau [3]. Afin que les pannes n'affectent pas la tâche première du réseau, il faut évaluer la capacité du réseau à fonctionner sans interruptions.

Les protocoles conçus pour les réseaux de capteurs doivent atteindre le niveau de tolérance aux pannes requit par les réseaux, cela dépend essentiellement de l'environnement de déploiement du réseau, des caractéristiques des nœuds capteurs, etc.

6.2.2 Coût de fabrication :

Les nœuds sont des produits fabriqués en série du fait de leur grand nombre. Il faut que le coût de fabrication de ces nœuds soit tel que le coût global du réseau ne soit pas supérieur à celui d'un réseau classique afin de pouvoir justifier son intérêt.

6.2.3 Topologie du réseau :

En raison de leur forte densité dans la zone à observer, il faut que les nœuds-capteurs soient capables d'adapter leur fonctionnement afin de maintenir la topologie souhaitée.

On distingue généralement trois phases dans la mise en place et l'évolution d'un réseau :

- *Déploiement* : Les nœuds sont soit répartis d'une manière prédéfinie soit de manière aléatoire (lancés en masse depuis un avion). Il faut alors que ceux-ci s'organisent d'une manière autonome.
- *Post-déploiement - Exploitation* : Durant la phase d'exploitation, la topologie du réseau peut être soumise à des changements dus à des modifications de la position des nœuds ou bien à des pannes.
- *Redéploiement* : L'ajout de nouveaux capteurs dans un réseau existant implique aussi une remise à jour de la topologie.

6.2.4 Consommation d'énergie *:*

L'économie de l'énergie est une des problématiques majeures dans les réseaux de capteurs. En effet, la recharge de ressources d'énergie est souvent trop coûteuse et parfois impossible. Il faut donc que les capteurs économisent au maximum l'énergie afin de pouvoir fonctionner. Les réseaux de capteurs fonctionnent selon un mode de routage par saut, chaque nœud du réseau joue un rôle important dans la transmission de données. Le mauvais fonctionnement d'un nœud implique un changement dans la topologie et impose une réorganisation du réseau. C'est pour cela que le facteur de consommation d'énergie est d'une importance primordiale dans les réseaux de capteurs. La majorité des travaux de recherche se concentrent sur ce

problème afin de concevoir des algorithmes et protocoles spécifiques à ce genre de réseau qui consomment le minimum d'énergie.

6.2.4.1 Phases de consommation d'énergie :

Détecter les événements dans l'environnement capté, élaborer un traitement de données local et rapide, et transmettre les résultats à l'utilisateur sont les principales tâches d'un nœud dans un réseau de capteurs. Les étapes de consommation d'énergie par ce nœud peuvent être, dès lors, divisées en trois phases : *le captage, la communication et le traitement de données*.

Phase de captage

L'énergie consommée au moment de captage varie suivant la nature de l'application. Un captage sporadique consomme moins d'énergie qu'un contrôle d'événement constant. La complexité de l'événement à détecter joue également un rôle crucial pour déterminer la quantité d'énergie consommée. Les environnements contenant un niveau de bruit élevé entraîne l'augmentation de l'énergie nécessaire pour cette phase.

Phase de communication

Parmi les trois phases citées auparavant, la phase de communication de donnée est celle qui consomme la plus grande quantité d'énergie [4], ceci, à cause de la multitude de composants électroniques intégrés au circuit responsable de cette opération. Cette phase implique les deux étapes d'*émission* et de *réception* de données. Il est démontré que pour les communications à courte portée, avec une faible puissance de radiation, les coûts énergétiques pour l'émission et la réception de données sont pratiquement égaux.

Durant cette phase, il est important de considérer l'énergie nécessaire pour la mise en marche du circuit de communication, le temps de démarrage étant égal à plusieurs centaines de microsecondes rend l'énergie consommée durant cette période non négligeable. L'influence de cette étape sur la quantité globale d'énergie consommée par la communication augmente quand la taille des paquets transmis diminue.

Par conséquent, une bonne politique de consommation d'énergie passe obligatoirement par éviter au maximum le recourt à la mise en marche et l'arrêt fréquents des circuits de communication.

Phase de traitement de données

Comparé à la phase de communication, l'étape de traitement local des données consomme beaucoup moins d'énergie [4]. En effet, le coût énergétique nécessaire pour transmettre 1 KB sur une portée de 100 m est approximativement égal à celui nécessaire pour exécuter 3 millions d'instructions à une vitesse de 100 millions instructions par seconde, ce fait, favorise largement le traitement local des données pour l'amélioration de la consommation d'énergie dans les réseaux de capteurs.

Les nœuds capteurs doivent donc posséder des moyens de traitement local de données, tout en restant capable d'interagir avec les nœuds avoisinants.

Enfin il est à noter qu'un nœud peut contenir des circuits additionnel pour le codage/décodage des données, en plus de certain circuits spécifiques aux applications du réseau, dans tous ces cas, la conception des algorithmes et protocoles du réseau est influencé largement par l'énergie consommée par ces circuits en plus de ceux invoqués précédemment.

1.3 Architecture protocolaire :

La pile de protocole utilisée par le « *puits* » ou *« Sink »* ainsi que par les autres nœuds-capteurs est donnée dans la figure 1.10. Cette pile de protocoles combine routage et gestion d'énergie et intègre les données avec les protocoles réseau. Elle communique de manière efficace (terme d'énergie) à travers le support sans fil et favorise les efforts de coopération entre les nœuds-capteurs. La pile protocolaire comprend *une couche application, une couche transport, une couche réseau, une couche liaison de données, une couche physique, un plan de gestion d'énergie, un plan de gestion de mobilité et un plan de gestion des tâches.* Selon les tâches de détections, différents types de logiciels d'applications peuvent être construits et utilisés dans la couche application. La couche transport contribue au maintien du flux de données si l'application de réseau de capteurs l'exige. La couche réseau s'occupe de l'acheminement des données fournies par la couche transport. Comme l'environnement sujet au bruit et que les nœuds-capteurs peuvent être mobiles, le protocole MAC doit tenir compte de la consommation d'énergie et doit être en mesure de réduire les collisions entre les nœuds voisins lors d'une diffusion par exemple. La couche physique répond aux besoins d'une modulation simple mais robuste, et de techniques de transmission et de réception.

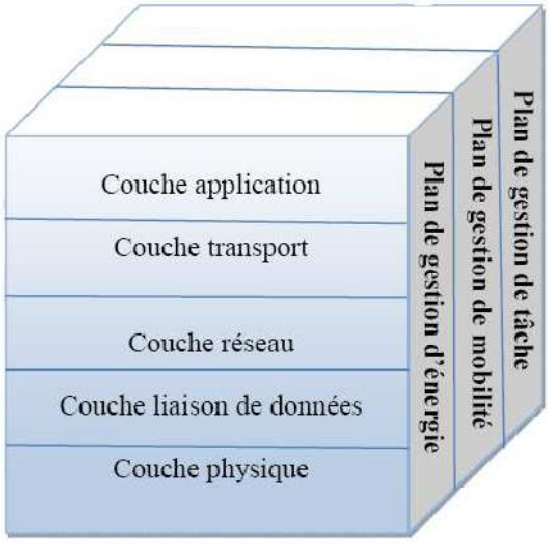


Fig 1.10 : La pile protocolaire des réseaux de capteurs.

En outre, les *plans de gestion d'énergie*, de *mobilité* et des *tâches* surveillent et gèrent la consommation d'énergie, les mouvements, et la répartition des tâches entre les nœuds-

capteurs. Ces plans aident à coordonner les tâches de détections et à réduire l'ensemble de la consommation d'énergie.

Plans 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 nœud diffuse à ses voisins une alerte les informant qu'il ne peut pas participer au routage. L'énergie restante est réservée au captage.

Plans de gestion de mobilité : détecte et enregistre le mouvement du nœud capteur. Ainsi un retour arrière vers l'utilisateur est toujours maintenu et le capteur peut garder trace de ses nœuds voisins. En déterminant leurs voisins, les nœuds capteurs peuvent balancer l'utilisation de leur énergie et la réalisation de tâche.

Plans 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 nœuds de cette région effectuent la tâche de captage au même temps, certains nœuds exécutent cette tâche plus que d'autres selon leur niveau de batterie.

1.4 Vue d'ensemble des plates-formes existantes :

Comme un certain nombre de technologies connues à ce jour, les nœuds de capteurs sans fil doivent être nés d'un projet militaire, ce qui entrave la mise en place d'une chronographie précise de leur développement. Cependant le titre de premier prototype de nœuds de capteurs sans fil correspond au module LWIM (Low-power Wireless Integrated Microsensors) développé dans le milieu des années 90 par l'Agence pour les Projets de Recherche Avancée de Défense (DARPA) des Etats-Unis et l'UCLA [1]. Il s'agissait d'un géophone équipé d'un capteur de transmission radiofréquence et d'un contrôleur PIC. Depuis un peu plus de 10 ans, la technologie de capteurs sans fil a beaucoup évolué. Les modules deviennent de plus en plus petits et les durées de vie prévues augmentent. Aujourd'hui, le marché de nœuds à été ouvert à l'industrie. Le fournisseur le plus connu est Crossbow Inc., avec son offre de capteur Mica2 et Micaz.

Le tableau 1.1 recense les différents composants actuellement disponibles sur le marché.

Le concept prévalent dans le développement des nœuds de capteurs est la conception modulaire. En effet, tous les nœuds de la table 1.1, sont en fait des cartes intégrées qui regroupent l'unité de communication et l'unité de traitement, tandis que l'unité de captage est conçue comme une carte distincte qui peut être attaché sur l'unité principale. Cela permet bien sûr de pouvoir réutiliser les mêmes unités pour différentes applications. Par exemple un nœud Mica2 peut être combiné avec une carte MTS310, qui comprend un capteur de température, capteur de lumière, un capteur de son, un capteur de champ magnétique, et un accéléromètre à deux axes. De même nous pouvons combiner le nœud Mica2 avec une carte MTS420 pour le doté d'un capteur d'humidité et d'un capteur de pression barométrique, et même d'un GPS

pour le positionnement géographique. Une autre possibilité pour la même unité est l'ajout d'une carte d'acquisition MDA320 [1].

Compte tenu des impératifs d'économie d'énergie que doivent respecter les nœuds de capteurs sans fil, un grand nombre de capteur peuvent basculés, par programmation, dans différents modes d'activités. Ainsi, un nœud capteur peut passer d'un mode actif, où le nœud est en pleine capacité de travail (toutes les unités sont opérationnelles), à un mode sommeil, où tout ou partie de ses éléments sont inactivés pour économiser l'énergie. Dans ce dernier mode, le minimum est laissé actif de sort que le nœud puisse revenir à l'état s'il juge nécessaire (par exemple, après un certain temps).

La plupart des fabricants adoptent des émetteurs RF à basse fréquence. Certains ont choisi de mettre en œuvre un protocole d'origine récente conçus pour les modules sans fil industriels et spécifié dans la norme IEEE 802.15.4. Ce protocole de transmission opère dans la bande de fréquence des 2.4GHz. Les microcontrôleurs choisis sont généralement d'une faible vitesse et de très faible consommation d'énergie. De même, la mémoire disponible pour les programmes et les données est très réduite en comparaison avec celle des équipements informatiques d'aujourd'hui.

Plate-forme	Fabriquant	Unité de Traitement	Unité de Communication	Unité de Captage	Unité de Puissance
MICA2	Crossbow	Atmel ATMega128L (128 Ko de mémoire de programme, 4 Ko RAM) 512 ko mémoire flash pour des données EEPROM 4 Ko (configuration)	CC1000 (radio transcepteur multi-freq. 868/916 - 433 - 315 MHz, 38.4Kbaud)	Connecteur pour carte de capteurs externe	2.7 - 3.3V
MICAZ	Crossbow	Atmel ATMega128L 512 ko mémoire flash pour des données EEPROM 4 Ko (configuration)	Chipcon CC2420 (radio transcepteur 802.15.4, bande ISM de 2400 à 2483.5 MHz, 250 kbps)	Connecteur pour carte de capteurs externe	2.7 - 3.3V
IRIS	Crossbow	Atmel ATMega1281 (128 Ko de mémoire de programme, 8 Ko RAM) 512 ko mémoire flash pour des données EEPROM 4 Ko (configuration)	Radio transcepteur 802.15.4 (bande ISM, de 2400 à 2480 MHz, 250 kbps)	Connecteur pour carte de capteurs externe	2.7 - 3.3V

Imote2	Crossbow	Intel PXA271 256 ko mémoire SRAM 32 Mo mémoire SDRAM 32 Mo mémoire flash	TI CC2420 (bande ISM, de 2400 à 2483.5 MHz, 250 kbps)	Connecteur pour carte de capteurs externe	3.2 - 4.5V
Tmote Sky	Moteiv (Sentilla)	Texas Instruments MSP430 F1611 (10Ko RAM, 48Ko Flash, 128o stockage d'information)	Chipcon CC2420	Connecteur pour carte de capteurs externe	2.1 - 3.6V
BTnode rev3	ETH	Atmel ATMega128L 64+180 Kbyte RAM EEPROM 4 Ko	Bluetooth, CC1000	Connecteur pour carte de capteurs externe	DC externe 3.8 - 5V ou 2AA
Particle 2/29	TECO	PIC 18F6720 (20 MHz), Mémoire interne : 128Ko de mémoire de programme, 4Ko RAM, 1Ko EEPROM, 512 Ko Mémoire flash pour des données	TR1001 (RFM, bande passante 125Ko, bands ISM 868.35 ou 315 MHz)	Connecteur pour carte de capteurs externe	0.9 - 3.3 V

Tab. 1.1: Caractéristiques de nœuds de capteurs existants actuellement

2. Application des réseaux de capteurs sans fil :

Plusieurs types d'applications peuvent être développés pour les réseaux de capteurs sans fil. Selon le mode de communication des données de mesure, nous identifions quatre grands scénarios d'applications :

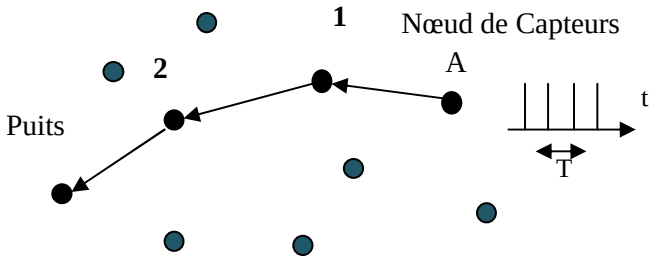
Applications périodiques : les capteurs prennent des mesures dans des intervalles de temps réguliers, et ils envoient des données au puits de manière périodiques. Dans l'exemple de la figure 1.12 (a), une image est capturée périodiquement par le nœud A, puis, il envoie les paquets vers le puits à travers les nœuds 1 et 2.

Applications à la demande (On-Demand) : Les capteurs attendent de recevoir un ordre du puits pour déclencher une mesure et l'envoyer. Cet ordre peut être généré par la demande manuelle d'un utilisateur humain ou d'une tâche automatique programmée. Dans l'exemple de la figure 1.12 (b), une demande est adressée au nœud source A, le message est acheminé à travers les nœuds 2 et 1, et à sa réception, A active son unité de captage et envoie ces mesures vers le puits, cette fois par le chemin constitué des nœuds intermédiaires 3, 4 et 2.

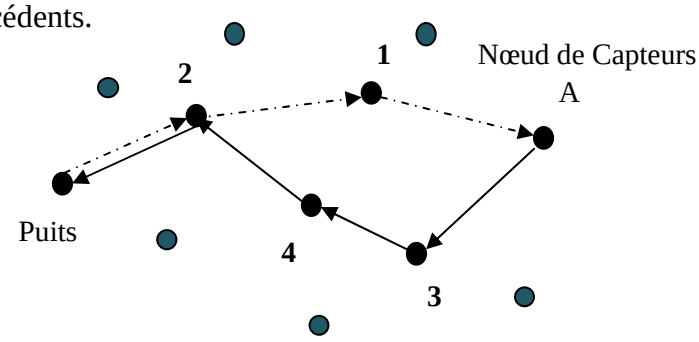
Applications événementielles (Event-Driven): Dans ce type d'applications, l'envoi de données vers le puits est déclenché lorsqu'un événement particulier est détecté. Les

événements peuvent être causés par un dépassement de seuil dans les mesures récoltés par le capteur. Dans l'exemple de la figure 1.12 (c), le nœud capteur A détecte un événement causé par un objet qui traverse sa zone de détection, et commence à envoyer ces mesures vers le puits à travers les nœuds capteurs 1 et 2.

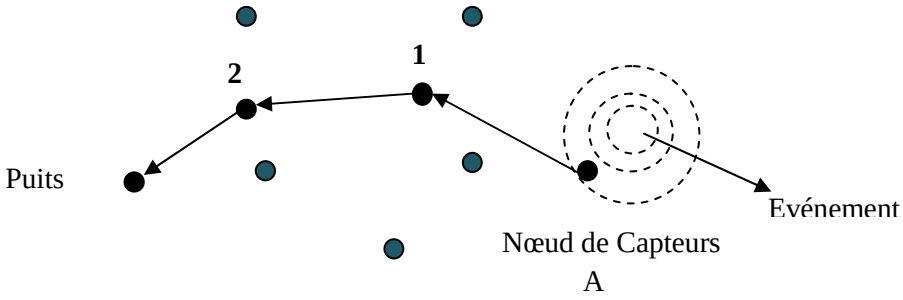
Applications hybrides : Toute alliance des cas précédents.



(a)Une application de capture périodique. Un nœud Prend des mesures périodiques, et envoie des paquets vers le puits à travers les nœuds 1 et 2



(b) Une application à la demande. Une demande est adressée au nœud capteurs A à travers les nœuds 2 et 1, puis A 1 et 2, puis A prend de mesures et les envois par paquet vers le puits à travers les nœuds 3, 4 et 2



(c) Une application événementielle. Un nœud de capteurs A détecte un événement causé par un objet qui traverse sa zone de détection, et commence a envoyer les valeurs de ces mesures.

Fig 1.11: Quelques exemples de types de scénario pour les réseaux de capteurs sans fil

Les réseaux de capteurs sans fil ont trouvé un ensemble de très vaste d'application dans divers domaines [6], parmi lesquels on peut citer les applications militaires, environnementales, industrielles et de surveillances en général.

Applications militaires :

Les premières applications potentielles des réseaux de capteurs ont concerné le domaine militaire. L'idée était de déployer un réseau de capteurs nanoscopiques (donc invisibles) sur des champs de bataille ou des zones ennemies pour surveiller les mouvements des troupes. Historiquement, le projet DARPA, qui a déjà été cité, a donné comme résultat les nœuds expérimentaux LWIM (très rudimentaires et assez volumineux) qui communiquaient selon une topologie en étoile. Les applications militaires sont les premières et certainement les plus représentatives des applications trouvées actuellement dans le domaine des réseaux de capteurs sans fil.

Dans [7], nous trouvons les résultats d'une expérience intitulée « A Line in the Sand » (« Une ligne dans le sable ») où un réseau de capteurs sans fil était déployé dans un scénario de sécurité. Ce réseau était constitué de 90 nœuds Mica2 dotés de capteurs de métaux et de capteurs de mouvement TWR-ISM-002. Il a été déployé sur la base militaire de MacDill (Air Force Base) à Tampa (Floride), et d'autres zones d'expérimentation de la même ampleur. L'objectif du réseau de capteurs était de détecter et suivre les mouvements d'objets mobiles intrus. Le système devait être en mesure de classer les objets détectés dans le champ d'action du réseau. Trois différents groupes d'objectifs ont été classés en tenant compte des caractéristiques détectables telles que leur quantité de métal et de leur rapidité de mouvement : personne non armée, soldat et véhicule blindé. Les résultats de l'expérience montrent une précision largement acceptable dans la reconnaissance des objets.

Applications environnementales :

Une application très représentative a été effectuée dans l'île Grand Duck (44.09N, 68.15W), à Maine. Un réseau de 32 nœuds a été déployé pour la surveillance de l'habitat d'espèces protégées [8]. Les unités déployées étaient des nœuds Mica et elles ont été utilisées pour étudier le comportement de l'océanité culblanc (*oceanodroma leucorhoa*), conformément aux changements climatiques. Les nœuds, dont certains ont été installés dans les nids des oiseaux, étaient capables de mesurer la température, la pression barométrique et d'humidité, et de transmettre les données dans un mode multi-saut jusqu'à un puits, puis vers une station de base accessible à partir d'Internet. Une application similaire peut être trouvée dans [9], concernant l'étude des oiseaux de mer dans une réserve nationale naturelle au Royaume-Uni.

De nombreuses applications de réseaux de capteurs se concentrent sur la mesure de phénomènes climatiques qui permettent d'étudier des changements dans l'environnement de certaines espèces animales ou végétales, afin de mieux comprendre leur comportement et, dans certains cas, supporter des études de réintroduction et de sauver des espèces qui sont en cours de disparition. Un exemple supplémentaire d'application est documenté dans [10], pour l'étude à long terme des espèces végétales en danger.

D'autres applications environnementales sont destinées à la surveillance de certains phénomènes climatiques afin de détecter ou de prévoir certaines catastrophes naturelles telles que l'éruption des volcans (Harvard Sensor Networks Lab, 2004 - 2008), les inondations [11] et les incendies de forêt [12].

Application industrielles :

Les technologies sans fil n'ont pas encore atteint leur apogée dans les industries, néanmoins nous commençons à voir aujourd'hui une augmentation du nombre de produits proposés pour ce milieu qui est WINA (Wireless Industrial Networking Alliance). Certains protocoles comme la norme 802.15.4 sont en cours d'évaluation afin de déterminer s'ils peuvent supporter certaines des contraintes typiques des applications industrielles, telles que la communication temps réel [13] et la robustesse aux erreurs de transmission [14]. Pour le moment, l'utilisation de technologies de réseaux de capteurs sans fil est encore, dans la plupart des cas, en stade expérimental.

***Applications médicales* :**

Le domaine médical constitue un intérêt de plus en plus grandissant pour les nouvelles technologies [5]. On utilise de plus en plus de matériels hautement technologiques en médecine. L'apport de ces techniques permet non seulement de garder le patient à son domicile mais aussi de lui éviter le traumatisme de l'hospitalisation. Il faut ajouter qu'à l'aide de capteurs, les comportements anormaux des personnes dépendantes tels que des chutes, des chocs ou des cris peuvent être détectés, ce qui facilitera les interventions immédiates. Dans le même sens, le projet STAR (Système Télé-Assistance Réparti) de l'équipe LIMOS de Clermont-Ferrand en collaboration avec le service de cardiologie du CHU de la même ville propose une plateforme de télésurveillance novatrice permettant de suivre en continu et à distance les personnes ayant des troubles du rythme cardiaque [5].

***La domotique* :**

Nous imaginons très bien les maisons du futur où une véritable interaction avec des capteurs embarqués permettra de contrôler localement ou à distance des appareils domestiques [5]. Ces smart home, c'est à dire ces maisons intelligentes, vont faciliter les activités domestiques quotidienne telles que l'automatisation de l'activation/l'extinction de la lumière qui existe déjà dans certains garages ou encore la mise en marche automatique de la télévision, la climatisation ou le chauffage lorsqu'il y a une présence dans le salon.

3. Types de réseaux de capteurs (« Mostly-on » et « Mostly-off »):

Le comportement global de la consommation dans les réseaux de capteurs sans fil est axé sur l'application. Par conséquent nous trouvons plusieurs types de réseaux en fonction du modèle de délivrance de données qu'impose l'application (continue, initié par la Station de Base ou Hybride). Ce modèle influe sur l'activité radio des nœuds, ce qui fait que plusieurs types de réseaux existent, en l'occurrence les réseaux *« Mostly-on »* et les réseaux *« Mostly-off »*.

Les réseaux de capteurs *« Mostly-off »* est une catégorie de réseaux de capteurs, où les nœuds gardent leurs radios éteintes pendant de très longues périodes. Par la suite ces nœuds se réveillent pour envoyer leurs données à un collecteur. Le problème qui revient le plus souvent dans ce type de réseaux est le problème de synchronisation entre les nœuds dû à la dérive d'horloge. Par antonymie, nous qualifierons de *« Mostly-on »* les réseaux de nœuds dont les radios sont la plus part du temps allumées.

4. Déploiement:

Le déploiement constitue la première étape de la mise en place d'un réseau de capteur. C'est une étape, tout comme l'analyse des informations collectées par la station de base, qui nécessite une intervention humaine. Ce déploiement peut se faire de deux manières :

Déterministe : c'est-à-dire que la position des capteurs est bien déterminée et donc connue à l'avance

Aléatoire : dans ce cas les capteurs s'auto-organisent pour former le réseau.

Le nombre important de nœuds utilisés dans un réseau de capteur empêche leur déploiement suivant un plan soigneusement établi, cependant un schéma général pour le déploiement initial doit être conçu pour permettre de réduire les coûts d'installation, augmenter la flexibilité d'arrangement des nœuds et pour faciliter l'auto-organisation des nœuds et leur tolérance aux pannes [3].

Les capteurs sont déployés dans le but d'acquérir des informations sur une zone bien précise d'où la nécessité de présenter les caractéristiques de la couverture de surface.

5. La couverture de surface et la collecte de données :

La couverture de surface peut se définir comme la capacité des capteurs à surveiller une zone. Les capteurs doivent non seulement couvrir une surface et acquérir des données mais aussi communiquer entre eux pour fusionner, comparer des informations, voir réaliser un multihop (multi-saut) jusqu'au Sink. Pour pouvoir exploiter les données collectées, il faut que le graphe soit *fortement connexe*, c'est-à-dire qu'il est impératif que tous les nœuds puissent trouver un chemin vers la station collectrice afin de lui transmettre les fruits d'une observation. La couverture de surface fait l'objet de nombreux sujet de recherche et s'appuis aussi sur des travaux existants tirés de la théorie des graphes.

11. L'énergie dans les Réseaux de Capteurs :

La contrainte énergétique constitue un grand défit dans les réseaux de capteurs. Une grande partie des études dans ce domaine se focalise sur des techniques d'économie d'énergie pour maximiser le temps de vie de ces réseaux. En effet, les nœuds capteurs sont alimentés principalement par des batteries. Ils doivent donc fonctionner avec un bilan énergétique frugal. En outre, ils doivent le plus souvent avoir une durée de vie de l'ordre de plusieurs mois, voir de quelques années, puisque le remplacement des batteries n'est pas une option envisageable pour des réseaux avec des milliers de nœuds.

11.1 La contrainte énergétique :

Nous constatons sur la figure 1.12 que malgré le développement rapide des processeurs, des mémoires, des technologies sans fil et des capacités de stockages, la capacité énergétique des batteries évolue à contrario lentement. C'est pour cette raison qu'il existe multiples technologies pour utiliser les batteries le plus longtemps possible telles que la mise en veille,

l'agrégation de données pour éviter l'envoie de donnée redondante ou encore la compression de donnée qui minimise la quantité de données à transmettre, qui sera sujet de notre travail.

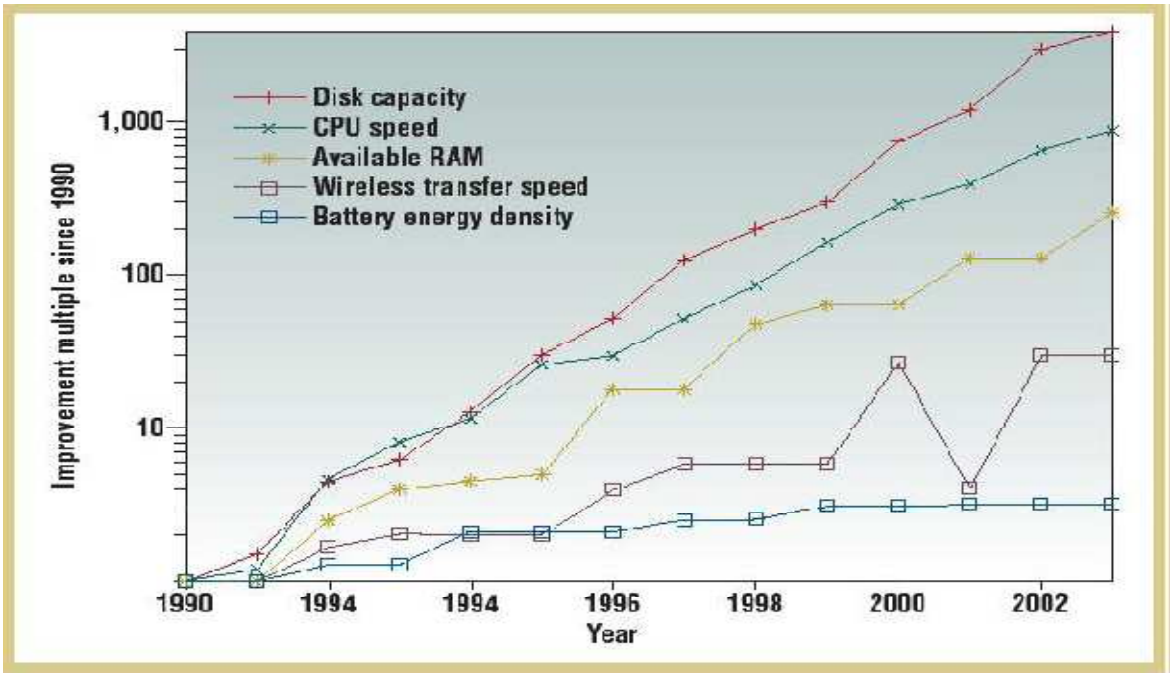


Fig. 1.12: Evolution des technologies informatique et électronique entre 1990 - 2003 [17]

Après avoir annoncé la contrainte sèvre qui est celle d'énergie, il faut notes que les protocoles de routage utilisés pendant la collecte de données doivent être particuliers. En effet, contrairement aux réseaux ad hoc qui sont moins denses et où les nœuds ont leur propre objectif, dans les réseaux de capteurs, les nœuds collaborent pour un objectif commun.

La gestion des flots de données diffère en ce sens que les réseaux de capteurs utilisent une technique *Many to one* tandis que dans les réseaux ad hoc c'est du *any to any*. La différenciation au niveau des flots de données et la contrainte d'énergie encore plus sévère chez les capteurs a donné naissance à de nouveaux protocoles de routage pour réseaux de capteurs, qui sont définis chacun selon un des trois modes de collecte d'information existant, *à intervalle régulier* ou *périodiques*, *sur demande* et *évènementiel*.

11.2Consommation d'énergie d'un nœud-capteur :

11.2.1 Formes de dissipation d'énergie :

Afin de concevoir des solutions efficaces en énergie, il est extrêmement important de faire d'abord une analyse des différents facteurs provoquant la dissipation de l'énergie d'un nœud capteur [4].
Cette dissipation d'énergie se fait de manière générale selon plusieurs modes :

- ***MCU (Unité du microcontôleur):*** Généralement les MCUs possèdent divers mode de fonctionnements : *actif, « idle »* et *sommeil*, à des fins de gestion d'énergie. Chaque mode est caractérisé par une quantité différente de consommation d'énergie. Par exemple, le microcontrôleur MSP430, consomme 3 mW en mode actif, 98 µW dans le mode *« idle »*, et seulement 15 µW dans le mode *sommeil* [2]. Toute fois la transition entre les modes de fonctionnement implique un surplus d'énergie et de latence. Ainsi les niveaux de consommation d'énergie des différents modes, les coûts de transition entre les modes, mais encore le temps passé par le MCU dans chaque mode ont une incidence importante sur la consommation totale de l'énergie d'un nœud-capteur.

- ***La radio :*** la radio opère dans quatre modes de fonctionnement : *émission, réception, « idle », sommeil.* Une observation importante dans le cas de la plupart des radios est que le mode *« idle »* induit une consommation d'énergie significative, presque égale à la consommation en mode *réception* [2]. Ainsi, il est plus judicieux d'éteindre complètement la radio plutôt que de passer en mode *« idle »* quand l'on a ni à émettre ni à recevoir de données. Un autre facteur déterminant est que, le passage de la radio d'un mode à un autre engendre une dissipation énergétique importante due à l'activité des circuits électrique. Par exemple, quand la radio passe du mode sommeil au mode émission pour envoyer un paquet, une importante quantité d'énergie est consommée pour le démarrage de l'émetteur lui-même [4]. Un autre point important est que les données des constructeurs sous-estiment assez régulièrement ces différentes consommations, en particulier concernant la consommation dans le mode *« idle »*.

- ***Le détecteur (capteur) :*** il y'a plusieurs sources de consommation d'énergie par le module de détection, notamment l'échantillonnage et la conversion des signaux physiques en signaux électriques, le conditionnement des signaux et la conversion analogique-numérique. Etant donné la diversité des capteurs, il y'a pas de valeur typiques de l'énergie consommée. En revanche, les capteurs passifs (température, sismique, …) consommant le plus souvent peu d'énergie par apport aux autres composants du nœud-capteur. Notons les capteurs actifs tels que les sonars, les capteurs d'images, etc. peuvent consommer beaucoup d'énergie.

En outre, il existe d'autres formes de dissipations d'énergie telles que les lectures et écritures mémoire. Un autre aspect non négligeable est le phénomène d'auto-décharge de la batterie. En effet, cette dernière se décharge d'elle-même et perd de sa capacité au fil du temps.

Il est difficile de porter ici une étude quantitative et comparative précise de la consommation de chaque composant d'un nœud-capteur en raison du grand nombre de plates-formes commerciales existantes. Cependant, des expérimentations ont montés que c'est la transmission de données qui est la plus consommatrice en énergie [4]. Le coût d'une transmission d'un bit d'information est approximativement le même que le coût au calcul d'un millier d'opérations [15]. La consommation du module de détection dépend du type spécifique du nœud-capteur.

11.2.2 Sources de surconsommation d'énergie :

Nous appelons surconsommation d'énergie toute consommation inutile que l'on peut éviter afin de conserver l'énergie d'un nœud-capteur. Les sources de cette surconsommation sont nombreuses, elles peuvent être engendrées lors de la détection lorsque celle-ci est mal gérée (par exemple par une fréquence d'échantillonnage mal contrôlée).

La surconsommation concerne également la partie communication. En effet, cette dernière est sujette à plusieurs phénomènes qui surconsomment de l'énergie surtout au niveau MAC où se déroule le contrôle d'accès au support sans fil. Certains de ces phénomènes sont les causes majeures de la perte d'énergie :

- ***Les collisions :*** elles sont la première source de perte d'énergie. Quand deux trames sont émises en même temps et se heurtent, elles deviennent inexploitables et doivent être abandonnées. Les retransmettre par la suite, consomme de l'énergie. Tous les protocoles MAC essayent à leur manière d'éviter des collisions. Les collisions concernent plutôt les protocoles MAC avec contention.

- ***L'écoute à vide (idle listening) :*** un nœud dans l'état « *idle* » est prêt à recevoir un paquet, mais il n'est pas actuellement en trains de recevoir quoi que ce soit. Ceci est coûteux et inutile dans le cas des réseaux à faible charge de trafic. Plusieurs types de radios présentent un coût en énergie significatif pour le mode *« idle ».* Eteindre la radio est une solution, mais le coût de la transition entre les modes consomme également de l'énergie, la fréquence de cette transmission doit alors rester raisonnable.

- ***L'écoute abusive (overhearing) :*** cette situation se présente quand un nœud reçoit des paquets qui ne lui sont pas destinés. Le coût de l'écoute abusive peut être un facteur dominant dans la perte d'énergie quand la charge du trafic est élevée et la densité des nœuds grande, particulièrement dans les réseaux *« mostly-on ».*

- ***L'overmitting :*** un nœud envoie des données et le nœud destinataire n'est pas prêt à les recevoir.

- ***L'overhead des paquets de contrôles :*** l'envoi, la réception et l'écoute des paquets de contrôle consomment de l'énergie. Comme les paquets de contrôles ne transportent pas directement des données, ils réduisent également le débit utile effectif.

11.3 Mécanismes de conservation de l'énergie :

Comme nous l'avons souligné au par avant, c'est la transmission de données qui se révèle extrêmement consommatrice par apport aux tâches du nœud capteur. Cette caractéristique conjuguée à l'objectif de maximisation de la durée de vie du réseau à suscité de nombreux travaux de recherche. Avant de citer ces travaux dans le chapitre suivant, nous introduisons dans ce paragraphe certains mécanismes de base :

- ***Mode d'économie d'énergie :*** ce mode est possible quelque soit la couche MAC adoptée. Cela consiste à éteindre le module de communication dès que possible. Par exemple, les protocoles MAC fondés sur la méthode *TDMA (Time Division Multiple Access)* offrent une solution implicite puisqu'un nœud n'échange de message que dans des intervalles de temps qui lui sont attribués. Il peut alors garder sa radio éteinte durant les autres slots. Il faut toutefois veiller à ce que le gain d'énergie obtenue en mettant en veille le module radio ne soit pas inférieur au surcoût engendrer par le redémarrage de ce module.

- ***Traitement local :*** L'idée de cette technique est que la source peut se censurer. Ainsi une programmation événementielle semble bien adaptée aux réseaux de capteurs. Seuls les changements significatifs de l'environnement devaient provoquer un envoi de paquets dans le réseau. Dans le même état d'esprit, une grande collaboration est attendue entre les capteurs d'une même région en raison de leur forte densité et dans la mesure où les observations ne varient presque pas entre les voisins très proche. Ainsi les données pourront être confrontées localement et agrégées au sein d'un seul et unique message. Cette stratégie de traitement local permet de réduire sensiblement le trafic.

- ***Organisation des échanges :*** ce procédé revient à limité les problèmes de retransmissions due aux collisions. La solution extrême consiste à utiliser la technique d'accès au medium TDMA [19]. Les collisions sont ainsi fortement réduites. Cette solution présente un inconvénient d'être un peu flexible et de demander une synchronisation fine des capteurs. Des solutions intermédiaires ont vu le jour, par exemple S-MAC *(Sensor MAC)* [16] qui est une méthode d'accès au canal de type CSMA-CA avec le mécanisme RTS/CTS *(Request to Send, Clear to Send)* qui permet d'éviter des collisions et le problème de la station cachée. La principale innovation, apportée par ce protocole, est d'avoir un mécanisme de mise en veille distribué sur chaque nœud du réseau dans le but de réduire la consommation d'énergie. La principale difficulté de S-MAC est également de synchroniser les nœuds entre eux pour que la communication soit toujours possible.

- ***Limitation des accusés de réception :*** l'acquittement systématique est mal adapté à des réseaux denses : il provoque une surcharge du réseau et donc des collisions et des

interférences avec les données utiles échangées dans le réseau. Les acquittements par « *piggy-backing* » seront à privilégier.

- ***Réparation de la consommation d'énergie :*** la formation de « *clusters* » (groupe de nœuds) permet d'envisager des réseaux comportant un très grand nombre de capteurs. Elle favorise une meilleure répartition de la consommation de l'énergie. En effet, dans le cas d'une transmission directe vers l'observateur Fig.1.13 (a), les capteurs éloignés vont plus rapidement manquer d'énergie et les autres nœuds peuvent être sujets au phénomène d'*overhearing* dans le cas des réseaux « *mostly-on* ». Au contraire, dans le cas d'une transmission par saut Fig.1.13 (b), les nœuds proches de l'observateur vont être vite en rupture de batterie car ils seront plus sollicités pour relayer les messages des autres. La solution consiste à hiérarchiser les échanges en divisant la zone d'observation en *clusters* Fig.1.13 (c). Un « *clusterhead* » est élu pour chaque *cluster*. Il s'occupe de récupérer les informations auprès des capteurs de son *cluster* et de les transmettre directement à l'observateur. En changent régulièrement de *clusterhead*, on obtient un réseau dans lequel aucun capteur n'est prédisposer à arriver en rupture de batterie avant les autres. Mettre en place des *clusters* va également permettre de cloisonner le réseau et ceci dans l'objectif de réduire les interférences. On améliore ainsi la qualité du lien radio et par conséquent, on limite les retransmissions liées aux reprises sur erreur. L'exemple phare d'une solution avec des *clusters* est le protocole LEACH [3].

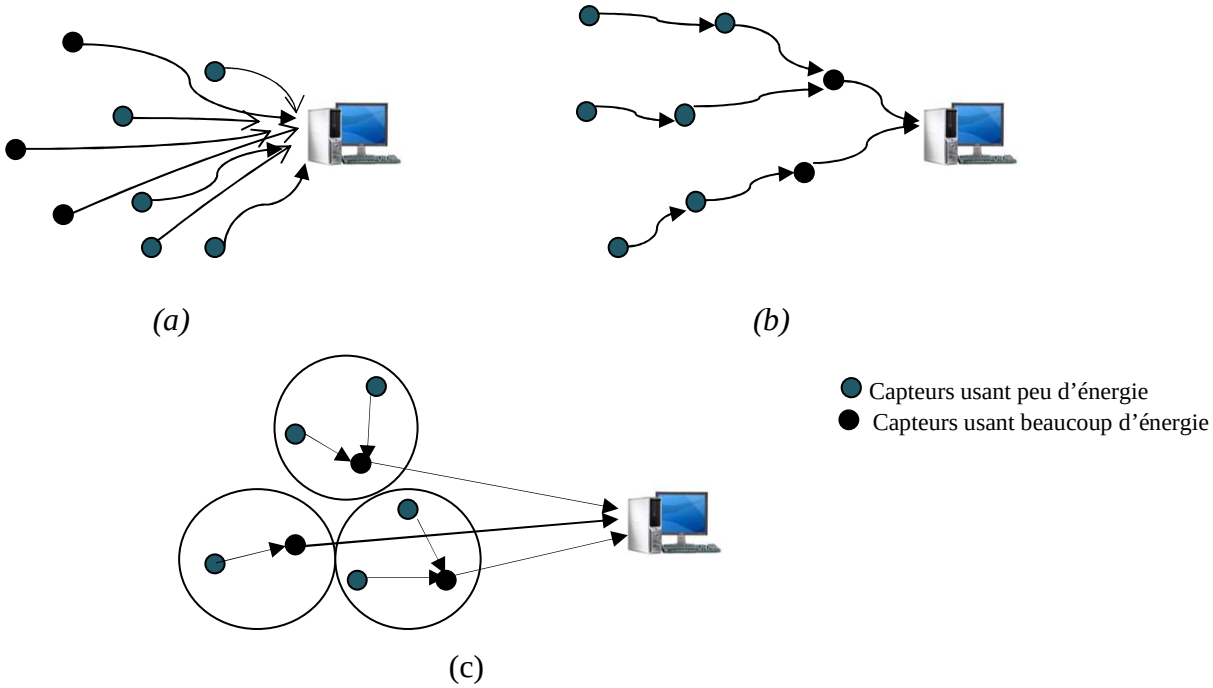


Fig 1.13 : (a) Transmission Directe. (b)Transmission Saut Par Saut. (c) Hiérarchisation en clusters

Par ailleurs, il existe dans la littérature d'autres mécanismes de conservation d'énergie, telles les techniques de compression, d'agrégation et de fusion de données, de mise en veille et d'autres technique de routage etc.

Nous les retrouverons dans les prochains chapitres.

12. Conclusion :

Les réseaux de capteurs sans fil présentent un intérêt considérable et une nouvelle étape dans l'évolution des technologies de l'information et de la communication. Cette nouvelle technologie suscite un intérêt croissant vu la diversité de ces applications.

Dans ce chapitre nous avons procédé à l'étude des réseaux de capteurs sans fil. Nous avons posé les briques de bases et fédéré quelques concepts nécessaires à la compréhension de nos problématiques dans la suite de ce manuscrit.

Cela fait que les réseaux de capteurs suscitent un engouement important dans la recherche. Nous avons remarqué à travers nos lectures que *« minimiser la consommation d'énergie d'un nœud-capteur »* est le « cheval de bataille » de toutes les solutions et protocole proposés. En effet, lorsque ce n'est pas l'objectif principal, alors c'est sûrement un critère de performance capital. Nous avons évoqué les formes de dissipations et les sources de surconsommation d'énergie par un nœud-capteur.

Dans la prochaine partie nous conjuguons cela avec la notion de durée de vie du réseau et nous dresserons un panorama des techniques de conservation d'énergie proposées dans la littérature afin de prolonger la durée de vie des réseaux de capteurs.

Chapitre II: Techniques de conservation d'énergie existantes

Sommaire

1. Introduction :

La durée de vie est sans doute la métrique la plus importante dans l'évaluation des performances d'un réseau de capteurs. En effet, dans un environnement contraint, toute ressource limitée doit être prise en compte. Toutefois, la durée de vie du réseau comme mesure de la consommation d'énergie, occupe une place exceptionnelle puisqu'elle constitue la borne supérieure de l'utilité de ce réseau. La durée de vie est également considérée comme un paramètre fondamental dans un contexte de disponibilité et de sécurité dans les réseaux de capteurs sans fil.

Maximiser la durée de vie du réseau revient à réduire la consommation énergétique des nœuds. Malgré les progrès qui ont été faits, la durée de vie de ces dispositifs à piles continue d'être un défi majeur et un facteur clé, exigeant davantage de recherches sur l'efficacité énergétique des plates-formes et des protocoles de communication.

Suite à l'introduction du chapitre précédent sur les formes de dissipation et de surconsommation d'énergie des nœuds-capteurs, nous présentons dans cette partie un panorama de techniques et de mécanismes de conservation d'énergie.

2. Energie et durée de vie d'un réseau de capteur:

La durée de vie est un élément essentiel pour tout réseau de capteur sans fil. Le but de n'importe quelle application est d'avoir des nœuds placés sur le terrain pour des mois ou des années. Le principal facteur limitant la durée de vie d'un réseau de capteurs est l'énergie. Comme la seule source d'énergie d'un capteur est une batterie à durée de vie limitée, l'optimisation énergétique doit être prise en considération quelque soit le problème traité. En effet, un réseau de capteur ne peu pas survivre si la perte de nœuds est très importante car ceci engendre des pertes de communication dues à une grande distance entre les nœuds restants. Donc il est très important que les batteries durent le plus longtemps possible, étant donné que dans la pluparts des applications il est impossible de les changer.

Il existe différentes définitions pour la durée de vie d'un réseau de capteurs (fondées sur la fonctionnalité désirée). Elle peut être définie par la durée jusqu'au moment où le premier nœud meurt. Elle peut également être définie par le temps jusqu'au moment où une proportion de nœuds meurt. Si la proportion de nœuds morts dépasse un certain seuil, cela peut avoir comme conséquence la non-couverture de sous-régions et/ou le partitionnement du réseau. Les définitions possibles et proposées dans la littérature sont les suivantes [2]:

1. La durée jusqu'à ce que le premier nœud épuise toute son énergie ;
2. La durée jusqu'à ce que le premier *clusterhead* épuise toute son énergie ;
3. La durée jusqu'à ce qu'il reste au plus une certaine fraction β de nœuds survivants dans le réseau ;

4. Demi-vie du réseau : la durée jusqu'à ce que 50% des nœuds épuisent leurs batteries et s'arrêtent de fonctionner
5. La durée jusqu'à ce que tous les capteurs épuisent leur énergie ;
6. La durée jusqu'à ce que le réseau soit partitionné : apparition de la première division du réseau en deux (ou plus). Cela peut correspondre aussi à la mort du premier nœud (si celui-ci tient une position centrale) ou plus tard si la topologie du réseau est plus robuste ;
7. k-couverture : la durée pendant laquelle la zone d'intérêt est couverte par au moins k nœuds ;
8. 100%-couverture
   _ (a) La durée pendant laquelle chaque cible est couverte par au moins un nœud ;
   _ (b) La durée pendant laquelle l'ensemble de la zone est couverte par au moins un nœud ;
9. α-couverture
   _ (a) La durée cumulée, au bout de laquelle au moins une portion α de la région est couverte par au moins un nœud ;
   _ (b) La durée pendant laquelle la couverture tombe en-dessous d'un seuil prédéfini α ;
   _ (c) La durée de fonctionnement continu du système avant que la couverture ou la proportion de paquets reçus (PDR pour Packet Delivery Ratio) tombent en-dessous d'un seuil prédéfini ;
10. La durée pendant laquelle un pourcentage donné de nœuds possèdent un chemin vers la Station de Base ;
11. L'espérance de l'intervalle complet pendant lequel la probabilité de garantir simultanément une connectivité et une k-couverture est au moins α ;
12. La durée jusqu'à la perte de la connectivité ou de la couverture ;
13. La durée jusqu'à ce que le réseau ne fournisse plus un taux acceptable de détection d'événements ;
14. La durée pendant laquelle le réseau satisfait continuellement les besoins de l'application

Finalement, nous constatons bien que plusieurs définitions convergent puisque certaines d'entre elles ne sont que des relaxations des autres et la majorité suggère que la durée de vie du réseau dépend de la consommation d'énergie de ses nœuds. Toutefois, il peut s'avérer judicieux d'introduire une métrique pour affiner ou choisir une de ces définitions telle que la fiabilité, la couverture, la robustesse, etc.

Ce que l'on peut également constater c'est que la définition même de la durée de vie va dépendre de l'application dévolue au réseau de capteurs.

L'emplacement des nœuds défaillants est également important. Si la proportion de nœuds qui ont manqué d'énergie est située dans une certaine partie critique du réseau, par exemple, reliant le nœud central (Station de Base) et le reste du réseau, cela peut avoir comme conséquence le dysfonctionnement précoce du réseau entier.

Il convient de noter que la simulation de la durée de vie du réseau peut être un problème statistique difficile. De toute évidence, plus ces durées sont longues, meilleur est le

fonctionnement du réseau. De manière plus générale, il est possible de chercher à estimer le complémentaire de la fonction de répartition des durées de vie des nœuds (avec la probabilité qu'un nœud survive un temps donné), ou la survie relative d'un réseau (à quel moment tel pourcentage de nœuds est encore opérationnel). Notons toutefois que des corrélations vont s'instaurer entre les consommations des différents nœuds, ces corrélations pouvant être positives et liées à une densité importante d'événements dans une partie de la zone de surveillance ou du phénomène d'*overhearing* ou bien négatives en raison du routage.

Toutes ces métriques peuvent bien sûr être évaluées avec un ensemble d'hypothèses sur les caractéristiques d'un nœud donné en terme de consommation d'énergie, sur la « charge » courante que le réseau est appelé à traiter (par exemple, où et quand les événements se produisent) et aussi sur le comportement du canal radio.

3. Conservation d'énergie

Des mesures expérimentales ont montré que, généralement, c'est la transmission des données qui est la plus consommatrice en énergie, et de façon significative, les calculs, eux, consomment très peu [4]. La consommation d'énergie du module de détection dépend de la spécificité du capteur. Dans de nombreux cas, elle est négligeable par rapport à l'énergie consommée par le module de traitement et, par dessus tout, le module de communication.

Dans d'autres cas, l'énergie dépensée pour la détection peut être comparable, ou supérieure à celle nécessaire à la transmission de données. En général, les techniques d'économie d'énergie se concentrent sur deux parties : la partie réseau (i.e., la gestion d'énergie est prise en compte dans les opérations de chaque nœud, ainsi que dans la conception de protocoles réseau), et la partie détection (i.e., des techniques sont utilisées pour réduire le nombre ou la fréquence de l'échantillonnage coûteux en énergie).

La durée de vie d'un réseau de capteurs peut être prolongée par l'application conjointe de différentes techniques. Par exemple, les protocoles efficaces en énergie visent à réduire au minimum la consommation d'énergie pendant l'activité du réseau. Toutefois, une quantité considérable d'énergie est consommée par les composants d'un nœud (CPU, radio, etc), même s'ils sont inactifs. Un plan de gestion dédié à l'énergie peut alors être utilisé pour éteindre temporairement les composants du nœud lorsqu'ils ne sont pas sollicités.

4. Techniques de *Duty-cycling :*

Cette technique est généralement utilisée dans l'activité réseau. Le moyen le plus efficace pour conserver l'énergie est de mettre la radio de l'émetteur en mode veille (*low-power*) à chaque fois que la communication n'est pas nécessaire. Idéalement la radio doit être éteinte dès qu'il n'y a plus de données à envoyer et ou à recevoir, et devrait être prête dès qu'un nouveau paquet de données doit être envoyé ou reçu. Ainsi, les nœuds alternent entre périodes actives et sommeil en fonction de l'activité du réseau. Ce comportement est généralement

dénommé *Duty-Cycling*. Un *Duty-Cycle* est défini comme étant la fraction de temps où les nœuds sont actifs.

Comme les nœuds-capteurs effectuent des tâches en coopération, ils doivent coordonner leurs dates de sommeil et de réveil. Un algorithme d'ordonnancement Sommeil/Réveil accompagne donc tout plan de *Duty-cycling*. Il s'agit généralement d'un algorithme distribué reposant sur les dates auxquelles des nœuds décident de passer entre l'état actif et l'état sommeil. Il permet aux nœuds voisins d'être actif au même temps, ce qui rend possible l'échange de paquets, même si les nœuds ont un faible *duty-cycle* (i.e., ils dorment la plupart du temps).

4.1 Protocoles *Sleep/Wakeup :*

Comme mentionner précédemment, un régime *Sleep/Wakeup* peut être défini pour un composant donné (i.e le module Radio) du nœud-capteur. On peut relever les principaux plans *Sleep/Wakeup* implantés sous forme de protocoles indépendants au-dessus du protocole MAC (i.e. au niveau de la couche réseau ou de la couche application). Dans le document [33], les protocoles *sleep/wakeup* sont divisés en trois grandes catégories : *à la demande*, *rendez-vous programmé* et *régimes asynchrones*.

- ***Les protocoles à la demande :*** utilisent l'approche la plus intuitive pour la gestion d'énergie. L'idée de base est qu'un nœud devrait se réveiller seulement quand un nœud veut communiquer avec lui. Le problème principal associé aux régimes à la demande est de savoir comment informer un nœud en sommeil qu'un autre nœud est disposé à communiquer avec lui. A cet effet, ces systèmes utilisent généralement plusieurs radios avec différents compromis entre énergie et performances (i.e une radio à faible débit et à faible consommation pour la signalisation, et une radio à "haut" débit mais à plus forte consommation pour la communication de données). Le protocole STEM *(Sparse Topology and Energy Management)* [20], par exemple utilise deux radios.

- ***L'approche de rendez-vous programmés :*** L'idée est que chaque nœud doit se réveiller en même temps que ses voisins. Typiquement, les nœuds se réveillent suivant un ordonnancement de réveil et restent actifs pendant un cours intervalle de temps pour communiquer avec leurs voisins. Ensuite, ils se rendorment jusqu'au prochain rendez-vous.

- ***Protocole Sleep/Wakeup :*** protocole *sleep/wakeup* asynchrone peut être utilisé. Avec les protocoles asynchrones, un nœud peut se réveiller quand il veut et tant qu'il est capable de communiquer avec ses voisins. Ce but est atteint par des propriétés impliquées dans le régime *sleep/wakeup*, aucun échange d'informations n'est alors nécessaire entre les nœuds.

4.2 Protocoles du niveau MAC :

Plusieurs protocoles MAC pour les réseaux de capteurs sans fil ont été proposés, et de nombreux état de l'art [3][21] sur les protocoles MAC sont disponibles dans la littérature. Nous avons recensés les protocoles MAC les plus communs en les classant en trois catégories : protocoles basés sur un séquencement temporel, protocoles basés sur la contention et protocoles hybrides.

4.2.1 Protocoles basés sur un séquencement temporel :

Les protocoles MAC appliquant un séquencement temporel découpent le temps en trames (périodiquement) et chaque trame se compose d'un certain nombre de *slots*. A chaque nœud est attribué un ou plusieurs slots par trame, selon un certain algorithme d'ordonnancement. Il utilise ces slots pour l'émission/réception de paquets de/vers d'autres nœuds.

Ces protocoles sont généralement économes en énergie car ils évitent les collisions, ils limitent le *idle listening* et le *overhearing* et mettent les nœuds en mode sommeil durant les slots de temps réservés aux autres nœuds. De plus, ils garantissent un délai borné de bout-en-bout si un algorithme d'allocation prend en compte les caractéristiques du trafic. En revanche, l'aspect centralisé et le besoin d'une synchronisation rendent ce type de protocoles rigide et non-adapté pour les topologies dynamiques et mobiles.

Par la suite, nous décrivons brièvement les protocoles TRAMA, FLAMA et LEACH.

TRAMA TRAMA *(TRaffic-Adaptive Medium Access control)* [20] divise le temps en slots de temps et se base sur le trafic annoncé par les nœuds pour désigner les émetteurs et les récepteurs pour chaque slot de temps. Pour ce faire, TRAMA applique trois mécanismes :

- Un protocole de voisinage appelé NP *(Neighbor Protocol)* pour échanger la liste des voisins à un saut entre les nœuds, ce qui permet d'établir une connaissance des voisins à deux sauts,
- Un protocole d'échange de calendrier appelé SEP *(Schedule Exchange Protocol)* où chaque nœud annonce ce qu'il a comme trafic à envoyer en précisant les récepteurs concernés
- Un protocole d'élection adaptative AEA *(Adaptative Election Algorithm)* qui choisit les émetteurs et récepteurs pour un slot de temps donné en fonction des informations collectées par NP et SEP.

TRAMA découpe le temps en alternant des intervalles de temps à accès planifié et des intervalles de temps à accès aléatoire. Chaque intervalle est constitué de slots de temps. TRAMA commence par des intervalles à accès aléatoire pour permettre au réseau de s'établir. Les échanges pour la découverte de voisinage du protocole NP sont effectués durant les

intervalles à accès aléatoire. Ces intervalles servent aussi pour permette aux nouveaux nœuds de rejoindre le réseau. L'envoi de trames de données se fait durant les intervalles à accès planifié. La figure 2.1 montre comment TRAMA alterne les deux types d'intervalles de temps.

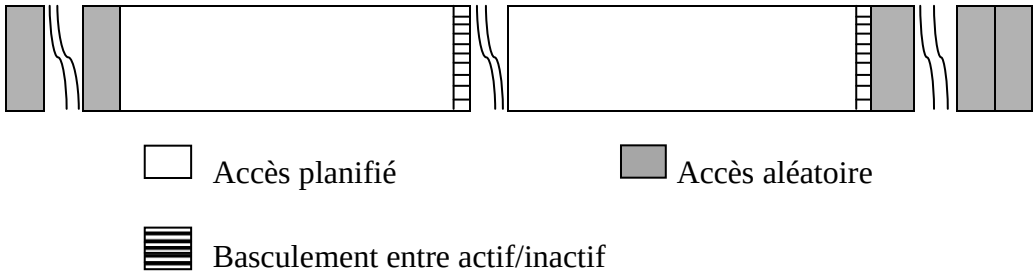


Fig. 2.1 Découpage temporel de TRAMA pour un nœud du réseau.

Pour réaliser son découpage, TRAMA a besoin que chaque nœud diffuse son calendrier de trafic et fasse une découverte de voisinage à deux sauts.

TRAMA se présente comme le premier protocole à introduire l'aspect économie d'énergie dans son mécanisme de découpage temporel en mettant en mode sommeil les nœuds non concernés par une transmission ou une réception durant un slot de temps donné. TRAMA améliore ainsi le débit utile en évitant les collisions des trames de données. En revanche, le découpage temporel séquentiel induit un délai élevé comparé aux protocoles à accès aléatoire. La découverte de voisinage se fait en accès aléatoire et génère par conséquence des collisions, de plus elle nécessite l'échange de trames de taille relativement grande par rapport à la taille des données applicatives dans les réseaux de capteurs.

FLAMA FLAMA *(FLow-Aware Medium Access)* [20] est proposé comme une amélioration de TRAMA. Comme TRAMA, FLAMA divise le temps selon deux modes d'activité : intervalle de temps à accès planifié et intervalle de temps à accès aléatoire. FLAMA a aussi besoin de connaître le voisinage d'un nœud à deux sauts et des informations concernant le flux de données des voisins à un saut.

En revanche, FLAMA ne diffuse pas de calendrier de trafic durant les intervalles de temps à accès planifié mais échange durant les intervalles de temps à accès aléatoire des informations par apport aux flux de données liés à l'application. En outre, FLAMA établit une synchronisation globale du réseau basé sur l'estampillage des trames et une topologie arborescente pour le relais de données.

Pour déterminer les émetteurs et les récepteurs concernés par chaque slot de temps lors de l'intervalle à accès planifié, FLAMA applique un algorithme plus simple que celui utilisé

dans TRAMA. Comme TRAMA, FLAMA assure des transmissions sans collision en ne permettant qu'à un seul nœud d'émettre dans un voisinage à deux sauts.

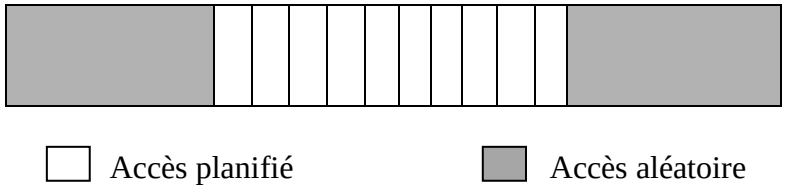


Fig.2.2 Découpage temporel FLAMA

FLAMA est plus performant que TRAMA en termes d'économie d'énergie et perte de trames. De plus FLAMA améliore le délai de bout en bout par rapport à TRAMA grâce à l'exploitation des routes définies à partir d'un arbre pour acheminement multi-sauts. En revanche, FLAMA surcharge le réseau par des échanges pour connaître le flux de l'application et établir une synchronisation globale de réseau de proche en proche.

LEACH LEACH **(***Low Energy Clustering Hierarchy)* [3] à été proposé comme un protocole de routage efficace en consommation d'énergie pour les réseaux de capteurs sans fil, il peut étendre la vie du réseau de facteur 8, comparé aux autres protocoles basés sur le routage multi-sauts ou à regroupement statique.
5% des nœuds sont élus pour être des chefs de clusters. Election périodique des chefs de clusters en fonction du % de nombre de cluster, nombre de tour depuis lequel le nœud n'a pas été chef de cluster. TDMA est utilisé dans le cluster et CDMA entre les clusters. Agrégation des données générées dans un cluster limite la redondance et la consommation énergétique. Suppose que les chefs de clusters soient capables d'atteindre la station de base qui récolte tout [20].

Dans le protocole LEACH [1], tous les nœuds sont considéré statiques, homogènes, et régis par la contrainte de consommation d'énergie. Ces nœuds sont prévus pour capter leur environnement d'une manière continue, et ont, donc, un taux de données fixe à transmettre, cette hypothèse n'est pas conforme au cas des réseaux de capteurs déployés pour la surveillance des objets mobiles tel que ceux utilisés pour le contrôle de trafic des véhicules.

De plus, les canaux de communication radios sont supposés symétriques, ceci dit, deux nœuds qui communiquent entre eux nécessitent les mêmes quantités d'énergie pour transmettre les messages dans les deux sens.

Les caractéristiques principales de l'algorithme LEACH se résument en : la coordination locale pour l'établissement des groupes et leurs fonctionnement, la rotation aléatoire des chefs de groupes, et la fusion locale des données pour réduire le taux de communications avec la station de base. Ce protocole est organisé en périodes où chacune commence par une phase d'initialisation suivie par une autre phase d'une durée plus longue pour le transfert des données captées.

E-MAC, L-MAC et AI-LMAC E-MAC *(Event MAC)* [20] découpe le temps en slots. Chaque slot est lui-même découpé en trois parties respectivement prévues pour accueillir les requêtes de communication, le trafic de contrôle et le trafic de données. E-MAC définit trois types de nœuds : les nœuds actifs possédant un slot, les nœuds passifs ne possédant pas de slot et ne transmettant qu'après avoir fait une requête pour utiliser le slot d'un voisin durant la partie dédiée aux requêtes de communications du slot du voisin, et les nœuds dormants qui dorment la plus part du temps et choisissent un mode passif ou actif quand ils se réveillent. Chaque nœud diffuse une information concernant les slots utilisés par ses voisins durant la partie trafic de contrôle de son slot. Les nœuds doivent tous se réveiller durant la partie trafic de contrôle des slots de leurs voisins.

L'allocation des slots commence par une station de base qui choisit un slot et annonce ce choix par une diffusion. Ensuite, ses voisins choisissent aléatoirement un slot. En cas de choix d'un même slot les nœuds signalent ce fait durant la partie trafic de contrôle. Les slots sont réutilisés à partir de trois sauts.

L-MAC *(Lightweight-MAC)* [20] adopte le même mécanisme d'allocation de slots et de découpage temporel que E-MAC. En revanche, L-MAC force tous les nœuds à avoir au moins un slot. Ainsi, la partie requête de communication du slot n'est plus présente dans L-MAC et un slot est fractionné en 2 parties seulement.

AI-LMAC *(Adaptive, Information-centric and Lightweight MAC)* [20] est une amélioration de L-MAC qui prend en compte les conditions du trafic applicatif et offre la possibilité aux nœuds d'avoir plusieurs slots. AI-LMAC regroupe les nœuds avec la relation père-fils. Le père surveille les conditions de trafic de ses fils et demande à ses fils de s'allouer plus de slots ou de se désallouerdes slots selon leur charge.

La Fig.2.3 montre le découpage temporel en slots des trois protocoles.

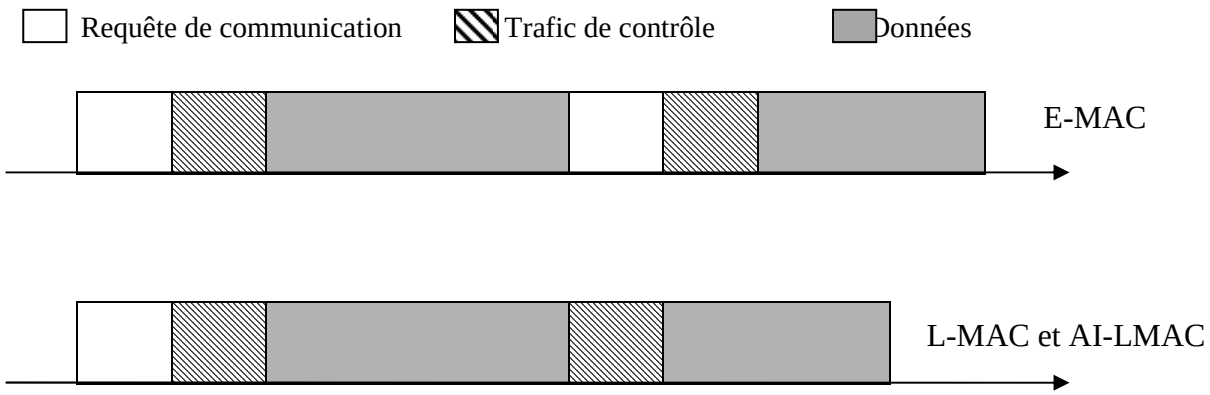


Fig. 2.3 Découpage temporel de E-MAC, L-MAC et AI-LMAC.

Le mécanisme d'allocation de slots peut prendre beaucoup de temps à cause des conflits d'allocation pour servir tous les nœuds du réseau. Dans AI-LAMAC, l'*overhead* généré pour maintenir une mise à jour des informations concernant les conditions de trafic est critique en terme d'utilisation mémoire, d'échange de trames de contrôle, de consommation énergétique et de temps de calcul.

4.2.2 Protocoles basés sur la contention :

Les protocoles avec contention sont les plus populaires et représente la majorité des protocoles MAC proposés pour les réseaux de capteurs sans fil. Ils assurent le *Duty-cycle* par une intégration étroite des fonctionnalités d'accès au canal avec un régime *sleep/wakeup*. La seule différence est que, dans ce cas, l'algorithme *sleep/wakeup* n'est pas un protocole indépendant.

CSMA/CA de la norme 802.11 CSMA/CA est une méthode d'accès de la même famille que CSMA/CD (Carrier Sense Multiple Access with Collision Detection) [20] puisqu'elle impose à un émetteur de s'assurer que le canal est libre avant d'émettre. Dans CSMA/CA, les collisions ne peuvent pas être détectées comme dans le CSMA/CD, un nœud essaie d'éviter les collisions (sans vraiment les éviter à 100 %). Ceci à cause de l'effet d'aveuglement du médium sans fil (Near Far Effect) qui empêche une entité de recevoir quand elle est en train d'émettre.

Deux modes de fonctionnement existent dans la norme IEEE 802.11 [20] [34] : le mode *infrastructure* où les entités communiquent via un nœud central appelé point d'accès, et le mode *ad hoc* où les nœuds peuvent communiquer entre eux, quand la topologie le permet, sans passer par un point d'accès. Il existe aussi deux techniques d'accès au canal : le mode DCF (Distributed Coordination Function) où le CSMA/CA est employé, utilisé dans les architectures de type ad hoc et infrastructure, et le mode PCF (Point Coordination Function) utilisé uniquement dans les architectures de type infrastructure où l'accès au canal est organisé selon une structure temporelle appelée supertrame et délimitée par des *beacons*.

S-MAC, T-MAC et D-MAC S-MAC *(Sensor-MAC)* [20] est multi-sauts célèbre dans les réseaux de capteurs. Il adopte un régime de communication avec planification par rendez-vous. Les nœuds échangent des paquets de synchronisations (leur calendrier de période d'écoute) en le diffusant à leurs voisins à un saut, afin de coordonner leurs périodes *sleep/wakeup*. Ainsi, chaque nœud connaît le calendrier de ses voisins et sait quand il faut se réveiller pour communiquer avec un nœud à sa portée. Chaque nœud peut établir son propre plan ou suivre le plan d'un voisin au moyen d'un algorithme distribué. Les nœuds utilisant le même plan forment un *cluster* virtuel. Un nœud peut éventuellement suivre deux plans s'ils ne se superposent pas, de sorte qu'il puisse faire un pond de communication entre différent *clusters* virtuels.

Les nœuds accèdent au medium en utilisant le **CSMA/CA** de IEEE **802.11** avec le mécanisme RTS/CTS. En outre, un champ supplémentaire est ajouté à tous les messages (y compris les messages RTS/CTS et les acquittements) indiquant la durée de l'échange, ce qui permet aux nœuds non concernés de dormir pendant cette durée.

Le temps d'accès au canal est divisé en deux parties. Dans la période d'écoute, les nœuds échangent des paquets de synchronisation et des paquets de contrôle pour éviter des collisions. Le transfert de données aura lieu dans le reste de la période.

Le fonctionnement de S-MAC est amélioré [20] en minimisant le délai de bout en bout. Pour se faire, ils obligent les nœuds, après avoir reçu RTS ou un CTS qui ne les concerne pas, à se réveiller pour une courte durée après la fin de la transmission pour vérifier s'ils sont la prochaine destination et recevoir la trame si c'est le cas.

La Fig.2.4 montre le séquencement des périodes d'écoute et de sommeil des nœuds avec le découpage en deux parties de la période d'écoute. Les émetteurs B et C, qui souhaitent communiquer avec le récepteur A, connaissent la période d'écoute de A grâce aux messages SYNC envoyés par A.

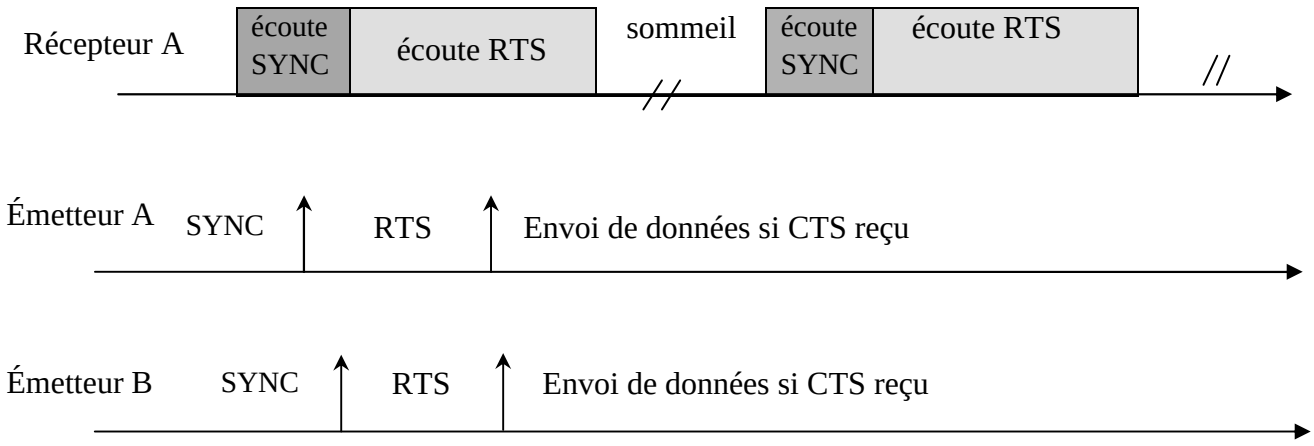


Fig.2.4 Séquencement des périodes d'écoute et de sommeil dans S-MAC

S-MAC apporte une amélioration par rapport au CSMA/CA de la norme 802.11 en terme d'économie d'énergie. Cependant, les messages de synchronisation et les messages RTS/CTS génèrent une surcharge du réseau d'autant plus grande que la longueur des messages de données échangés dans les réseaux de capteurs est relativement courte.

T-MAC *(Timeout-MAC)* est spécialement conçue pour une charge de trafic variable [2]. T-MAC propose de mettre un nœud en mode sommeil après un temps TA durant lequel le nœud n'a reçu aucun message. Ainsi, il réduit l'*idle listening* par rapport à S-MAC. La Fig.2.5

montre le mode de fonctionnement de T-MAC par rapport à celui de S-MAC pour un même nœud.

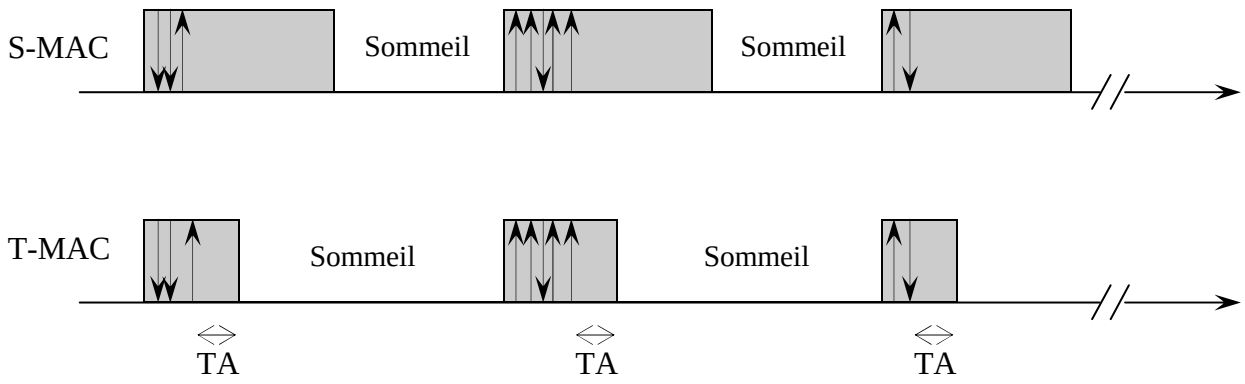


Fig.2.5 Séquencement des périodes d'écoute et de sommeil de T-MAC et de S-MAC

Avec cette approche, T-MAC fait dormir un nœud sans s'être assuré que ses voisins n'ont plus de données à lui envoyer. En effet, les données à envoyer du voisin ont pu être retardées à cause d'un échec d'accès au canal. Ce problème est appelé le sommeil prématuré dans [20]. T-MAC propose d'envoyer un FRTS (*Future Request To Send)* après la réception d'un CTS. Ce FRTS informe la destination qu'elle ne doit pas se mettre en mode sommeil après le TA. Pour éviter une collision entre les données et le FRTS, le nœud qui devait envoyer après le CTS reporte sa transmission pour la durée d'envoi d'un FRTS. Une autre solution est de prendre l'initiative en transmettant un RTS à la réception d'un RTS. Au lieu de répondre par un CTS, le nœud peut décider d'envoyer un RTS pour éviter que le destinataire des messages de ce nœud dorme au bout de TA.

Dans [20], il est mentionner que T-MAC réduit la consommation énergétique jusqu'à 96% sous faible charge par rapport à CSMA/CA de 802.11, et que T-MAC consomme autant que S-MAC sous forte charge. En revanche, T-MAC génère plus de trafic de contrôle que S-MAC.

Dans S-MAC et T-MAC, les nœuds choisissent leur période de réveil soit aléatoirement, soit de manière à ce qu'elle coïncide avec celle d'un voisin. Cela induit des retards sur le délai de bout-en-bout.

D-MAC [21] *(Data gathering MAC)* propose un séquencement des périodes d'activité qui favorise la collecte d'information dans une topologie arborescente. Les nœuds de même niveau se réveillent au même temps. Les fils d'un nœud accèdent au medium au même temps en utilisant des délais (appelé *backoff*) aléatoire pour éviter des collisions systématiques. De ce fait, D-MAC sous-estime la probabilité de collisions en supposant que les nœuds fils peuvent se détecter. D-MAC suppose aussi que la durée nécessaire pour envoyer le trafic cumulé dans les nœuds les plus proches de la racine reste inférieure à la durée nécessaire pour

envoyer le trafic cumulé dans les nœuds les plus proches de la racine reste inférieure à la durée du cycle d'activité.

LPL : B-MAC, WiseMAC, B-MAC+, X-MAC et DW-LPL le LPL *(Low Power Listening)* est l'une des premières approches pour réduire l'*idle listening* en introduisant une période d'inactivité au niveau de la couche physique. L'idée est de pénaliser les émetteurs en envoyant un préambule long pour économiser l'énergie des récepteurs. Les récepteurs activent leurs modules radio périodiquement pour détecter la présence d'un préambule. La durée de transmission du préambule doit être de la même longueur que l'intervalle entre deux réveils d'un récepteur. La Fig.2.6 montre comment le nœud B s'active de temps en temps et détecte un préambule envoyé par A. LPL est un mécanisme qui peut être ajouté à la plupart des protocoles MAC.

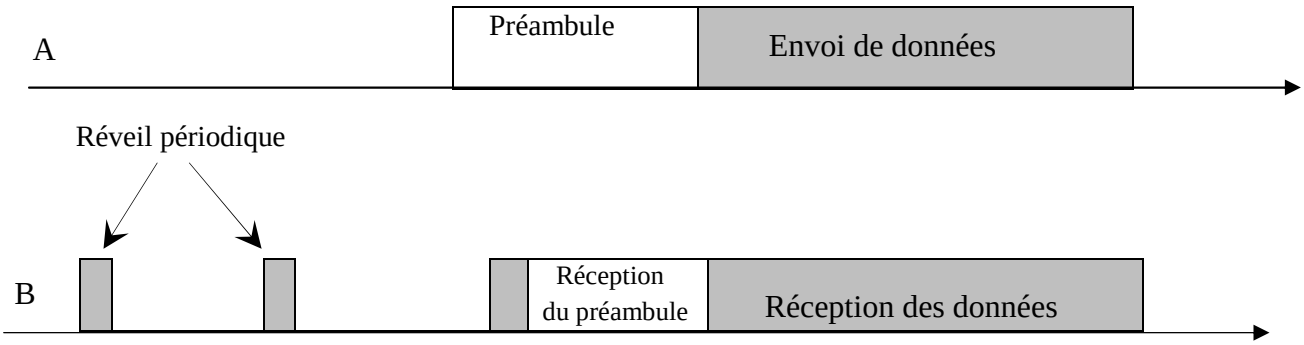


Fig. 2.6 LPL : l'émetteur utilise un long préambule pour permettre au récepteur d'activer son module radio seulement de temps en temps.

L'un des premiers protocoles MAC pour les réseaux de capteurs sans fil basés sur cette technique est B-MAC *(Berkeley-MAC)* [21]. Dans [35], des résultats de simulations ont permis de constater que la technique de LPL réduit l'*idle listening* et améliore les performances en terme d'économie d'énergie de S-MAC et T-MAC.

WiseMAC [21] a été proposé pour réduire la longueur du préambule en fonction des périodes de réveil des récepteurs voisins. De cette façon, WiseMAC réduit l'*overhead* du long préambule et l'*overhearing* engendré par ce préambule, étant donné que tous les nœuds doivent rester éveillés à la réception d'un préambule pour attendre la trame de données, afin de savoir s'ils sont destinataires du message ou non.

B-MAC+ [20] améliore B-MAC en remplaçant le long préambule par plusieurs petits messages appelés les paquets de *countdown* contenant chaqu'un l'identifiant du destinataire et le temps restant pour commencer l'envoi des données. Cela permet aux nœuds non concernés de se mettre en mode sommeil en détectant un petit préambule au lieu de rester éveillé pour la durée de la transmission du long préambule traditionnel.

X-MAC [20] décompose lui aussi le long préambule en plusieurs préambules de petite taille incluant l'identifiant du destinataire du message. Contrairement B-MAC+, le destinataire prévient l'émetteur de son écoute avec un acquittement envoyé entre deux préambules consécutifs. L'émetteur peut alors commencer la transmission des données. Cela réduit l'*overhead* au niveau de l'émetteur qui arrête la transmission du préambule dès la réception de l'acquittement.

DW-LPL *(Dual Wake-up LPL)* [21] limite le problème de l'*overhearing* en employant la technique du LPL pour les messages diffusés uniquement. L'envoi de messages unicast utilise une technique de signalement par des trames spécifiques envoyées par les récepteurs pour informer leur entourage qu'ils sont prêts à recevoir des trames en unicast. Les instants et la périodicité d'envoi des messages de signalement sont indépendants d'un nœud à l'autre. Ceci, en contre partie, génère des collisions dues au problème du terminal caché.

IEEE 802.15.4 [5] Le protocole IEEE 802.15.4, qui constitue la couche MAC et la couche physique de ZigBee, est un protocole adapté aux réseaux de capteurs à cause de sa faible consommation énergétique. La particularité de ce standard est sa couche MAC qui offre une *Supertrame* capable de mettre le module de transmission en veille en fonction d'un schéma prédéterminé selon le mode utilisé. En effet, ce standard qui a un débit de 250 kb/s spécifie deux modes d'accès au médium : un mode non synchronisé (CSMA/CA classique) et un mode synchronisé où des nœuds dans une topologie étoile se synchronisent avec un coordinateur. Le protocole IEEE 802.15.4 offre deux types de dispositifs réseau : les *FFDs (Full Function Device)* ayant toutes les fonctions possibles et les *RFDs (Reduced Function Device)* qui sont de simples dispositifs avec des ressources modestes. Les *RFDs* constituent en général les feuilles du réseau car leurs ressources sont limitées, tandis qu'un *FDD* est une unité de traitement sans contrainte énergétique avec plusieurs fonctionnalités dont le routage.

Ces caractéristiques sont en accord avec la topologie en étoile qu'offre 802.15.4 où tous les nœuds communiquent avec un nœud central appelé coordinateur *PAN*. Dans cette topologie, il n'y a pas de communication entre RFDs. L'autre topologie qu'offre 802.15.4 est une architecture *point à point* où il n'y a aucune hiérarchie entre les nœuds (Fig.2.7).

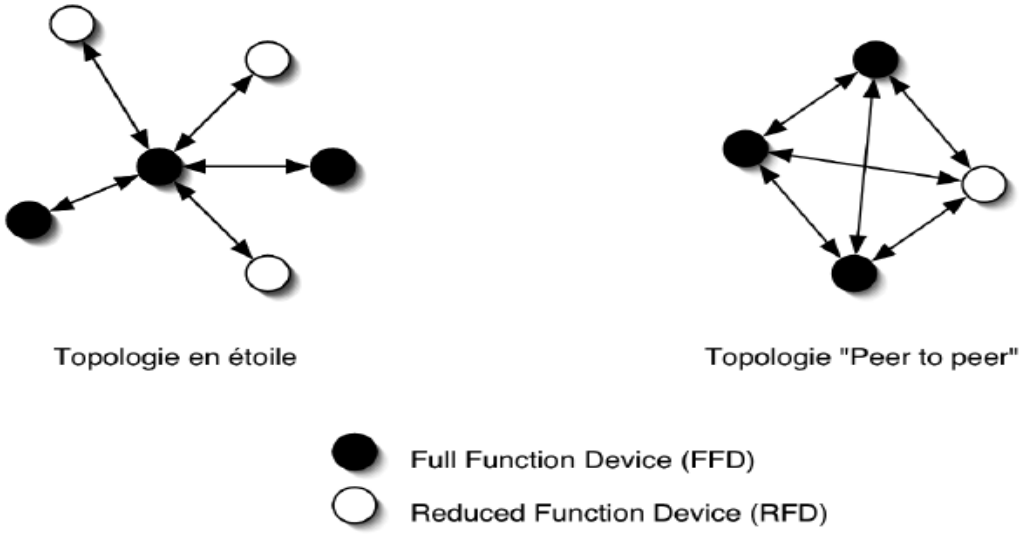


Fig. 2.7 Topologies proposées par le standard IEEE 802.15.4 [5]

Nous allons nous focaliser sur le mode synchronisé qui offre de meilleures performances au niveau de la gestion énergétique. Dans ce mode, les communications sont garanties par un espace temporel appelé *Supertrame* (Fig. 2.8). La *Supertrame* est composée de deux principales portions : une portion active et une portion inactive. Les nœuds interagissent pendant la période active qui est divisée en 16 slots de même durée et entrent en somnolence pendant la période inactive pour économiser de l'énergie.

La portion active se compose de deux périodes : une période avec contention (noté *CAP*) et une période sans contention optionnelle (noté *CFP*). La méthode d'accès au médium avec contention correspond au protocole CSMA/CA classique tandis que dans la méthode d'accès sans contention, qui est optionnelle, les accès sont contrôlés par le coordinateur.

Aussi, cette dernière méthode offre la possibilité aux nœuds du réseau de demander un ou plusieurs slots au coordinateur. Ces slots temporels dédiés, qui sont alloués par le coordinateur lorsque le réseau le permet, porte le nom de *Guaranteed Time Slots (GTS)*. Cette période constitue les derniers slots de la *supertrame* pendant que la période avec contention occupe les premiers slots.

Structure d'une supertrame de IEEE 802.15.4 [20]:

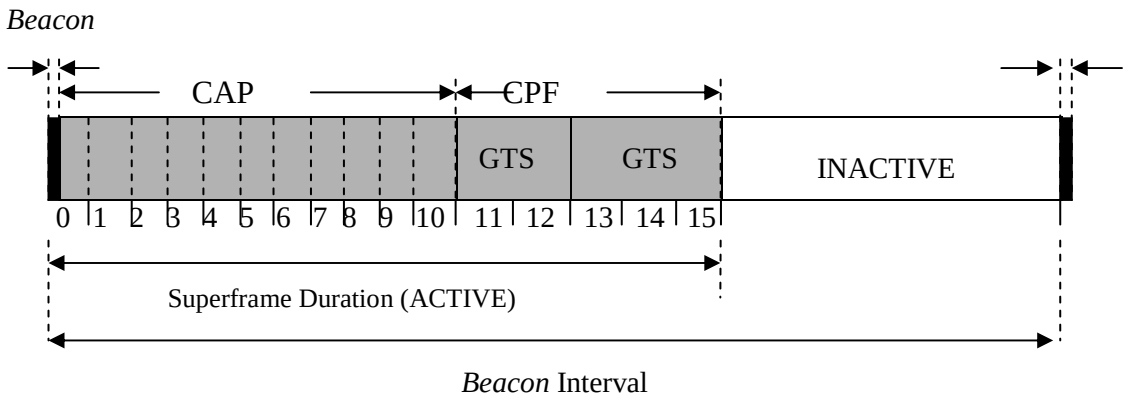


Fig. 2.8 Structure d'une supertrame de IEEE 802.15.4

Dans le mode non-beacon enabled, il n'y a pas de structure en supertrame, i.e. les nœuds sont toujours à l'état actif et utilisent l'algorithme Unslotted CSMA/CA pour l'accès au canal et la transmission de données. Dans ce cas, la conservation d'énergie a lieu au niveau des couches supérieures.

Les protocoles fondés sur la contention sont robustes et garantissent le passage à l'échelle. En outre, ils ont généralement un délai plus faible que ceux reposant sur TDMA et ils peuvent facilement s'adapter aux conditions de trafic. Malheureusement, leur dissipation d'énergie est plus élevée que celle des protocoles TDMA à cause de la contention et des collisions. Des mécanismes *Duty-cycle* peuvent contribuer à réduire la surconsommation d'énergie, mais ils doivent être conçus avec soin pour être flexibles et à faible latence.

4.3. Protocoles hybrides :

L'idée de base des protocoles hybrides (changement du comportement du protocole entre TDMA et CSMA en fonction du niveau de contention) n'est pas nouvelle. Ainsi, ces protocoles essaient d'avoir les avantages des deux méthodes en alternant les deux dans le temps ou en les combinant d'une manière intelligente. Dans la suite, nous allons décrire trois protocoles hybrides : Z-MAC, G-MAC et Funneling-MAC.

Z-MAC Z-MAC [20] *(Zebra-MAC)* est un protocole hybride qui alterne des périodes de TDMA et CSMA/CA selon le nombre d'entités en concurrence en un instant donné. Z-MAC utilise un découpage temporel basé sur TDMA et gère les accès durant les slots avec le CSMA/CA de IEEE 802.11.

Une fois le réseau déployé, Z-MAC commence par une phase de découverte du voisinage à deux sauts suivie par une assignation de slots aux nœuds en utilisant DRAND (*Distributed Randomized TDMA Scheduling For Wireless Adhoc Networks)*. DRAND est un protocole distribué qui assure qu'un slot de temps n'est pas assigné à deux nœuds situés à moins de trois

sauts l'un de l'autre. La synchronisation nécessaire est obtenue à l'aide de deux protocoles utilisés de façon complémentaire.

Pour accéder au médium, si le nœud est le propriétaire du slot courant, il attend un temps aléatoire plus petit qu'une valeur T_0 puis effectue un CCA. Si le canal est libre, il émet. Sinon, il attend que le canal devienne libre et il recommence la même démarche. Si le slot actuel appartient à un voisin à deux sauts et si le nœud a reçu une indication de forte contention d'un de ses voisins à deux sauts, le nœud n'a pas le droit d'utiliser ce slot. Sinon, il attend un temps aléatoire compris entre T_0 et T_{n0} avant d'effectuer un CCA.

La Fig.2.9 montre le découpage temporel pour une topologie simpliste représentée par quatre nœuds : A, B, C et D. Notons que D est à 3 sauts de A. Ceci permet à A et D de partager le même intervalle de temps.

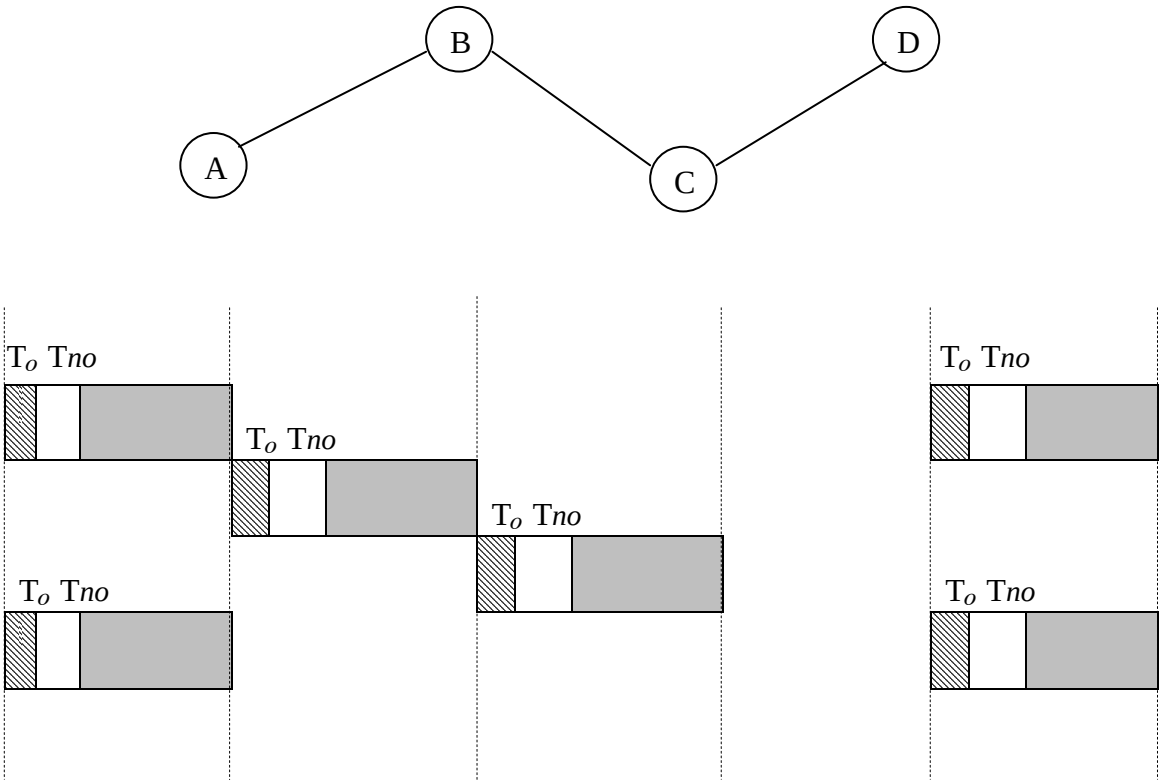


Fig. 2.9 Découpage temporel de Z-MAC. Notons l'existence d'un temps pendant lequel le medium n'est pas utilisé

Z-MAC utilise la techniques LPL. Comme les nœuds peuvent émettre durant tous les intervalles, les nœuds écoutent le médium périodiquement pour vérifier s'il y'a des émissions qui les concernent.

Z-MAC est plus performant que B-MAC sous forte et moyenne charge, et moins performant sous faible charge, surtout en terme d'économie d'énergie.

Les nœuds peuvent être soit en mode faible niveau de contention (LCL pour *Low Contention Level*), soit en mode haut niveau de contention (HCL pour *High Contention Level*). Un nœud persiste dans le mode LCL sauf s'il a reçu une notification (ECN pour *Explicit Contention Notification*). Dans le mode HCL, seuls les propriétaires du slot et leurs voisins à deux sauts sont autorisés à concourir pour l'accès au canal. En LCL (à la fois les propriétaires et les non-propriétaires) peuvent concourir pour transmettre dans n'importe lequel des slots. En revanche les propriétaires ont une priorité sur les autres. De cette façon, Z-MAC peut atteindre un niveau élevé d'utilisation du canal, même en faible contention, car un nœud peut transmettre dès que le canal est disponible [2].

Z-MAC est fait pour optimiser l'utilisation du canal en se comportant comme CSMA/CA sous faible charge et comme TDMA sous forte charge. Le CSMA/CA utilisé par Z-MAC n'est pas économe en énergie, car chaque nœud écoute le médium tant qu'il est détecté occupé. Z-MAC n'emploie pas un mécanisme de report afin de ne pas déborder sur le slot suivant et ne garantit pas qu'un seul nœud soit propriétaire d'un slot. Cela est dû aux imprécisions de synchronisation et aux erreurs potentielles d'assignation de slots. Une réutilisation de slots à partir de deux sauts génère des collisions à cause des acquittements de niveau MAC.

G-MAC G-MAC *(Gateway MAC)* [20] organise les échanges à l'intérieur d'un *cluster*. Dans ce domaine un *cluster* est un ensemble de nœuds géré par une station appelée *passerelle.* G-MAC découpe le temps en deux périodes : une période de collecte où les échanges se font en CSMA/CA, et une période de distribution où les échanges se font en TDMA.

Durant la période de collecte, les nœuds envoient à leur passerelle deux types de trafic : les requêtes FRTS *(Future Request To Send)* pour réserver un slot de temps dans la période de distribution pour échanger avec d'autres nœuds appartenant au même *cluster*, et le trafic *inter-cluster* destiné aux nœuds appartenant à des *clusters* différents. Durant un slot, les échanges se font selon un séquencent RTS-CTS-données-acquittement. La période de dissipation commence par une diffusion d'un message GTIM *(Gateway Traffic Indication Message)* qui contient des indications temporelles des deux périodes suivantes ainsi que le séquencement des échanges entre les nœuds du *cluster* qui ont réussi à envoyer une requête FRTS.

Pendant la période de collecte et pendant les slots non réservés à la période de distribution la passerelle échange les données inter-*cluster* avec d'autres passerelles une fois la collecte terminé.

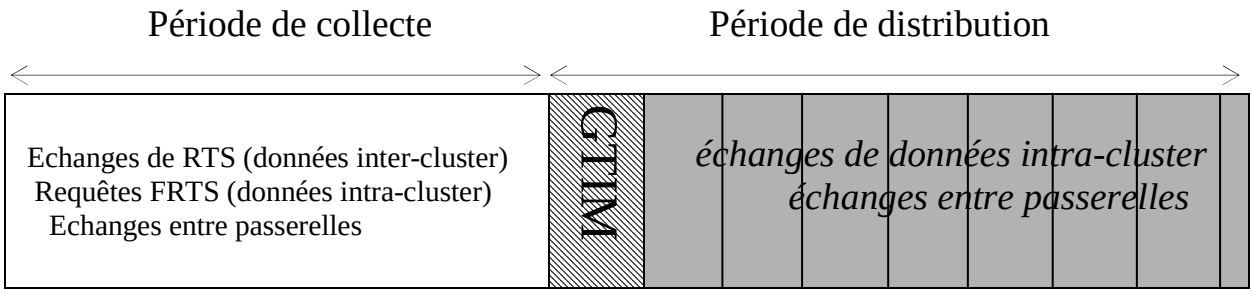


Fig. 2.10 Découpage temporel de G-MAC

En comparant [20] G-MAC à 802.11, B-MAC, S-MAC et T-MAC. G-MAC double la durée de vie dans le pire de *scenarii*. En revanche, dans cette évaluation la taille du réseau est limitée à taille d'un *cluster* et donc à la portée de la passerelle, mais suppose au même temps l'existence de plusieurs *clusters* et le fait que les passerelles échangent entre elles les messages inter-*clusters.* Néanmoins, si les *clusters* sont à portés les uns des autres cela génère des conflits de réservation des slot TDMA dans la période de distribution.

Funneling-MAC Funneling-MAC [21] est un protocole qui prend en compte le goulot d'étranglement dont souffrent la pluparts des applications des réseaux de capteurs. Ce phénomène survient quand une station du réseau joue le rôle d'un puits de données vers lequel un ensemble de capteurs dirige son trafic. Cela est présenté sur la Fig.2.11 sur laquelle est identifiée une zone à forte charge.

Funneling-MAC adopte une méthode d'accès en CSMA/CA dans l'ensemble du réseau durant un intervalle de temps suivi par un intervalle de temps durant lequel une méthode d'accès en TDMA est utilisée pour la zone à forte charge uniquement pour offrir plus de temps d'accès aux nœuds à proximité du puits.

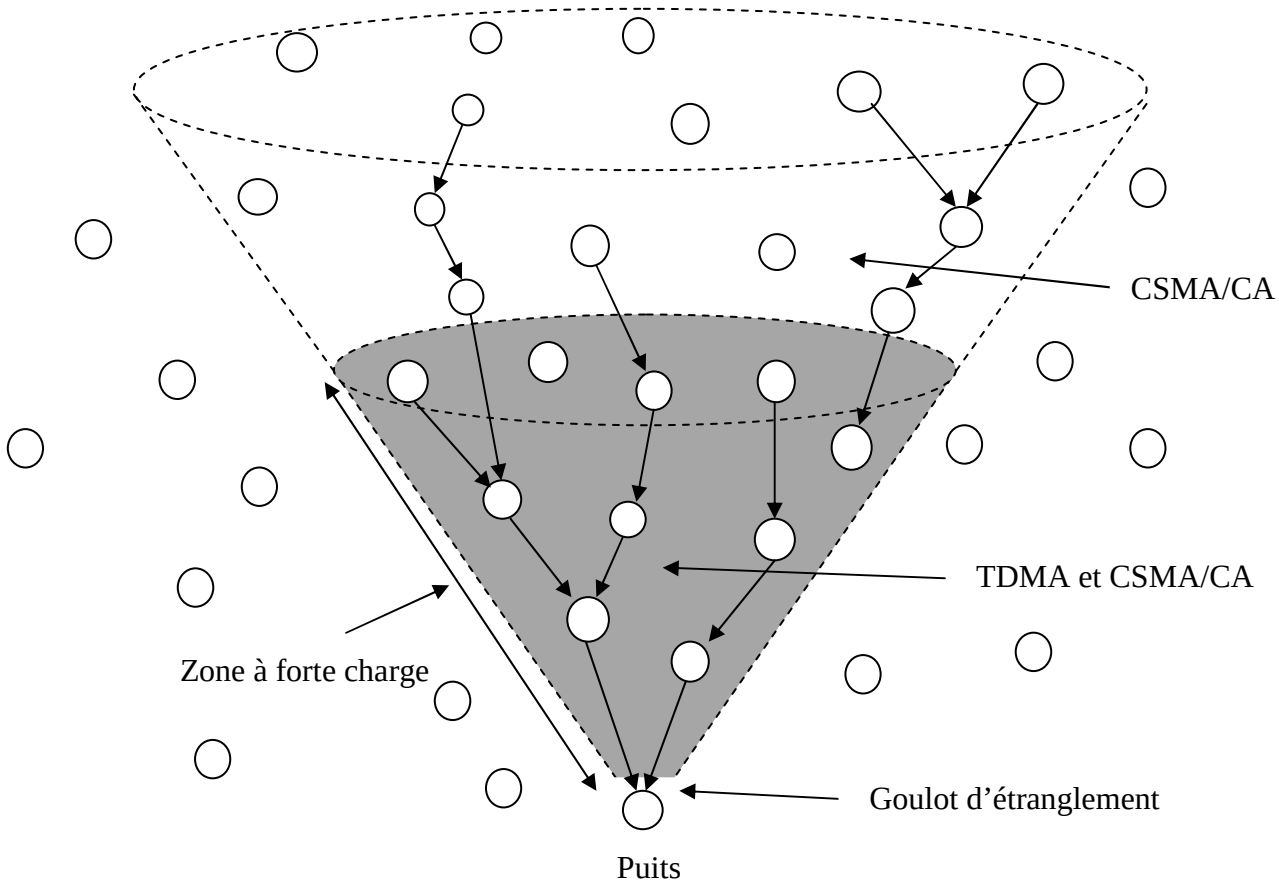


Fig.2.11 Gestion de goulot d'étranglement dans le Funniling-MAC

Ces deux intervalles de temps constituent une supertrame. Ce découpage est représenté sur la Fig.2.12.

La zone à forte charge est dimensionnée par une diffusion d'un *beacon* par le nœud puits. Les nœuds qui ne reçoivent pas ce *beacon* appliquent le CSMA/CA.

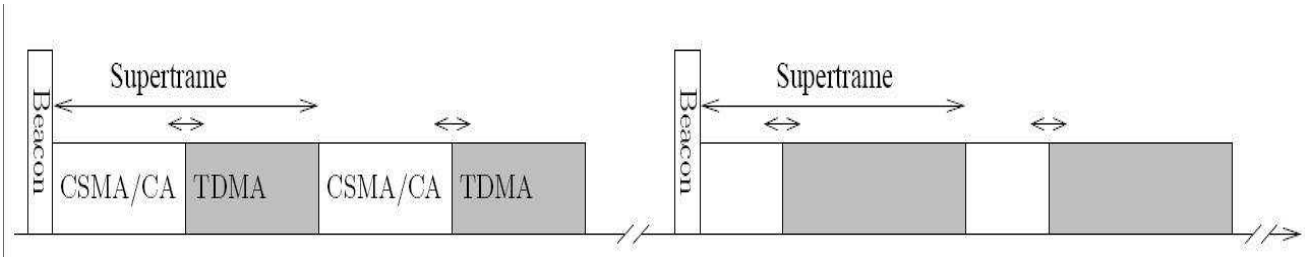


Fig.2.12 Découpage temporel dans Funneling-MAC.

Le puits se base sur le chemin parcouru par les trames qu'il reçoit pour définir le séquencement et le dimensionnement des slots TDMA alloués aux nœuds de la zone à forte charge. Seuls les nœuds appartenant à la zone à forte charge mettent à jour le chemin parcouru par une trame. Pour éviter que les nœuds qui se trouvent au delà de la zone à forte charge interfèrent avec le découpage temporel du TDMA et le *beacon* transmit par le nœud puits, les nœuds de la zone à forte charge génèrent périodiquement une trame contenant les informations concernant la durée de la période CSMA/CA, la durée de la période TDMA et le nombre de supertrames jusqu'au prochain *beacon*.

Funneling-MAC s'avère plus performant que B-MAC et Z-MAC en terme de débit, d'économie d'énergie et de nombre de paquets de contrôle. Funneling-MAC considère que le puits possède plus de capacité de calcul et de ressources énergétiques que les autres nœuds du réseau et qu'il est donc capable d'exécuter les différents algorithmes du protocole.

Les protocoles hybrides tentent de combiner les point forts des protocoles MAC fondés sur TDMA et ceux avec contention tout en compensant leurs faiblesses. Toutefois, ces techniques semblent être complexes pour être réalisables dans un déploiement d'un grand nombre de nœuds.

Récapitulatif des caractéristiques des protocoles présentés :

Le tableau 2.1 récapitule [21], d'une manière globale, l'apport de chacun des protocoles présentés concernant les 2 besoins essentiels des applications industrielles : l'économie d'énergie et la qualité de service.

Nom du protocole	**Economie d'énergie**	**Qualité de service**
Protocoles basés sur TDMA		
TRAMA FLAMA E-MAC L-MAC AI-LMAC	Un nœud économise de l'énergie durant les slots dans lesquels il n'est ni émetteur ni récepteur	Accès garanti suite à une réservation en accès non-garanti
Protocoles basés sur CSMA/CA		
S-MAC	Tous les nœuds peuvent économiser de l'énergie	Aucune
T-MAC	Améliore l'économie d'énergie de S-MAC	Aucune
D-MAC	Economise plus d'énergie que S-MAC	Diminue le délai de bout-en-bout par rapport à S-MAC
Protocoles basés sur LPL		

B-MAC WiseMAC B-MAC+ X-MAC DW-LPL	Améliorent l'économie d'énergie des protocoles CSMA/CA	Aucune
Protocoles Hybrides		
Z-MAC	Plus économe sous forte charge que BMAC	Réduit le déterminisme du TDMA de base pour améliorer l'utilisation du canal
G-MAC	Limitée aux nœuds à portée d'un nœud central	Limitée aux nœuds à portée d'un nœud central
Funneling-MAC	Plus économe sous forte charge que ZMAC	Limitée aux nœuds proches d'un nœud central

Tab.2.1 Récapitulatif des protocoles étudiés en terme d'économie d'énergie et de qualité de service.

Nous pouvons constater qu'aucun protocole ne garantit l'économie d'énergie et la qualité de service en même temps. Il y a toujours un compromis à faire entre ces deux aspects. Un autre compromis est aussi la complexité des algorithmes utilisés pour assurer le bon fonctionnement de ces protocoles, notamment les protocoles ayant des périodes basées sur TDMA, plus les algorithmes sont simples, moins intéressant est leur rendu en terme de performances de réseaux.

5. Techniques orientées données :

Généralement les plans *Duty-cycling* ne tiennent pas compte des données prélevées par les nœuds. Par conséquents, des approches orientées données peuvent être utiles pour améliorer l'efficacité en énergie. En fait, les détections (ou prélèvement de données) affecte la consommation d'énergie de deux manières :

- ***Des échantillons inutiles*** : les données échantillonnées ont souvent de fortes corrélations spatiales et/ou temporelles, il est donc inutiles de communiquer les informations redondantes à la Station de Base. Un échantillonnage inutile implique une consommation d'énergie à son tour inutile. En effet, même si le coût de l'échantillonnage est négligeable, cela induit aussi des communications tout le long du chemin qu'empreinte le message.

- ***La consommation électrique*** du module de détection : réduire la communication ne suffit pas lorsque le capteur est lui-même très consommateur.

Des techniques orientées données sont conçues pour réduire la quantité d'échantillonnage de données en garantissant un niveau de précision acceptable dans la détection pour l'application.

5.1 Réduction des données :

Réduire les données en termes de volume ou de nombre de paquets, dans les réseaux peut avoir un impact majeur sur la consommation d'énergie due à la communication. En effet, il a été montré dans plusieurs publications scientifiques que la transmission d'un bit est équivalente, en termes d'énergie, à l'exécution d'environ 1000 instructions. Cette valeur augmente avec la portée de la radio [23]. Plus le capteur devrai transmettre loin, et par conséquent augmenter sa puissance d'émission, plus il va consommer de l'énergie, et par conséquent réduire sa durée de vie. Il convient donc de réduire les données. Parmi les méthodes de réduction de données, nous trouverons :

In-network processing qui consiste à réaliser de l'agrégation de données au niveau des nœuds intermédiaires entre la source et le *Sink*. Ainsi, la quantité de données est réduite tout en parcourant le réseau vers le *Sink*. Une agrégation appropriée est spécifique à l'application. Par exemple [23], si un réseau est déployé pour mesurer la température et que le *Sink* n'est intéressé que par la moyenne des températures, un nœud intermédiaire pourra additionner les valeurs reçues de ses enfants et envoyer le résultat à son père. Le *Sink* recevra alors qu'un seul message contenant la somme des données au lieu de *n* messages (où *n* est le nombre de capteurs).

Les techniques d'agrégations sont souvent utilisées. Elles sont cependant difficiles à mettre en œuvre lorsque les données sont chiffrées car le traitement des données devient alors très délicat.

La compression de données peut être appliquée également pour réduire la quantité d'informations transmises par les nœuds sources, et les représentées de la manière la plus compacte possible. Ce régime implique l'encodage des informations au niveau des nœuds qui engendrent des données, et le décodage au niveau du *Sink*.
Le but de la compression est d'utiliser le débit de communication le plus faible possible tout en garantissant de pouvoir reconstituer les informations à la Station de Base avec une erreur n'excédant pas une distorsion donnée [23]. Il existe différentes méthodes de compression de données citées dans le chapitre III.

5.2 Acquisition de données efficace en énergie :

De nombreuses applications émergentes ont d'application a de réelles contraintes dues à la détection. Ceci va à l'encontre de l'hypothèse générale selon laquelle la détection n'est pas significative d'un point de vue consommation d'énergie. En fait, la consommation d'énergie du module de détection peut, non seulement être significative, mais encore supérieure à la

consommation d'énergie de la radio ou même plus grande que la consommation d'énergie du reste du nœud-capteur [2]. Cela peut être dû à différents facteurs [24].

Traducteur gourmant en énergie : Certains capteurs ont intrinsèquement besoins d'une forte puissance pour s'acquitter de leur tâche d'échantillonnage. Par exemple, des capteurs d'images CMOS, voire des capteurs multimédias [2] ont également besoins de beaucoup d'énergie. Les capteurs chimiques ou biologiques peuvent aussi être gourmands en énergie.

Convertisseurs A/D gourmands : des capteurs tels que les transducteurs acoustiques et sismiques [2] nécessitent généralement des convertisseurs A/D à haut débit et à grande résolution. La consommation d'électricité des convertisseurs représente la part la plus importante de la consommation d'énergie du sous-système de détection.

Capteur actif : une autre classe de capteurs peut obtenir des données du phénomène perçu par l'utilisation de transducteurs actifs (par exemple, sonar, radar, ou laser). Dans ce cas, les capteurs doivent envoyer un signal de sondage afin d'obtenir des informations sur la grandeur observée.

Temps d'acquisition long : le temps d'acquisition peut être de l'ordre de plusieurs centaines de millisecondes, voire de quelques secondes. Par conséquent, l'énergie consommée par le sous-système de détection peut être élevé, même si la consommation d'énergie du détecteur reste modérée.

Dans ce cas, réduire les communications peut s'avérer insuffisant, mais les stratégies de conservation d'énergie doivent réellement réduire le nombre d'acquisitions (échantillons de données). Il faudrait également préciser que les techniques d'acquisition de données efficaces en énergie ne visent pas exclusivement à réduire la consommation d'énergie du module de détection. En réduisant les données prélevées par des nœuds sources, elles diminuent aussi le nombre de communications. En fait, beaucoup de techniques d'acquisition de données efficaces en énergie ont été conçues pour réduire au minimum l'énergie consommée par la radio, en supposant que la consommation de la radio est négligeable.

La classification des approches d'acquisition de données efficaces en énergie présentée dans [24] est comme suit :

L'échantillonnage adaptatif : comme les échantillons mesurés peuvent être corrélés, les techniques d'*échantillonnage adaptatif* exploitent de telles similitudes pour réduire la quantité de données à acquérir par le transducteur. Par exemple, les données intéressantes peuvent changer lentement en fonction du temps.

Dans ce cas, des corrélations temporelles peuvent être exploitées pour réduire le nombre d'acquisition. Une approche semblable peut être appliquée lorsque le phénomène étudié ne change pas brusquement entre les régions couverte par les nœuds voisins. L'énergie due au prélèvement et à la communication peut être alors réduite en profitant des corrélations

spatiales entre les données prélevées. Clairement, des corrélations temporelles et spatiales peuvent être conjointement exploitées pour réduire sensiblement la quantité de données à acquérir.

L'échantillonnage hiérarchique : suppose que les nœuds sont équipés de sondes (ou détecteurs) de différents types. Alors que chaque sonde est caractérisée par une résolution donnée et sa consommation d'énergie associée, cette technique choisit dynamiquement la classe à activer, afin d'obtenir un compromis entre la précision et l'économie d'énergie.

L'échantillonnage actif : fondé sur un modèle adopte une approche semblable à la précision de données. Un modèle du phénomène mesuré est établit lors des prélèvements de données, de telle sorte que les valeurs futures puissent être prévues avec une certains précision. Cette approche exploite le modèle obtenu pour réduire le nombre d'échantillons de données, et également la quantité de données à transmettre à la Station de Base bien que ce ne soit pas leur objectif principal.

6. Techniques de mobilité :

Dans certains cas où les nœuds sont mobiles, la mobilité peut être utilisée comme outil pour réduire la consommation d'énergie (au-delà du *duty-cycling* et des techniques orientées données). Dans un réseau de capteurs statiques, les paquets provenant des nœuds suivent des chemins multi-sauts vers la station de base. Ainsi, certains chemins peuvent être chargés (sollicités plus que d'autres), et les nœuds proches de la Station de Base relayent plus de paquets [2] et sont plus sujets à l'épuisement prématuré de leurs batteries *(funneling effet)*. Si certains nœuds (éventuellement, la station de base) sont mobiles, le trafic peut être modifié si les nœuds mobiles sont chargés de collecter des données directement à partir de nœuds statiques.

Les nœuds ordinaires attendent le passage d'un dispositif mobile pour lui envoyer leurs messages de telle sorte que la communication ait lieu à proximité (directement ou au plus avec un nombre limité de sauts). Par conséquent, les nœuds ordinaires peuvent économiser de l'énergie parce que la longueur du chemin, la contention et les *overheads* de diffusion sont ainsi réduits. En outre, le dispositif mobile peut visiter le réseau afin de répartir uniformément la consommation d'énergie due à la communication. Lorsque le coût de la mobilité des nœuds de capteurs est prohibitif, l'approche classique consiste à attacher un capteur à des entités qui seront en itinérance dans le champ de détection, comme des autobus ou des animaux.

7. Conclusion :

La durée de vie d'un réseau de capteurs est étroitement liée à la vie nodale. Cette dernière, quant à elle, dépend essentiellement de la consommation d'énergie du nœud. Nous avons présenté dans ce chapitre quelques approches de conservation d'énergie dans les réseaux de capteurs sans fil.

Le premier axe des techniques de conservation d'énergie vise à réduire le *duty-cycle* des nœuds. Cela se traduit par la réduction de la durée de l'activité radio afin d'éviter toute surconsommation d'énergie due à la communication. Dans cette optique, plusieurs méthodes ont vu le jour soit sous forme de protocoles MAC à faible *duty-cycle* ou bien sous forme de protocoles indépendants de niveau supérieur fondés sur des ordonnancements *sleep/wakeup*.

Le second axe s'intéresse à l'acquisition des données. En effet, un point intéressant est que plusieurs solutions protocolaires proposées dans la littérature supposent que la consommation d'énergie de la radio est plus élevée que celle due à l'échantillonnage ou au traitement de données. En revanche, de nombreuses applications réelles ont montré que la consommation d'énergie du détecteur est comparable, voire supérieure à la consommation nécessaire à la radio. En outre, l'échantillonnage peut requérir beaucoup de temps (comparé à la durée nécessaire pour les communications) ce qui se traduit par une consommation d'énergie très élevée. Certains travaux de recherche laissent à penser que la conservation d'énergie centrée sur l'acquisition de données n'a pas encore été pleinement analysée. Cela ouvre la voie au développement de techniques pratiques pour réduire la consommation d'énergie des capteurs.

Dans le dernier axe, nous avons évoqué les méthodes centrées sur la mobilité des nœuds relais ou bien des puits de données. Il existe un intérêt croissant pour ce type d'approches car si certaines applications pratiques envisagent des déploiements moins denses, alors pour des raisons d'efficacité et de robustesse, les protocoles de communication peuvent exploiter de façon appropriée la mobilité des nœuds collecteurs.

Il existe bien évidemment beaucoup d'autres techniques de conservation d'énergie. Par exemple, les paradigmes éminents de l'auto-organisation des systèmes, les mécanismes cross-layers et d'autres protocoles indépendants de niveau réseau ou de niveau application.

Ainsi, nous allons nous focaliser dans le chapitre suivant sur du monitoring centré sur des données. Nous allons étudier une stratégie d'économie d'énergie rattachée aux applications qui se placent dans un contexte sans contraintes temporelles, à savoir : la compression de données.

Chapitre III: Techniques de compression de données dans les réseaux de capteurs

"Rien ne se perd, rien ne se crée, tout se transforme." Anaxagore de Clazomènes

Sommaire

1. Introduction :

La notion de réseau se rattache à la circulation et au partage d'informations par plusieurs entités interconnectées. L'évolution des réseaux tels que Internet, le réseau par excellence, ou encore les réseaux de capteurs, réseau dense par définition, montrent que la quantité de données échangée devient de plus en plus conséquente. Pour augmenter les performances d'un réseau, c'est à dire éviter son encombrement, garantir la rapidité du transfert des données ou permettre des envois simultanés, la transformation des informations est nécessaire d'où l'utilisation de techniques de compression de données. Comme le stipule l'adage "Rien ne se perd, rien ne se crée, tout se transforme", la compression consiste à transformer les données de sorte que la taille physique des blocs d'information soit réduite, ce qui facilite leur stockage ou encore leur transport sur un réseau jusqu'à un destinataire donné. Ce destinataire, une fois la quantité de données compressées reçue, doit les décompresser. Si les données décompressées sont identiques aux données avant compression on parle de compression sans pertes sinon, si les données ont subi des transformations, on parle de compression avec pertes.

La compression de données constitue un nouvel axe de recherche pour les réseaux de capteurs. En effet, comme mentionné dans [36, 37], la principale source de consommation énergétique pour un capteur reste dans la grande majorité des cas le transceiver, c'est pour cette raison qu'une stratégie pour économiser de l'énergie est de l'éteindre temporairement (mode somnolence) en accord avec un schéma de réveil bien précis lié à l'application. Cependant, la mise en veille du transceiver réduit le nombre de communications radio puisque le dispositif est indisponible. On peut alors se demander comment garantir le même taux de données envoyé à la station de base en réduisant le nombre de transmissions ?

Une tentative de réponse à cette question est l'utilisation de la compression de données. En effet, les nœuds de niveau bas doivent économiser de l'énergie et la compression de données peut constituer une bonne alternative. Cependant, parler de compression dans les RdCs suppose que nous ne sommes pas dans le cas d'applications dirigées par évènements où le délai est une contrainte, mais plutôt sur l'observation d'un environnement et la collecte de données sans contraintes temporelles.

Toutefois, les nœuds capteurs sont par nature très limités en capacité de calcul et de mémoire, c'est une conséquence directe des contraintes de consommation d'énergie, de miniaturisation et de coût de fabrication. Et la technique de compressions consomme elles même de l'énergie. Ils ne peuvent donc pas implanter des algorithmes de compression de trop grande complexité.

Dans le travail qui suit, nous allons présenter les principaux algorithmes de compression, leurs limites par rapport aux réseaux de capteurs, les adaptations existantes pour réseaux de capteurs.

Nous nous plaçons dans un contexte bien précis qui est celui d'applications non contraintes par le temps pour la remontée d'informations, car il faut noter que le temps de compression constitue une activité donc un temps supplémentaire qui peut influer sur les performances d'une application temps réel. Dans la prochaine section, nous allons tout d'abord présenter les principaux algorithmes de compressions.

2. Traitement de données dans les réseaux de capteurs

a. Compression locale :

Dans le domaine des réseaux de capteurs sans fil, quelques propositions considèrent la compression au niveau de la source, c'est-à-dire par les algorithmes locaux, sans distribution de la charge de traitement de données avec d'autres nœuds (par exemple S-LZW). La compression va servir à réduire le volume des données que la source aura à transmettre. Le traitement de données à la source est aussi nécessaire pour anticiper des possibles pertes d'information pendant la transmission de paquets jusqu'au nœud puits, ou pour contrôler la quantité de données à envoyer selon les conditions du réseau.

Il existe plusieurs algorithmes de compression dans la littérature. Certains peuvent fournir des taux de compression élevés mais toutefois, ils ne sont pas applicables dans les réseaux de capteurs en raison des limitations de ressources des nœuds de capteurs. Une plate-forme pour évaluer les performances de plusieurs algorithmes traditionnels de compression d'images sur un nœud de capteur à été présenté [1]. Ils ont analysé cinq algorithmes bien connus : JPEG2000, SS, DCT, SPITH et JPEG. Les résultats montrent que pour JPEG2000, DCT, SPITH et JPEG, le coût d'énergie des calculs est supérieur au coût de transmission de l'image non compressée. Les résultats montrent SS est le seul des algorithmes à testé qui amène des économies d'énergie par rapport au cas sans compression. Il amène à une réduction de la consommation d'énergie d'environ 29%. De toute manière le coût d'énergie et d'implantation (quantité de mémoire requise notamment) dépendent des caractéristiques du matériel aussi.

Il est attendu de la compression locale des données plusieurs avantages :

- ***Extension de la durée de vie du nœud source :*** En effet, moins la source aura de données à transmettre, et moins d'énergie elle consommera au niveau du transcepteur radio. Cette affirmation est vraie tant que la complexité de l'algorithme de compression adopté sera suffisamment fiable pour être rentable. Le processus de compression ne doit pas coûter plus cher en termes de consommation d'énergie que le gain qu'il amène sur la communication, sinon la présence d'un processus de compression pourrait diminuer la vie utile du nœud.

- ***Extension de la durée de vie des nœuds intermédiaires :*** pour les mêmes raisons, la réduction de la quantité de donnée à la source sera nécessairement bénéfique pour les nœuds chargés de relayer les paquets entre le nœud source et le puits. Ils recevront moins de paquets de données, donc ils auront moins de paquets et d'acquittement à transmettre.

- ***Contribution à la diminution des congestions du réseau :*** une diminution de la quantité de données circulant sur le réseau va entraîner une diminution des risques de

congestions du réseau, donc une diminution de perte de paquets et des retards de transmission.

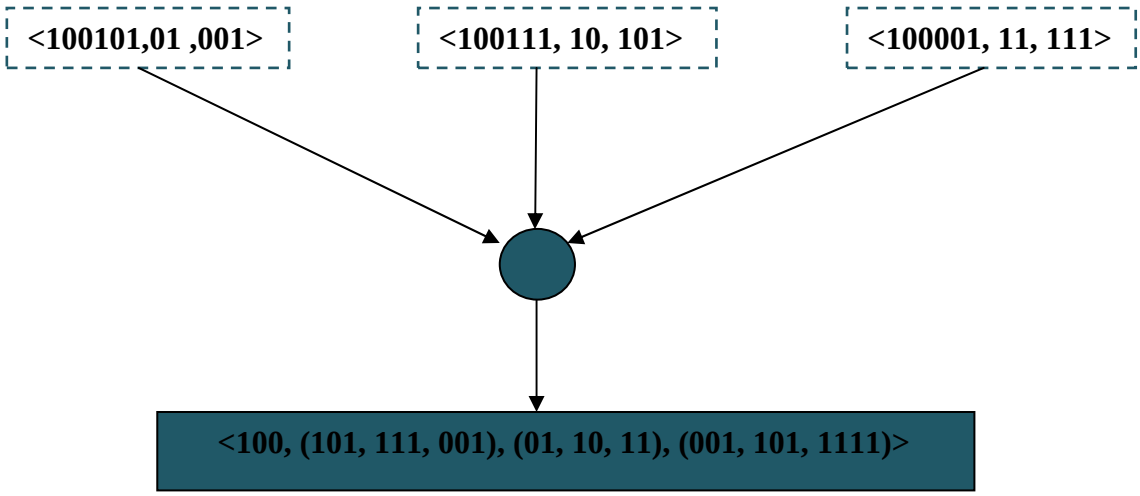
- ***Contribution à la tolérance aux pertes :*** quelques renforcements sur la tolérance aux pertes de paquets peuvent être atteints par l'application de quelques mécanismes de traitement à la source.

b. Compression distribuée :

En raison de la nature distribuée des réseaux de capteurs, il était évident que bon nombre de travaux seraient menés sur la compression distribué. Notons que plusieurs propositions ont été publiées sous la qualification de *compression distribuée*, même s'ils ne réalisent pas vraiment du traitement distribué, mais plutôt du traitement collaboratif (même pas dans plusieurs cas).

Dans la littérature plusieurs techniques d'agrégation des données sont proposées. Cependant la plupart semblent assez lourdes à mettre en œuvre ou bien supposent des connaissances *a priori* sur les données récoltées.

Pour illustrer ce propos, un exemple à été développé dans [38]. Le schéma ci-dessous est un schéma de compression appelé *Pipelined In-Network Compression.*



Cette technique permet de factoriser les parties communes de plusieurs paquets. Son point faible est donc de supprimer les similitudes entre les paquets transmis. Son efficacité dépend de la longueur des parties communes entre paquets. Un autre défaut de cette méthode est qu'il suppose un buffer de grande taille pour stocker les paquets arrivant, avant de tenter de les factoriser. Ce point est problématique car un capteur a une mémoire de capacité relativement limitée.

Le principe est relativement compréhensible, cependant il ne répond pas à toutes les situations que l'on peut rencontrer.

D'un autre coté, l'article [38] propose une méthode qui fournit des résultats impressionnant. Elle repose sur un mélange entre deux traitements distincts des paquets qui peuvent être agrégés sans traitement particulier, ou bien soumis à un algorithme DCS (Distributed Compressed Sampling) qui repose sur des principes mathématiques. Malheureusement toutes les clés de la méthode ne sont pas données, et l'algorithme est présenté seulement dans les grandes lignes.

3. Les algorithmes de compressions :

Avec l'évolution des réseaux, la compression de données devient une technique de plus en plus répondue. Les algorithmes de compression se multiplient et deviennes de plus en plus performants. Le principal objectif d'un algorithme de compression est de réduire la quantité de données pour facilité son transport ou son stockage. Dans le domaine des réseaux de capteurs, elle constitue une stratégie pour économiser l'énergie en réduisant le nombre de transmissions car c'est l'activité la plus coûteuse pour les capteurs. Aussi, en réduisant la quantité d'information, elle facilite son transport pour ces plateformes qui utilisent des technologies sans fil à faible débit.

Les deux principales opérations dans la compression de données sont : la *compression* et la *décompression*. Ces étapes permettent de définir deux catégories de compression : la compression sans pertes et la compression avec pertes. Lorsque l'objet avant compression est identique à celui après décompression, on parle d'un algorithme sans perte sinon lorsqu'il y'a une différence entre les deux objets, on parle d'algorithme avec pertes. L'indicateur de performance d'un algorithme de compression s'appelle le taux de compression. La principale raison de l'utilisation d'algorithmes de compression avec pertes est le souci d'amélioration du taux de compression pour des données dont certaines informations peuvent être perdues sans affecter significativement le contenu, par exemple parce que les bits ne représente que du bruit.

3.1 Les algorithmes avec pertes :

Les algorithmes de compression de données avec pertes ont pour principale objectif d'améliorer le taux de compression en éliminent quelques informations mais en restant le plus proche possible des données originales. Un exemple d'algorithme de compression avec pertes très répondu dédié aux images est JPEG. C'est un algorithme qui est disponible dans la littérature.

Cependant, comme le taux de perte est ajustable, plus la qualité de l'image est élevée, moins l'algorithme est efficace et inversement. Les images faisant partie des données les plus difficiles à transporter sur un réseau à cause de leur taille, des méthodes de compression

permettant de réduire cette taille et donc de faciliter leur transport ont été mise en place. La principale contrainte est l'efficacité de la compression qui se fait au détriment de la qualité de l'image.

La technologie JPEG :

Dans [39], la technologie JPEG est souvent incorporée dans des produit destinés à l'utilisateur final (les éditeurs graphiques notamment), car la méthode de compression est simple à programmer et surtout relativement rapide. Plusieurs paramètres permettent de jouer sur la qualité de l'image et par la même sur le taux de compression. Cette technologie présente un inconvénient : pour un degré de qualité donné, taille d'un fichier JPEG est souvent supérieur à celle que permettent d'obtenir les procédés rivaux.

La compression JPEG :

Dans l'acronyme JPEG, le terme Joint trahit la double origine de cette norme, développée à la fois par les organismes de normalisations *CCITT (Consultatif Comitte Internationnal Telephone Telegraph)* devenu l'*ITU (International Telecommunication Union)* et l'*ISO (Internationnal Standard Organisation).*

JPEG a fourni un format capable de stocker des images en million de couleurs tout en conservant une très bonne efficacité en matière de compression. Et en étant de surcroît raisonnablement rapide.

Caractéristiques de JPEG :

JPEG n'est pas un algorithme de compression banal, c'est plutôt une méthode de compression d'image qui permet d'altérer une image, de produire des images de très petite taille avec une très pauvre qualité ou des images de grande taille avec une excellente qualité. En parlant de grande taille, cette dernière est quand même beaucoup plus petite que la taille originale.

JPEG est également différent dans le sens où c'est une méthode de compression avec perte contrairement à RLE ou LZW. Le grand avantage des systèmes à pertes étant qu'ils peuvent obtenir des taux de compression bien supérieurs à ceux obtenu avec des systèmes sans pertes. JPEG à été conçu spécifiquement pour éliminer des informations que l'œil humain à de la peine à voir.

L'algorithme JPEG :

Les spécifications définissent une suite d'opérations minimale à respecter. C'est un schéma de codage basé sur la *Transformée en Cosinus Discret (DCT).* DCT est par nature avec perte, elle est capable d'atteindre de hauts degrés de compression avec un minimum de pertes si la différence entre deux pixels adjacents est faible.

Le principal de la compression du JPEG est constitué de plusieurs étapes [40]:

- la préparation ;
- la transformation ;
- la quantification ;
- l'encodage.

Prenons l'exemple suivant : une image 640x480 RGB 24 bits par pixel.

- La première étape consiste à transformer le codage de la couleur RGB, de l'image bitmap de départ, en un codage YIQ, avec les combinaisons linéaires:

  Y = 0.299 R + 0.587 G + 0.114 B
  I = 0.596 R - 0.275 G - 0.321 B
  Q = 0.212 R - 0.523 G + 0.311 B

  Mais pour gagner un peu de place, on arrondit généralement les coefficients de la *luminance* Y et de la *chrominance* (I et Q), pour obtenir :

  Y = 0.3 R + 0.59 G + 0.11 B
  I = 0.6 R - 0.28 G - 0.32 B
  Q = 0.21 R - 0.52 G + 0.31 B

  Les résultats sont ensuite regroupés en matrice 640x480. Nous obtenons donc trois matrices de taille 640x480 représentant la *luminance* et la *chrominance* (2 matrices). Une des astuces du code JPEG est d'éliminer les variations de la chrominance entre deux pixels. En effet, l'œil de l'être humain n'est que peu sensible à ces minuscules variations. Aussi, l'algorithme rétrécit la taille des matrices de I et Q, en effectuant la moyenne des deux composantes de la chrominance, dans des carrés de 2x2 pixels. Nous obtenons alors des matrices de chrominance de 320x240.

- La deuxième action est ce qu'on appelle la *transformation*. L'algorithme de compression découpe premièrement les matrices en blocs de 8x8 pixels, et leur applique ensuite la fonction *DCT (Discrete Cosinus Transform)*. Cette fonction DCT est une transformation en série (Fourier), qui délivre une représentation non plus spatiale, mais dans le domaine fréquentiel. En effet, la ligne et la colonne de la matrice représentent les axes X et Y dans le domaine spatial, et la valeur d'une case particulière représente la valeur de Z. Nous raisonnons donc en 3 dimensions. La transformation de la matrice donne une nouvelle matrice contenant cette fois ci, les différentes puissances spectrales pour chaque fréquence. L'élément (0,0) représente la valeur moyenne du signal.

- La *quantification* réside dans le faite que l'algorithme attribue à chaque fréquence, à chaque cellule de la matrice 8x8 pixels, un coefficient de perte. L'algorithme annulera ou diminuera les hautes fréquences qui, représentent les plus petits détails de l'image.

L'atténuation de l'amplitude des différentes fréquences est déterminée par le ratio demandé par l'utilisateur.

- Enfin, la matrice obtenue est linéarités en zigzag, selon le *codage* RLE, et est compressée avec la méthode de Huffman.

Le décodage reprend ces points dans l'ordre inverse en appliquant la fonction inverse.

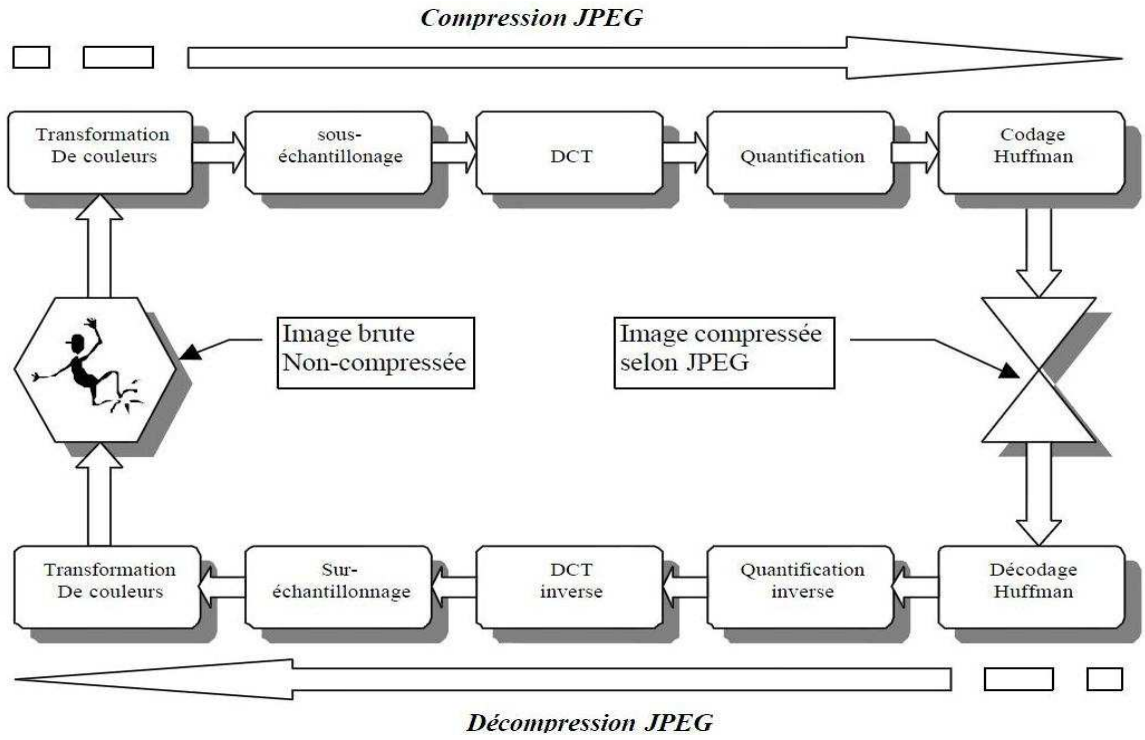


Fig. 3.1 Etapes de la compression et décompression JPEG [39]

3.2 Les algorithmes sans pertes :

La compression *sans pertes* [41] signifie que lorsque des données sont compressées et ensuite décompressées, l'information originale contenue dans les données a été préservée. Aucune donnée n'a été perdue ou oubliée. Les données n'ont pas été modifiées.

Les algorithmes de compression de données sans pertes concernent les données dont l'usage doit conserver l'intégralité des informations et où toute modification peut avoir des influences néfastes sur les données définitives comme par exemple la modification d'un exécutable. Il existe trois principaux algorithmes de compression sans pertes définis dans la littérature et repris par différents format de données :

- ***Huffman* :**

L'algorithme de Huffman [42] est un codage statistique dont le but est de calculer le nombre d'occurrences de chaque caractère de sorte que les caractères les plus fréquents soient codés avec le moins de bits. Cet algorithme se base sur une méthode de construction d'arbres binaires pour coder les octets.

➢ *Principe de l'algorithme de Huffman:*

L'algorithme de Huffman [43] utilise une table contenant les fréquences d'apparition de chaque caractère pour établir une manière optimale de les représenter par une chaîne binaire (cela reprend le principe du morse qui tend à minimiser le nombre de symboles à utiliser pour les lettres les plus fréquemment employées). On peut décomposer la procédure en plusieurs parties :

- Tout d'abord, la création de la table de fréquence d'apparition des caractères dans le texte initial.
- Ensuite la création d'un arbre binaire (usuellement dénommé arbre de Huffman) suivant la table précédemment calculée.
- Enfin coder les symboles en représentation binaire optimale.

1. *Table de fréquence d'apparition :*

Afin de construire cette table, il suffit simplement de dénombrer le nombre d'occurrences de chaque symbole *s* puis de calculer la fréquence f_s de chacun d'entre eux grâce à la formule suivante :

$$fs = \frac{nombre\ d'occurrences\ de\ s}{nombre\ de\ symbole\ s}$$

Ensuite on trie le tableau en fonction de la fréquence d'apparition (de façon croissante) puis suivant le symbole suivant.

2. *Construction de l'arbre de Huffman :*

L'arbre binaire de Huffman est la structure de données qui va nous permettre d'attribuer à chaque symbole une représentation binaire optimale. Afin de construire l'arbre, on utilise la table de fréquences précédemment construite qu'on appelle T et on applique l'algorithme suivant :

Algorithme 1: Construction de l'arbre

Données :

- *T* : la table de fréquence
- *Q* : Une file d'attente de nœuds de l'arbre binaire. Chaque feuille est étiquetée avec un symbole et son nombre d'occurrences. Chaque nœud interne est étiqueté avec la somme des occurrences des feuilles de sa sous arborescence.
- *o* : Une fonction qui à chaque nœud de l'arbre associe une valeur. Si le nœud est une feuille alors *o* renvoie le nombre d'occurrences du symbole, autrement *o* renvoie la somme des occurrences des feuilles de la sous-arborescence du nœud.

Résultat : - *A* : L'arbre binaire résultant

begin

Initialisation de *Q* tel que :
Q contienne les feuilles représentant les symboles de la table *T*

Tant que *(Q non vide)* **faire**

Créer un nouveau nœud z dans A tel que :
gauche(z) = x = extraire-min (Q)
droite(z) = y = extraire-min (Q)
o(z) = o(x) + o(y)
Insérer (z,Q)

Fin

End

De façon informelle, on utilise une file d'attente *Q* dans laquelle on place les nœuds correspondants au couple [symbole : nombre d'occurrences du symbole] de tous les symboles. Ensuite on extrait de la file d'attente les 2 nœuds ayant la valeur minimale puis on crée un nouveau nœud dans l'arbre de Huffman ayant pour fils les deux sélectionnés, on rajoute ensuite le nœud nouvellement crée dans la file d'attente, et on réitère jusqu'à ce que la file soit vide.

- ***RLE :***

Cet algorithme, acronyme de *Run-Length Encoding*, est une forme de codage de données très simple consistant à simplifier l'écriture de flux de données consécutifs qui se répètent (ex : la chaîne aaaaaaabbccccc devient 7a2b5c lorsqu'on applique RLE, ce qui constitue une chaîne plus courte).

Le principal inconvénient de RLE est son efficacité en terme de taux de compression qui dépend de la nature de données. En effet, les flux de données n'ayant pas de séquences répétées consécutives de valeurs donnent de mauvais résultats avec cet algorithme. Ex : Cannibalisation devient 1c1a2n1i1b1a1l1i1s1a1t1i1o1n, ce qui constitue une chaîne plus longue. Cet algorithme est utilisé par les formats bitmaps : BMP, TIFF et PCX et permet de réduire les données graphiques redondantes.

- ***LZW :***

La méthode Lempel-Ziv Welch [44] est un algorithme très populaire, variant de LZ77, qui permet à l'aide d'un dictionnaire de données d'avoir des taux de compression intéressant. Contrairement à RLE, son efficacité n'est pas autant contrainte par la source de données.

Cet algorithme reste incontestablement l'un des algorithmes les plus utilisé dans la compression de données en informatique. Aussi, contrairement à d'autres algorithmes de compression de données basés sur des dictionnaires, il va plus loin en ce sens qu'il n'est pas nécessaire au destinataire de disposer du dictionnaire de données pour décompresser l'information. Lempel-Ziv et ses variantes sont utilisés par plusieurs formats de compression tels que le standard de compression pour modem V24bis ou encore PKZIP.

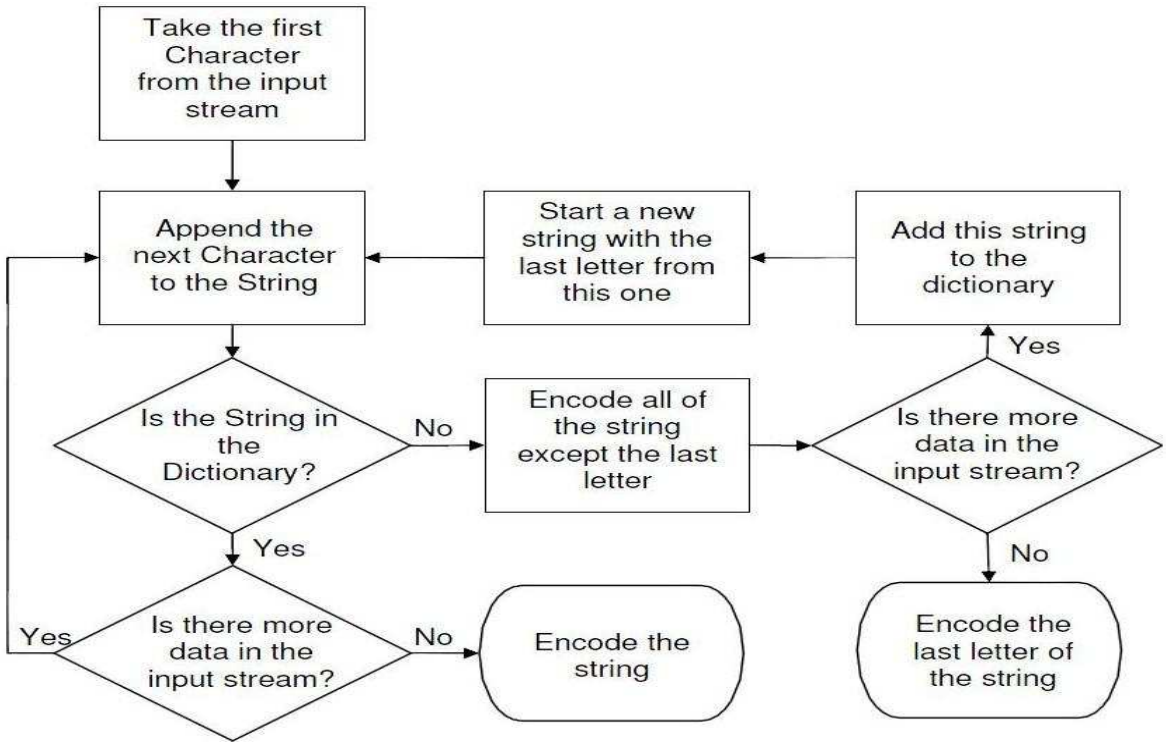


Fig.3.2. Diagramme de déroulement de la compression LZW
(Flow chart of LZWcompression).

Voici l'algorithme de compression de LZW [41]:

Algorithme: L'algorithme LZW de compression

Données :
– Le dictionnaire des symboles rencontrés : Dico
– Le fichier à compresser : Fichier

Résultat :
– Le fichier compressé : Fichier

début
s=premier octet du fichier
tant que le fichier n'est pas à sa fin
t=octet suivant
u=concaténation(s,t)
si (u appartient Dico) s=u
sinon
ajouter (u) dans Dico
écrire adresse de s
s=t
fin si
fin tant que
écrire adresse de s
fin

L'utilisation d'un dictionnaire de données par LZW ou encore la construction d'arbres qui est la méthode utilisée par l'algorithme de Huffman ne sont pas adaptés pour les capteurs qui ont des unités de calculs limitées et des capacités mémoires réduites.

Cependant, il existe une adaptation du célèbre algorithme LZW pour RdCs nommé S-LZW (Sensor-LZW) [45]. Il faut noter que la compression de données dans les réseaux de capteurs sans fil reste encore un axe de recherche prometteur mais peu développé. La littérature sur la compression dans les réseaux de capteurs est pauvre et S-LZW est l'algorithme de référence dans le domaine.

4. L'algorithme S-LZW

L'adaptation de LZW pour réseaux de capteurs, nommée S-LZW (LZW for Sensor Nodes), a permis d'évaluer un algorithme de compression de données basé sur un dictionnaire pour les RdCs. Cependant, la construction d'un dictionnaire de données est un processus gourmand en mémoire, ce qui constitue de prime abord un obstacle à l'utilisation de ce type de techniques sur les capteurs qui ont des mémoires limitées.

De ce fait, pour adapter LZW aux capteurs, une nouvelle morphologie de cet algorithme s'articulant autour de trois principaux points corrélés a été définie. Ces points sont : la limitation de taille du dictionnaire, la limitation de la taille des données à compresser et la

procédure à suivre lorsque le dictionnaire est rempli. Les principaux paramètres par défaut de S-LZW sont les suivants :

- Un bloc de 528 octets pour compresser les données et qui représente deux pages flash. En effet, S-LZW divise les flux de données d'entrée en blocs de taille fixe et compresse chaque bloc de manière indépendante.

- Un dictionnaire limité à 512 entrées. Comme LZW, cet algorithme initialise le dictionnaire avec les 256 valeurs du code ASCII extendu, ce qui représente déjà 256 entrées. Pour chaque block utilisé dans la compression, le dictionnaire est réinitialisé. Une nouvelle chaîne dans le flux d'entrée crée une nouvelle entrée dans le dictionnaire, ce qui fait que le flux de données à compresser est limité. Cependant, différentes stratégies ont été développées afin de résoudre le problème de dictionnaire rempli. Deux options existent qui sont : soit le gel du dictionnaire et l'utilisation telle quel pour compresser le reste des données dans le bloc, ou il peut être remis à zéro et recommencer à zéro. Toutefois, ce problème ne se produit pas lorsque le flux de données à compresser est petit, et donc le dictionnaire n'est pas plein.

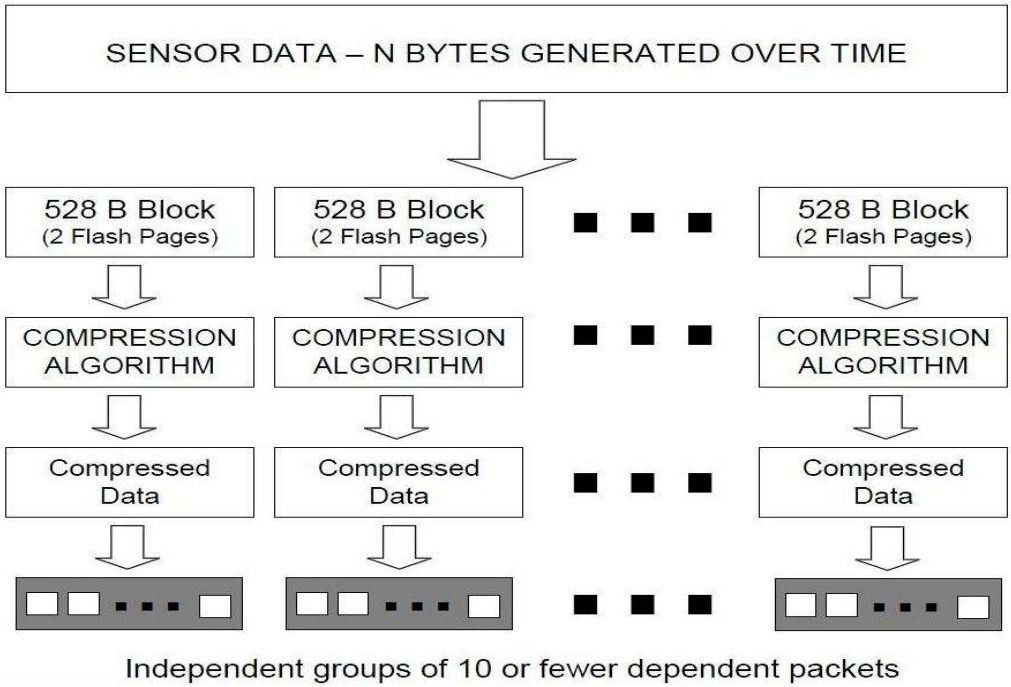


Fig.3.3 Structuration des parquets en blocs indépendants

- Un mini-cache, qui est une table de hachage à 32 entrées est ajouté à S-LZW afin d'obtenir un avantage de la répétition des données du capteur. En effet, les capteurs ayant tendance à avoir des données répétitives sur des intervalles courts, le mini-cache associé au dictionnaire conserve les entrées du dictionnaire récemment utilisées et

crées. L'index de hachage correspond aux quatre derniers bits de l'entrée du dictionnaire.

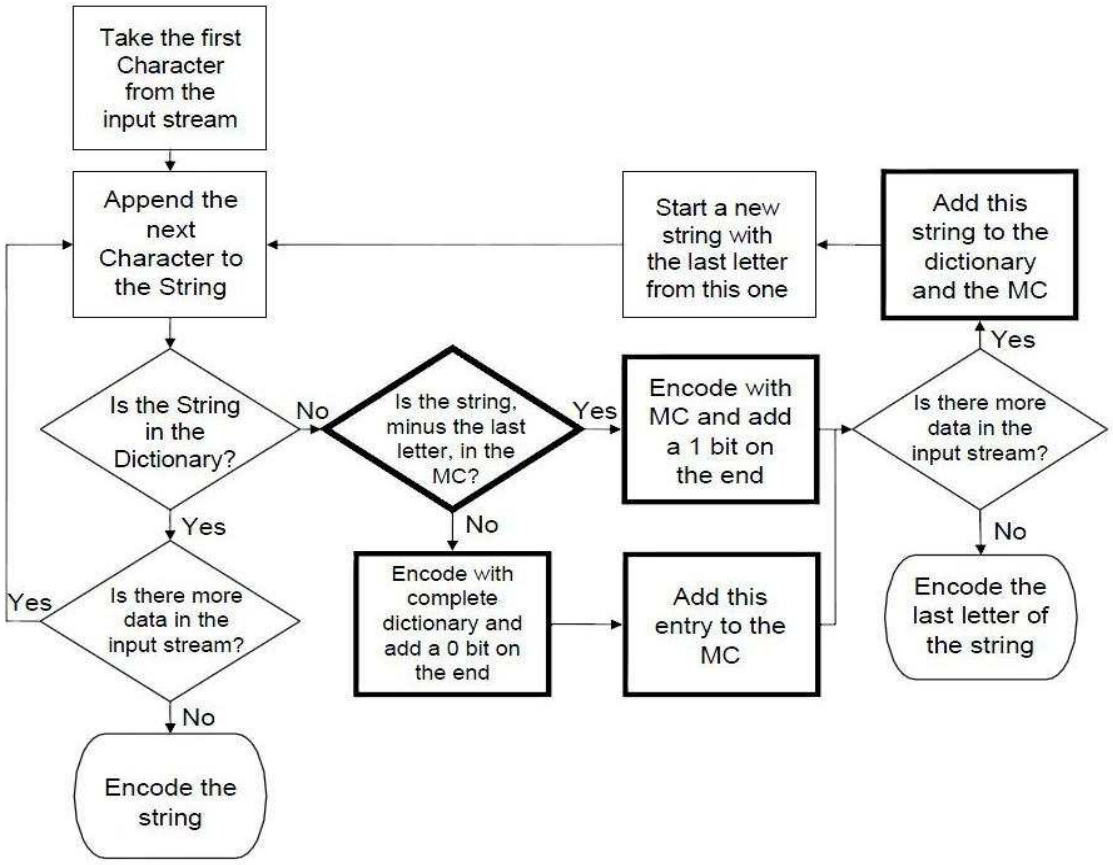


Fig.3.4. Diagramme de déroulement de la compression S-LZW-MC [45]

(Flow chart of S-LZW-MC).

Malgré les adaptations qui précèdent, cet algorithme, basé sur un dictionnaire de données, a besoin d'une mémoire RAM significative et donc n'est pas utilisable sur toutes les plateformes telles que celles par exemple s'articulant autour d'un microcontrôleur TI MSP430F149 avec 2 Ko de mémoire RAM. Dans [45], les performances de S-LZW ont été évaluées sur un microcontrôleur TI MSP430F1611 avec 10 Ko de mémoire RAM.

C'est pour cette raison que nous allons définir de nouveaux algorithmes de compression qui sont utilisables par plusieurs types de plateformes quelque soit la taille de leur mémoire RAM. La question est donc de savoir : comment définir un nouvel algorithme de compression de données au moins aussi efficace que S-LZW et utilisant très peu de mémoire ?

5. L'algorithme RLE (Run-Length Encoding)

Run-Length Encoding (RLE) est un algorithme de compression simple et sans pertes qui a été très utilisé pour les images et l'envoi de fax. Comme décrit dans [5], l'idée de base de cet algorithme est la suivante :

Si un item *d* apparaît *n* fois consécutivement dans le flux de données d'entrée, nous remplaçons les *n* occurrences avec la simple paire *nd*.

La simplicité de RLE est un atout par rapport aux contraintes liées aux ressources des capteurs telles que la capacité mémoire limitée ou encore la puissance de calcul réduite. Cependant un problème perdure, celui de l'efficacité de cet algorithme. En effet, l'efficacité de cette technique de compression est étroitement liée à la nature répétitive des données d'où la question de savoir comment améliorer les performances de cet algorithme en conservant sa simplicité pour des données de statistiques différentes ? Pour répondre à cette question, nous allons tout d'abord nous intéresser aux variantes de RLE, même si, celles-ci sont très spécifiques aux images.

En effet, pour des images matricielles par exemple, il existe des variantes pour encoder de manière séquentielle soit *colonne par colonne, en verticale*, par *flot 4x4 pixels* ou encore en *zigzag* (Fig.3.5). Cependant, toutes ces variantes sont spécifiques aux images matricielles et ce n'est que le type de parcours qui diffère.

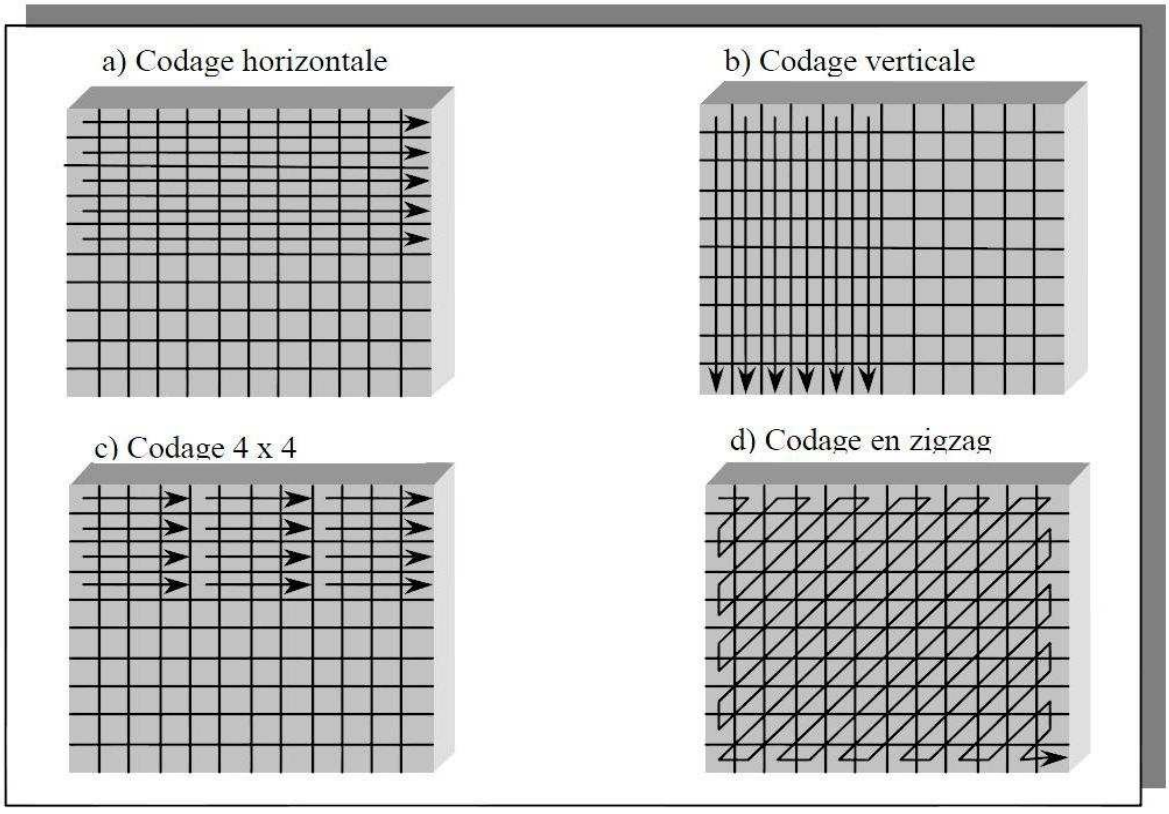


Fig. 3.5 Les variantes de RLE

Des variantes peuvent intervenir au niveau *bits*, *octets* ou *pixels* mais ne change pas l'algorithme en soit, c'est l'atomicité qui change.

Une autre variante de RLE est MNP5, utilisée par les modems et qui évite d'utiliser un caractère de compression spécifique. Le principe de base de cette variante est le suivant :

- un caractère pas répété (singleton) ou répété deux fois est codé tels quels ;
- un caractère répété au moins trois fois est écrit 3 fois, puis suivi du nombre de ses occurrences restantes.

Exemple : bbbbbvbbbvv donne bbb2vbbb0vv.

Cependant, les performances de MNP5 ne pourraient être supérieures à l'algorithme de référence dans le domaine des réseaux de capteurs qui est S-LZW. C'est pour cette raison que nous allons introduire un algorithme avec pertes pour accroître les performances de RLE quelque soient les statistiques des données.

Les variantes de RLE avec pertes sont très rares même s'il en existe qui ignorent des données par exemple en ne tenant pas compte d'un ou deux bits de poids faible pour chaque pixel. L'idée de cette variante qui est spécifique aux images constituera une piste intéressante dans nos choix.

Tandis que plusieurs variantes précédentes sont dédiées aux images et que l'efficacité de MNP5 dépend aussi de la nature répétitive des données, un nouvel algorithme de compression de données à été introduit dans [5], qui conserve la simplicité de RLE tout en améliorant ses performances quelque soient les statistiques des données. Cet algorithme, nommé *K-RLE* est donc inspiré de RLE et signifie RLE avec une précision de K.

6. L'algorithme K-RLE (K-Run-Length Encoding)

Principe

L'idée principale de l'algorithme K-RLE [5] est la suivante :

Définition : Soit *K* un nombre, Si un item *d* ou un item entre *d-K* et *d+K* apparait *n* fois consécutivement dans le flux de données d'entrée, nous remplaçons les *n* occurrences avec la simple paire *nd*.

Dans cette définition, nous introduisons un nouveau paramètre K qui est une précision. Il faut noter que K a la même unité que le flux de données d'entrée, en l'occurrence des degrés pour des températures. Lorsque K est proche de zéro, les résultats de K-RLE et RLE sont similaires.

Cependant, tandis que RLE est une technique conservative, c'est-à-dire sans pertes, l'introduction du paramètre K introduit une modification des données, ce qui fait de K-RLE une technique de compression non conservative, c'est-à dire avec pertes.

Néanmoins, il est important de constater que l'utilisateur peut définir *K* de sorte qu'une précision de *K* sur les données n'influe pas sur ses analyses, c'est-à-dire que les items *d*, *d+K* ou *d-K* soient considérés comme étant les mêmes.

Ainsi, nous allons introduire un nouveau concept *d'algorithme de compression sans pertes au niveau utilisateur*. Dans un premier temps, nous allons nous placer dans le cas bien précis de collecte de données scalaires (la température) et déterminer *K* de manière intuitive en considérant par exemple qu'un écart de 2° n'influencera pas considérablement sur les analyses d'un utilisateur final.

7. Résultats expérimentaux

Dans cette section, nous allons décrire les résultats obtenus en évaluant les algorithmes de compression précédents à l'aide d'un tableau de températures réelles de 500 octets. Ce tableau a été construit grâce aux données collectées sur le site Weather Underground [46] depuis le 1er janvier 2008. Quatre différents lieux du globe ont été pris en compte : Libreville au Gabon, Cayenne en Guyanne, Montbéliard en France et Svalbard en Norvège. Nous avons choisi ces différents lieux afin d'étudier le comportement des algorithmes précédents avec des variations de températures différentes (Fig.3.6). Ces algorithmes seront évalués selon une représentation compatible avec une mesure réalisée par un microcontrôleur TI MSP430 dédié aux applications faible consommation.

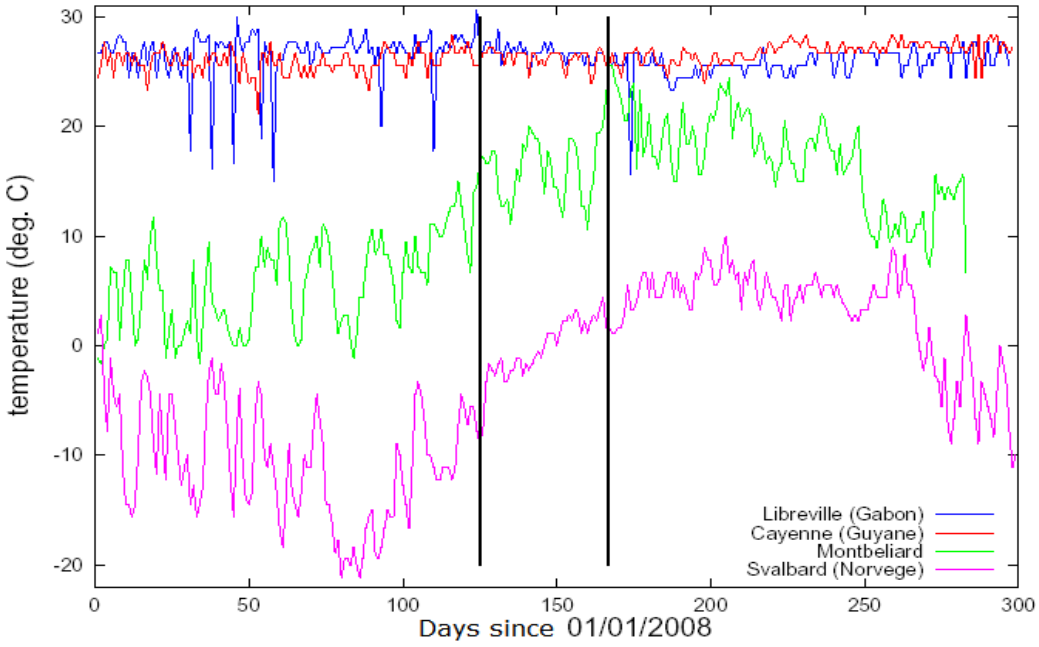


Fig.3.6 Représentation des variations de températures de différents endroits du globe [5]

Le module ADC12 [5] est présent sur les microcontrôleurs MSP430x14x et MSP430x16x. L'ADC12 est un convertisseur analogique-numérique 12 bit de haute performance dont les caractéristiques inclus un module de capture de températures. La fonction de transfert du capteur de température est la suivante :

*VTEMP = 0.00355 * (TEMPC) + 0.986*

Grâce à cette fonction de transfert, les séquences de mesures de températures en convertissant les archives quotidiennes de Weather Underground [46] ont été simulés par [5] en valeurs hexadécimales sur 12 bits telles que les auraient mesuré l'ADC du microcontrôleur TI MSP430. Ces températures sont donc codées sur trois octets :

b71 b7c b91 b7c b86 b71 b86 b86 b7c b86 ...

La Fig. ci-dessus représente les archives de températures des quatre lieux allant de latitudes proches de l'équateur au cercle polaire arctique. Nous constatons que les variations de températures sont d'autant plus importantes qu'on s'éloigne de l'équateur. Nous allons donc évaluer les algorithmes dans ses différentes conditions réelles avec des variations de températures différentes.

Comparaison des taux de compression des algorithmes de compression

Nous allons observer le taux de compression des deux algorithmes *S-LZW* et *RLE* (Fig.3.7). La Fig.3.7 montre que même si les variations des résultats de compression sont les mêmes, il y a une grande différence au niveau des taux de compression.

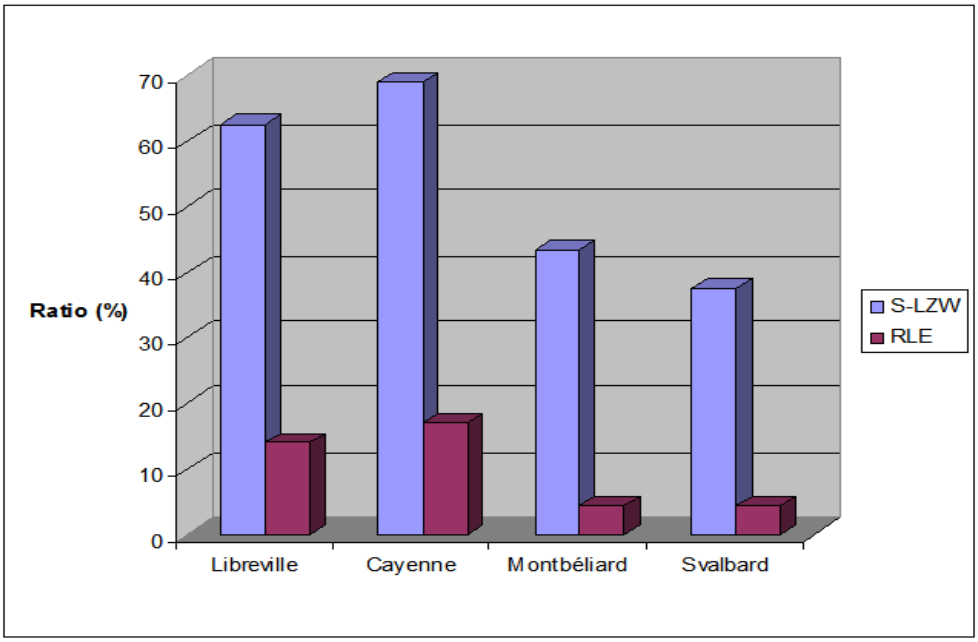


Fig.3.7 Comparaison entre S-LZW et RLE [5]

En effet, pour chacun des algorithmes, les taux de compression sont de moins en moins bons lorsqu'on s'éloigne de l'équateur. Cependant, S-LZW offre de bien meilleurs résultats que RLE, ce qui était prévisible mais intéressant à évaluer. Tandis que le meilleur taux de compression de RLE pour les quatre villes confondues est 17%, le taux moyen pour S-LZW est d'environ 53%.

Les résultats qui précèdent illustrent bien l'intérêt d'introduire un nouvel algorithme de compression simple et non gourmand en mémoire, capable non seulement d'être utilisé par plusieurs plateformes telles que celles basées sur un microcontrôleur TI MSP423F149, mais également capable d'être aussi performant que les algorithmes qui utilisent un dictionnaire de données et donc qui nécessitent de la mémoire comme S-LZW. Ces différents critères ont guidés vers une variante de RLE appelé *K-RLE*.

La Fig.3.8 décrit les résultats de 2-RLE par rapport à S-LZW. Dans la majorité des cas, 2° RLE offre de meilleurs résultats que S-LZW. Le taux de compression moyen pour 2-RLE est de 56% pendant qu'il est de 53% pour S-LZW.

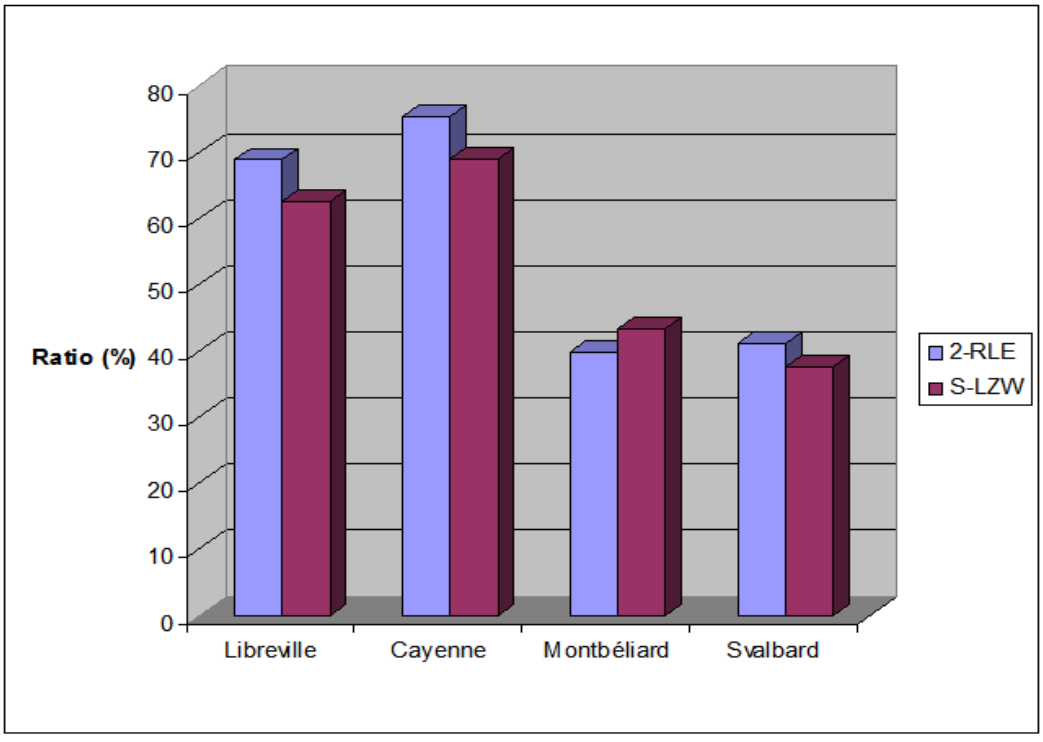


Fig.3.8 Comparaison entre S-LZW et K-RLE [5]

Le taux de données modifiées est le pourcentage de données rendues identiques à celles qui précèdent proches de K (Fig.3.8). Nous constatons que plus nous sommes proches de l'équateur, plus les variations de températures sont faibles, ce qui fait que le taux de données modifiées dans ces lieux est très élevé car les variations sont petites. Nous avons en moyenne

50% des données modifiées sur l'ensemble des zones étudiées, ce qui signifie que la moitié des données a été modifiée pour une précision de K égale à 2°.

Ces résultats montrent que le nouvel algorithme de compression K-RLE, inspiré de RLE, améliore les performances de RLE au détriment des données dont 50% en moyenne sont modifiées pour une précision de K = 2°.

Cependant, contrairement à S-LZW qui est un algorithme de compression sans pertes et qui offre d'aussi bons taux de compression que 2-RLE, certes en moyenne 3% moins bons, K-RLE a beaucoup moins de contrainte mémoire et est utilisable sur plusieurs plateformes telles que celles basées sur un microcontrôleur TI MSP423F149 qui n'a que 2 Ko de mémoire RAM.

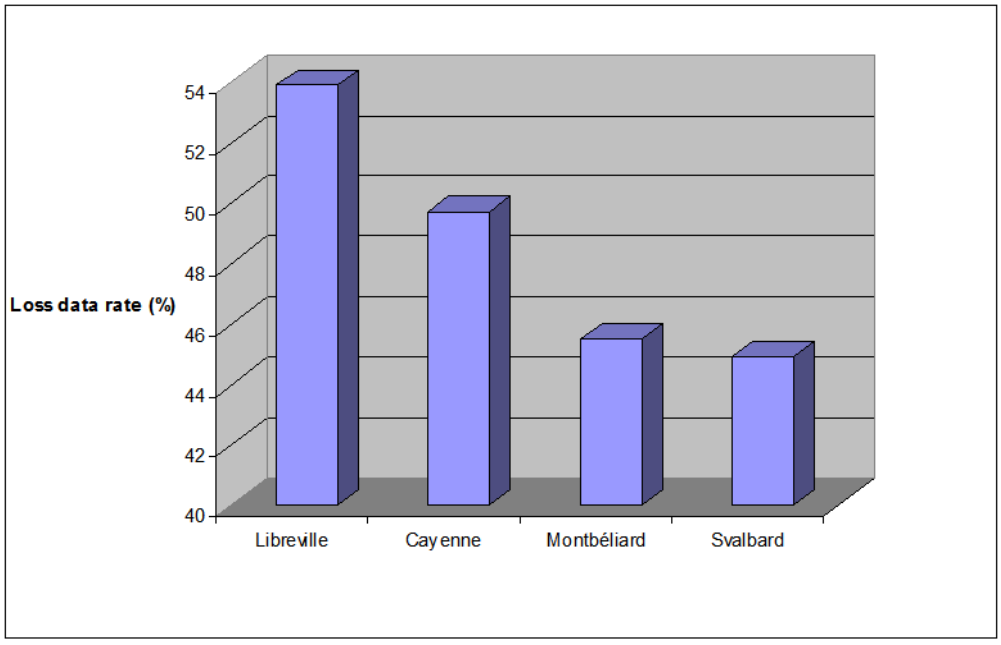


Fig.3.9 Représentation du taux de données modifié par 2-RLE

Après avoir évalué l'efficacité de nos algorithmes en terme de compression, nous allons nous intéresser à l'évaluation de la consommation énergétique pendant la compression.

Comparaison de la consommation énergétique des algorithmes de compression

Dans cette section, une évaluation de la consommation énergétique des algorithmes de compression précédents à l'aide d'un simulateur détaillé de plateforme matérielle appelé *WSim* et développé par une équipe du Laboratoire CITI, INRIA Rhône Alpes. Ce simulateur constitue une partie d'un environnement intégré pour le développement et le prototypage rapide des applications de réseaux de capteurs sans fil connue sous le nom de Worldsens [5].

Nous allons observer les résultats obtenus [5] à l'aide du simulateur WSim pour les différents algorithmes dans les différentes zones.

La Fig.3.10 détaille la consommation énergétique des différents algorithmes pendant la phase de compression. Nous constatons que l'algorithme S-LZW étant sans perte avec des taux de compression intéressants, consomme le plus d'énergie. S-LZW utilise en moyenne 0,0224 mJ tandis que RLE consomme globalement 0,0053 mJ et 2-RLE 0,0103 mJ. RLE consomme moins que les deux autres algorithmes. Ces résultats mettent en lumière le compromis existant entre l'efficacité de compression et la consommation énergétique.

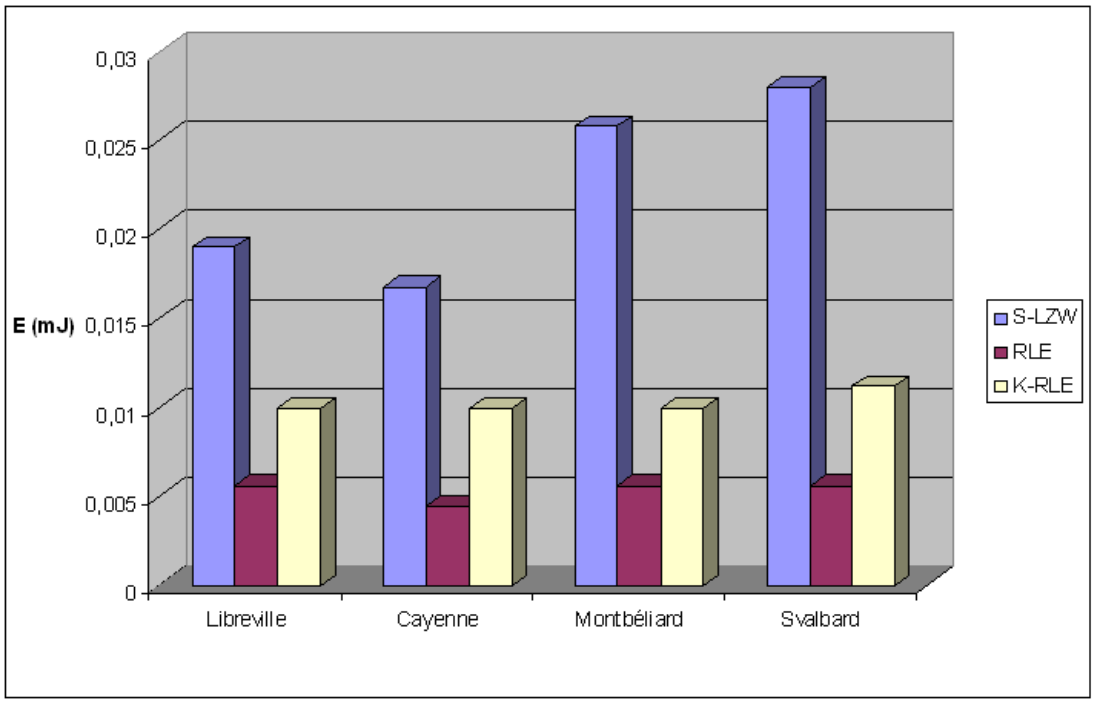


Fig. 3.10 Evaluation de consommation de la phase de compression des algorithmes

Aussi, nous remarquons que pendant que RLE et 2-RLE ont des variations constantes au niveau de la consommation, S-LZW utilise plus d'énergie quand il y a beaucoup de variation au niveau des données d'entrée, typiquement pour des zones proches du cercle polaire arctique.

La Fig.3.11 montre que 2-RLE consomme plus que RLE pendant la phase de compression, il consomme beaucoup moins, environ 0,0011 mJ, pendant la phase de décompression. Cependant, S-LZW et RLE consomment respectivement en moyenne 0,015 mJ et 0,00165mJ. La remarque précédente est encore confirmée, S-LZW consomme plus quand il y a beaucoup de variation au niveau des données d'entrée.

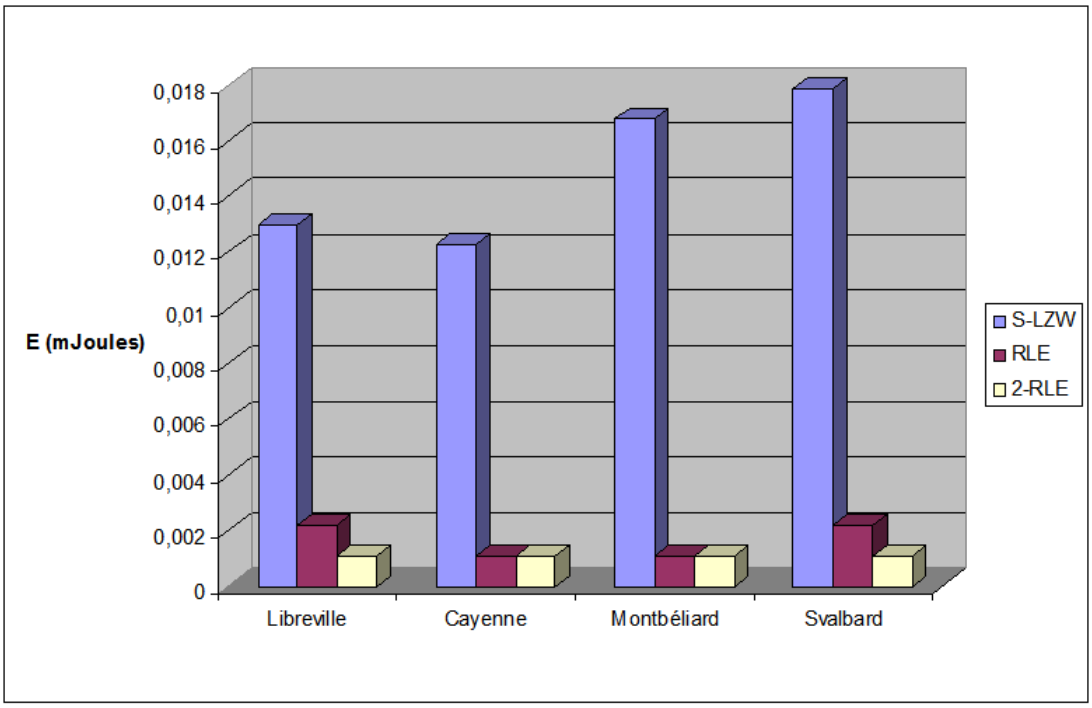


Fig. 3.11 Evaluation de consommation de la phase de décompression des algorithmes

Le tableau récapitulatif Tab.3.1 décrit les caractéristiques de chacun des algorithmes étudiés. K-RLE, qui est le plus performant d'entre eux, a un point faible qui reste la modification des données pour améliorer ses performances.

Algorithmes	**Avantages**	**Inconvénients**
Sensor-Lempel Ziv Welch (S-LZW)	Sans pertes et efficace avec un bon taux de compression	Gourmand en mémoire
Run Length Encoding (RLE)	Très simple, rapide et sans pertes	Dépend de la nature des données
K-Run Length Encoding (K-RLE)	Très efficace, simple et rapide avec un bon taux de compression	Modification des données

Tab.3.1 Tableau récapitulatif des caractéristiques des algorithmes de compression précédents

8. Conclusion :

La compression de données dans les réseaux de capteurs se révèle être une technique efficace pour économiser l'énergie mais la mise en place d'un algorithme de compression performant dépend de critères souvent antagonistes. En effet, les ressources limitées des capteurs, les contraintes énergétiques, l'efficacité de compression et la conservation des données sont des critères souvent opposés.

Dans ce chapitre, nous avons dressé un panorama d'algorithmes de compression de données existant dans la littérature adaptés pour les réseaux de capteurs sans fils, notamment une nouvelles techniques de compression K-RLE. K-RLE est un algorithme de compression avec perte dont le niveau de perte dépend du paramètre K. Différentes méthodes pour trouver K ont été introduites qui sont l'écart-type et la variance d'Allan. Tandis que l'application de K-RLE avec l'écart-type offre de bons taux de compression, la variance d'Allan permet de maitriser la perte des données.

Toutefois, avec K-RLE, il à été prouvé qu'on est capables d'avoir de meilleures performances par rapport à l'algorithme de référence S-LZW.

En somme, la compression de données, qui peut dans certains cas associée économie d'énergie et optimisation du temps de transfert des données par rapport à une transmission sans compression, ouvre des perspectives de recherche intéressantes dans les réseaux de capteurs.

Chapitre IV: Plateforme de Simulation et Simulation

Sommaire

Partie I : plateforme de Simulation

1. Introduction :

Il existe de nombreuses applications du monde réel dont lequelle les capteurs sont utilisés. Généralement ils sont utilisés dans des applications de recueil et de transfert de l'information essentielle. Avec les progrès de la technologie, il ya une utilisation accrue des capteurs dans tous les domaines de la vie à extraire plus d'information pour l'avancement de l'humanité non seulement en termes de sécurité, confort, mais aussi dans le commerce. Comme la propagation et l'utilisation des capteurs augmentent, les applications qui sont écrites pour eux se sont diversifiées. Par conséquent, il est nécessaire de rendre ce processus de développement de ces applications plus facile en utilisant des outils d'automatisation et de simulation.

2. Différentes approches de tests :

Que ce soit pour un protocole ou une application, le développeur est confronté aux mêmes problèmes de test et d'évaluation que dans les réseaux filaires. En effet, lors du développement d'un protocole ou d'une application répartie, il est nécessaire d'évaluer le comportement des différents mécanismes proposés, de pouvoir les comparer avec ceux existants et mesurer les performances dans des conditions réseaux particulières. Pour les réaliser, il est possible d'utiliser plusieurs méthodes: *le test en environnement réel*, *la simulation* et *l'émulation* [48].

2.1 Test en environnement réel

La première solution consiste à utiliser un réseau de capteurs réel pour bien mener son évaluation. Bien qu'elle permette d'avoir des conditions parfaitement réalistes, cette solution n'est pas complètement satisfaisante car elle ne permet pas de contrôler l'ensemble des paramètres de l'environnement. De plus, cette solution ne permet pas de reproduire plusieurs fois de suite les mêmes conditions avec précision. En effet, les conditions de propagation du signal à l'intérieur du réseau ne sont pas contrôlables, si bien que d'une expérience à l'autre les conditions observées peuvent évoluer et parfois modifier considérablement le résultat des tests.

2.2 Simulation

La seconde solution, souvent privilégiée par les chercheurs, est la simulation. Celle-ci permet d'évaluer un modèle d'application ou de protocole dans un environnement entièrement contrôlable. Pour cela, la simulation s'appuie sur des modèles décrivant l'environnement, des modèles décrivant les couches de communication utilisées par les terminaux sans fil ainsi que d'autres équipements du réseau et un modèle du trafic circulant sur le réseau. Cependant, la simulation ne travaille pas en temps réel ce qui empêche par exemple l'évaluation d'applications interactives. Utiliser un modèle au lieu de l'implémentation réelle est également pénalisant: la validité du modèle ne garantit pas le bon

déroulement de son implémentation et son déploiement. Des erreurs de programmation peuvent toujours survenir au moment de l'implémentation si bien que cette implémentation doit ensuite être évaluée en environnement réel pour vérifier qu'elle se comporte selon son modèle.

2.3 L'émulation

L'émulation peut être vue comme un compromis entre les deux solutions précédentes en permettant d'évaluer un protocole ou une application dans un environnement contrôlable et reproductible qui simule en temps réel les conditions telles que: les débits, les délais et les pertes que l'on observe dans le réseau cible. Pour cela, l'émulation va simuler les effets des couches basses de telle manière qu'un protocole ou une application s'exécute dans les mêmes conditions que celles de l'environnement réel. Cette solution de test peut par ailleurs être vue comme une étape supplémentaire dans le cycle de développement entre la phase de simulation, qui permet de concevoir les mécanismes spécifiques à un protocole ou à une application et le déploiement dans un environnement réel.

Bien que l'émulation paraisse plus intéressante et porte plus d'avantages, elle impose cependant certaines contraintes. Comme l'émulation travaille en temps réel, les modèles utilisés pour simuler les couches basses ne peuvent pas être trop complexes ce qui implique un impact négatif sur le réalisme de l'émulation rendue. A cet effet, l'approche de simulation détient toujours sa place comme solution pour le test et la validation de protocoles et d'applications.

3. Caractéristiques des environnements de simulation de réseaux sans fil

Un simulateur réseau ou à événements discrets se caractérise par le fait que les changements d'états dans le réseau simulé (événements) se produisent à des instants répartis de manière discrète sur l'axe de temps. Classiquement, une simulation consiste à reproduire le comportement du système complet dans un environnement synthétique où le temps est simulé par une horloge à événements discrets. La simulation demande une modélisation complète, depuis les applications jusqu'au réseau physique, ce qui peut se révéler à la fois complexe et coûteux en terme de temps d'exécution et d'utilisation mémoire dans le cas de réseaux de taille importante. En général, un simulateur réseau doit avoir les caractéristiques suivantes [49]:

3.1 Génération de la topologie du réseau

Chaque simulateur a son mécanisme pour générer la topologie du réseau. Plusieurs possibilités existent: un scripte ou un langage de configuration spécifique au simulateur, des fichiers textes ou via des interfaces graphiques. Le simulateur doit aussi permettre la création de topologies hiérarchiques et la génération automatique de topologies aléatoires.

3.2 Génération et la gestion du trafic réseau

Afin de générer le trafic de données circulant sur l'environnement de simulation, il est nécessaire d'utiliser des générateurs de données suivant une distribution qui doit être la plus

représentative du trafic réel et qui permettra d'établir les statistiques nécessaires pour tirer des conclusions.

3.3 Contrôle du réseau

Pendant l'exécution, il est très utile de contrôler dynamiquement le réseau soit par flux de données ou par nœuds via des interfaces graphiques. Les traces et les résultats du contrôle peuvent êtres sauvegardés dans des fichiers afin de faire des comparaisons ou ré-exécuter la simulation avec un paramétrage plus affiné.

3.4 Modèle protocolaire, de mobilité et de propagation radio

Un bon simulateur dans son modèle protocolaire doit disposer de plusieurs protocoles dans les différentes couches de communication. Dans certains cas pour les réseaux sans fil, nous avons besoin de gérer la mobilité des nœuds. Cela se fait généralement par l'intégration d'un modèle de mobilité dans l'environnement de simulation. De plus, le modèle de la propagation radio pour les réseaux sans fil a un grand impact sur les résultats de la simulation, donc le simulateur doit fournir plusieurs modèles de propagation radio qui doivent être les plus proches possible du cas réel.

3.5 Flexibilité et extensibilité

Le simulateur doit donner la possibilité de contrôler facilement les composants dans différents niveaux (changer leurs états par exemples). De plus, les modèles disponibles peuvent êtres remplacés, modifiés ou enrichis par l'ajout de nouveaux modèles sans perturber le bon fonctionnement du simulateur. D'une manière générale, l'architecture du simulateur doit être modulaire et ouverte.

3.6 Temps d'exécution

Le temps d'exécution est un facteur qui doit être pris en considération dans le choix du simulateur et surtout s'il s'agit de simuler des réseaux de grandes tailles comme c'est le cas pour les réseaux de capteurs.

3.7 Ergonomie, visualisation des résultats et statistiques

L'environnement de simulation doit être le plus ergonomique que possible et doit faciliter les tâches de configuration et de suivie des différentes étapes de la simulation via des interfaces graphique. De plus, une représentation graphique des résultats et statistiques à travers des diagrammes, graphes ou sur des cartes géographiques est très importante afin de permettre une meilleure interprétation des résultats.

4. Tinyos : environnement de développement dédié aux capteurs

Un système d'exploitation, *Operating System* en anglais et abrégé OS, se définit comme un programme garantissant la liaison entre les ressources matérielles et logiciels (applications

informatiques). Pour faire fonctionner un capteur, un système d'exploitation peut constituer un outil intéressant, cependant soumis aux contraintes sévères des capteurs, il est spécifique.

En effet, les systèmes d'exploitation classiques utilisés par les ordinateurs et les plateformes disposant de ressources importantes ne sont pas adaptés aux capteurs car ils offrent un support natif pour un grand nombre de périphériques, ce qui demande des ressources conséquentes que les capteurs n'ont pas forcement. Mais ce défaut est aussi un avantage car l'architecture qui est mise en œuvre permet à une application d'être en mesure de pouvoir fonctionner sur n'importe quelle machine sans pour autant connaître les caractéristiques matérielles, donc bas niveau de celle-ci.

Par contre, l'informatique embarquée se base historiquement sur des langages de bas niveau le C ou l'assembleur. Le programme dialogue directement avec le matériel, ce qui le rend plus léger en terme de ressource et de capacités. En effet, il n'est plus sur un modèle de plus grand dénominateur commun comme l'OS mais sur un modèle de spécificité qui lui apporte cette caractéristique. Cependant, cet avantage est aussi son inconvénient. Le logiciel et le matériel étant fortement liés, toute modification matérielle entraîne nécessairement une modification logicielle. De là, on peut noter qu'un système embarqué impose une solution pointue et économe en terme de ressources qui rend l'utilisation des systèmes d'exploitation classiques difficile. De ce fait, différents OS ont été développés pour systèmes embarqués et aussi plus spécialement pour réseaux de capteurs [5] tels que :

- **Contiki**: est un système d'exploitation *open source*, léger, multitâche, générique développé pour tout système embarqué ayant des contraintes mémoire. La portabilité de ce système va de l'ordinateur ayant une architecture 8 bits aux systèmes embarqués sur des microcontrôleurs comme certaines plateformes pour RdCs.

- **FreeRTOS** : est un noyau de système d'exploitation temps réel pour système embarqué et portable sur plusieurs microcontrôleurs.

- **TinyOS :** est le premier système d'exploitation spécialement dédié aux réseaux de capteurs. C'est un OS *open source* initialement conçu à l'Université de Berkeley en Californie et qui connait une notoriété dans le domaine des réseaux de capteurs non seulement par ce qu'il est le premier mais aussi parce qu'il est utilisé par les plateformes CROSSBOW [31], les plus populaires dans le domaine qui ont été aussi développés par la même université. Le langage utilisé par TinyOS est nesC [50] qui est une extension du langage C.

- **Mantis OS:** est aussi un OS pour réseaux de capteurs entièrement écrit en C et développé par une équipe de l'université du Colorado. Ce système utilise une approche différente des autres OS pour réseaux de capteurs sans fil qui est *thread driven execution model* différent de TinyOS qui utilise un *event driven model*.

- **SOS:** est un OS pour réseaux de capteurs utilisant le même modèle, *non-preemptive event driven scheduler*, que TinyOS mais écrit en C. Il a été développé à l'université de Californie, Los Angeles (UCLA).

- **NutOS :** c'est un OS temps réel open source conçu pour système embarqué. Il est multitâche et offre une pile TCP/IP. Il a constitué une base pour ETHERNUT ou encore pour l'OS BTNut utilisé dans les BTnodes et où une pile Bluetooth open source a été rajoutée.

Les OS qui précèdent sont spécialement dédiés aux plateformes de réseau de capteurs qui ont des ressources limitées. Aussi, il faut noter qu'il existe d'autres plateformes de haut niveau avec plus de ressources.

Après avoir énuméré les différents OS pour réseaux de capteurs, nous allons présenter le plus populaire d'entre eux, qui est également celui que nous avons utilisé.

5. TinyOS: Tiny Microthreading Operating System

Contrairement aux systèmes d'exploitation classiques, TinyOS se présente comme une solution intermédiaire entre une application bas-niveau apportant une faible empreinte mémoire mais trop spécifique au matériel et un système d'exploitation apportant une réelle notion d'abstraction entre l'application et le matériel mais qui n'est pas prévu pour du matériel à faible ressources.

TinyOS n'est pas un système d'exploitation au sens actuel du terme. Il n'offre pas par exemple de notion de multitâches, d'utilisateurs ou de système de fichier. Il n'y a pas de notions de mode utilisateurs et de mode noyau. En fait, TinyOS met un ensemble d'outils à la disposition du programmeur en vue de simplifier le développement. Il utilise une extension du langage C pour systèmes embarqués appelée NesC [51] et l'exécutable produit est en tout point semblable à ce que pourrait générer une application bas niveau.

L'intérêt de TinyOS ne réside pas spécialement dans l'application installée sur le capteur mais au niveau de la simplification du développement. Il apporte la même notion d'abstraction qu'un système d'exploitation classique, sans imposer une trop forte dépendance entre le matériel et le logiciel rendant dès lors possible une portabilité quasi transparente sur une multiplicité de plateformes différentes, et surtout sans aucunes contraintes fortes liées au matériel.

TinyOS a été spécifiquement conçu pour les réseaux de capteurs [52] et donc répond à leurs exigences en terme de ressources. Il utilise un fonctionnement évènementiel différent du fonctionnement dit temporel où les actions du système sont gérées par une horloge donnée.

L'architecture de TinyOS est basée sur une association de *composants*, ce qui a réduit la taille du code nécessaire à sa mise en place. Une application écrite dans ce langage se

compose d'un ou plusieurs composants assemblés. Les composants ont deux champs principaux : un pour leur spécification qui contient les noms de leurs *interfaces*, et un deuxième pour l'*implémentation*. Un composant fournit et utilise des interfaces.

Les *interfaces* sont fournies dans le but de représenter les fonctionnalités qu'offre le composant à son utilisateur dans sa spécification. Il existe deux types de composants dans nesC : *module* et *configuration*. Les *modules* fournissent les implémentations d'une ou de plusieurs interfaces. Les *configurations* sont utilisées pour l'assemblage d'autres composants ensemble, en connectant les interfaces utilisées par les composants pour les interfaces fournies par d'autres.

5.1 Propriétés de la plateforme TinyOS

TinyOS [52] a été conçu pour la programmation des réseaux de capteurs et il est caractérisé par :

- ***Disponibilité et sources*** : TinyOS est un système principalement développé et soutenu par l'université américaine de Berkeley, qui le propose en téléchargement sous la licence BSD et en assure le suivi;

- ***Event-driven*** : Le fonctionnement d'un système basé sur TinyOS s'appuie sur la gestion des évènements qui se produisent instantanément. Ainsi, l'activation de tâches, leur interruption ou encore la mise en veille du capteur s'effectue à l'apparition d'évènements, ceux-ci ayant la plus forte priorité. Ce fonctionnement basé sur les évènements (Event-driven) s'oppose au fonctionnement dit temporel (time-driven) où les actions du système sont gérées par une horloge donnée ;

- ***Langage*** : TinyOS a été programmé en langage NesC que nous allons détailler plus tard ;

- ***Préemptif*** : Le caractère préemptif d'un système d'exploitation précise si celui-ci permet l'interruption d'une tâche en cours. TinyOS ne gère pas ce mécanisme de préemption entre les tâches mais donne la priorité aux interruptions matérielles. Ainsi, les tâches entre-elles ne s'interrompent pas mais une interruption (sous forme d'un évènement) peut stopper l'exécution d'une tâche ;

- ***Temps réel*** : Lorsqu'un système est dit « temps réel » celui-ci gère des niveaux de priorité dans ses tâches permettant de respecter des échéances données par son environnement. Dans le cas d'un système strict, aucune échéance ne tolère de dépassement contrairement à un système temps réel mou. TinyOS se situe au-delà de ce second type car il n'est pas prévu pour avoir un fonctionnement temps réel ;

- ***Consommation*** : TinyOS a été conçu pour réduire au maximum la consommation en énergie d'un nœud capteur. Ainsi, lorsqu'aucune tâche n'est active, il se met automatiquement en mode veille.

Le tableau ci-dessous résume ses propriétés :

Propriété	Valeur pour Tinyos
Type	**Even-Driven**
Disponibilité	**Open Source**
Langage	**NesC**
Préemptif	**Non**
Temps Réel	**Non**

Tab.4.1 Propriété de TinyOS

5.2 Cible de TinyOS

Il existe de nombreuses cibles possibles pour ce système d'exploitation embarqué. Malgré leurs différences, elles respectent toutes globalement la même architecture basée sur un noyau central autour duquel s'articulent les différentes interfaces d'entrée-sortie, de communication et d'alimentation. Voici un schéma représentant cette architecture :

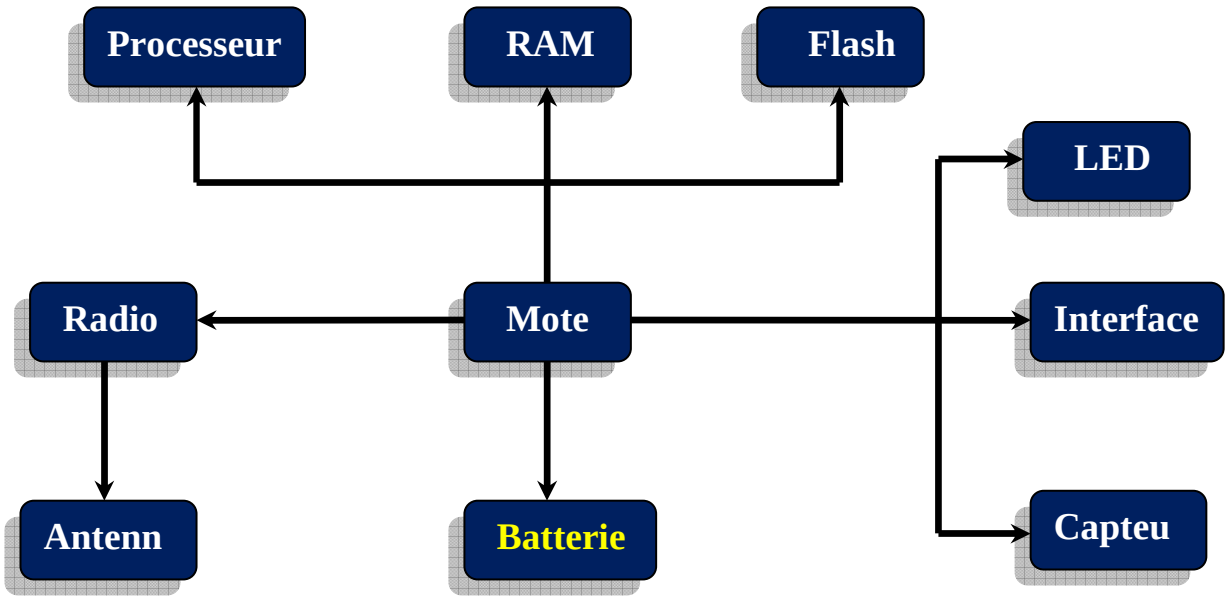


Fig.4.1: architecture d'un nœud capteur

- ***Mote, processeur, RAM et Flash*** : On appelle généralement *Mote* la carte physique utilisant TinyOS pour fonctionner. Celle-ci a pour cœur le bloc constitué du *processeur* et des mémoires *RAM* et *Flash*. Cet ensemble est à la base du calcul binaire et du stockage, à la fois temporaire pour les données et définitif pour le système TinyOS.

- ***Radio et antenne*** : TinyOS est prévu pour mettre en place des réseaux sans fils, les équipements étudiés sont donc généralement équipés d'une *radio* ainsi que d'une *antenne* afin de se connecter à la couche physique que constitue les émissions hertziennes.

- ***LED, interface, capteur*** : TinyOS est prévu pour mettre en place des réseaux de capteurs, on retrouve donc des équipement bardés de différents types de détecteurs et autres entrées.

- ***Batterie*** **:** Comme tout dispositif embarqué, ceux utilisant TinyOS sont pourvus d'une alimentation autonome telle qu'une *batterie*.

5.3 Gestion Allocation de la mémoire

Il est important de préciser de quelle façon un système d'exploitation aborde la gestion de la mémoire. C'est encore plus significatif lorsque ce système travaille dans un espace restreint. TinyOS a une empreinte mémoire très faible puisqu'il ne prend que 300 à 400 octets dans le cadre d'une distribution minimale. En plus de cela, il est nécessaire d'avoir 4 Ko de mémoire libre qui se répartissent :

- ***La pile*** : sert de mémoire temporaire au fonctionnement du système notamment pour l'empilement et le dépilement des variables locales ;

- ***Les variables globales*** : réservent un espace mémoire pour le stockage de valeurs pouvant être accessible depuis des applications différentes ;

- ***La mémoire libre*** : pour le reste du stockage temporaire.

La gestion de la mémoire possède de plus quelques propriétés. Ainsi, il n'y a pas d'allocation dynamique de mémoire et pas de pointeurs de fonctions. Bien sur cela simplifie grandement l'implémentation. Par ailleurs, il n'existe pas de mécanisme de protection de la mémoire sous TinyOS ce qui rend le système particulièrement vulnérable aux crash et corruptions de la mémoire.

5.4 Modèle d'exécution de TinyOS

Pour maintenir une grande efficacité requise par les réseaux de capteurs, TinyOS utilise une programmation par évènement. Ce modèle permet un très haut niveau de concurrence pour un espace très réduit de mémoire.

L'implémentation des composants de TinyOS s'effectue en déclarant des tâches, des commandes ou des évènements. Nous allons détailler ceux-ci dans le tableau suivant :

Action	Utilisation
Tâche	**Travaux de «longue durée »**
Commande	**Exécution d'une fonctionnalité précise dans un autre composant**
Evénement	**Equivalent logiciel à une interruption matérielle**

Tab4.2 Différents actions dans TinyOS

TinyOS dispose de 2 types d'entités ordonnançables : *interruptions* (évènements) et *tâches*

5.5 Programmation par évènement

Dans un système basé sur la programmation par évènement, chaque exécution est partagée entre les différentes tâches de traitement. Dans TinyOS, chaque module est désigné pour fonctionner en attendant continuellement de répondre aux évènements inattendus. Quand un évènement est signalé, le traitement correspondant est exécuté. Une fois totalement terminé, la main est redonnée au système pour continuer sa tâche antérieure.

En plus de l'efficacité de l'allocation du CPU, la programmation par évènement permet d'obtenir des opérations économiques en énergie. Il est très important aussi pour la consommation d'énergie que les applications signalent la fin de leurs évènements et l'utilisation du CPU. Dans TinyOS, les tâches associées avec un évènement sont exécutées très rapidement après leurs signalisations. Une fois terminé, le CPU entre en veille en attendant une nouvelle réception d'évènement.

5.6 Tâches

Un des facteurs limitant la programmation par évènement est la longue exécution des tâches qui peut interrompre d'autres programmes importants. Si l'exécution d'un évènement ne finit jamais, toutes les autres fonctions vont être interrompues. Pour éviter ce problème, TinyOS fourni un mécanisme d'exécution appelé *tâche*. Une tâche est un bout de programme

qui s'exécute jusqu'à la fin sans interférer avec les autres évènements. Les tâches sont utilisées pour effectuer la plupart des blocs d'instruction d'une application.

A l'appel d'une tâche, celle-ci va prendre place dans une file d'attente de type FIFO mais elle ne sera exécuté que lorsque il n'y a plus d'évènements. En plus les tâches peuvent être interrompues à tout moment par des évènements.

D'autre part, il n'y a pas de mécanisme de préemption entre les tâches et une tache activée s'exécute en entier. Ce mode de fonctionnement permet de bannir les opérations pouvant bloquer le système (inter blocage, famine, …). Par ailleurs, lorsque la file d'attente des taches est vide, le système d'exploitation met en veille le dispositif jusqu'au lancement de la prochaine interruption (on retrouve le fonctionnement Event-driven).

5.7 Description de l'ordonnanceur TinyOS

La gestion des tâches et des évènements est vitale pour TinyOS. Le choix d'un ordonnanceur déterminera le fonctionnement global du système et le dotera de propriétés précises telles que la capacité à fonctionner en temps réel.

L'ordonnanceur TinyOS c'est :

- **2 niveaux de priorité** (bas pour les tâches, haut pour les évènements) ;
- **1 file d'attente FIFO** (disposant d'une capacité de 7).

Par ailleurs, entre les tâches, un niveau de priorité est défini permettant de classer les tâches, tout en respectant la priorité des interruptions (ou évènements). Lors de l'arrivée d'une nouvelle tache, celle-ci sera placée dans la file d'attente en fonction de sa priorité (plus elle est grande, plus le placement est proche de la sortie). Dans le cas ou la file d'attente est pleine, la tâche dont la priorité est la plus faible est enlevée de la FIFO.

6. NesC

Le développement dans TinyOS s'effectue à travers NesC. Le langage NesC [51] est une extension du langage de programmation C qui été désigné pour faciliter l'implémentation des structures et des composants TinyOS.

Dans le concept de NesC il y a une séparation entre la construction et la composition. Les programmes sont construits à l'intérieur des composants qui sont ensuite relié entre eux pour former toute une application. Les composants NesC utilise et fournie des interfaces bidirectionnelles qui représente les seuls points d'accès pour les composants TinyOS.

Le langage NesC [50][51] possède trois types de fichiers : *module, configuration et interface* :

- ***module :*** ils constituent les briques élémentaires de code et implémentent une ou plusieurs interfaces. Une application peut faire appel à des **fichiers de configuration** pour regrouper les fonctionnalités des modules ;

- ***configuration* :** c'est un fichier qui permet de faire le lien entre tous les composants ;

- ***interfaces*** : c'est un fichier décrivant les commandes et les évènements proposés par le composant qui les implémente. L'utilisation des mots clefs « Use » et « Provide » au début d'un composant permet de savoir respectivement si celui-ci fait appel à une fonction de l'interface ou redéfini son code.

La Fig.4.2 présente la relation qui peut exister entre deux composants.

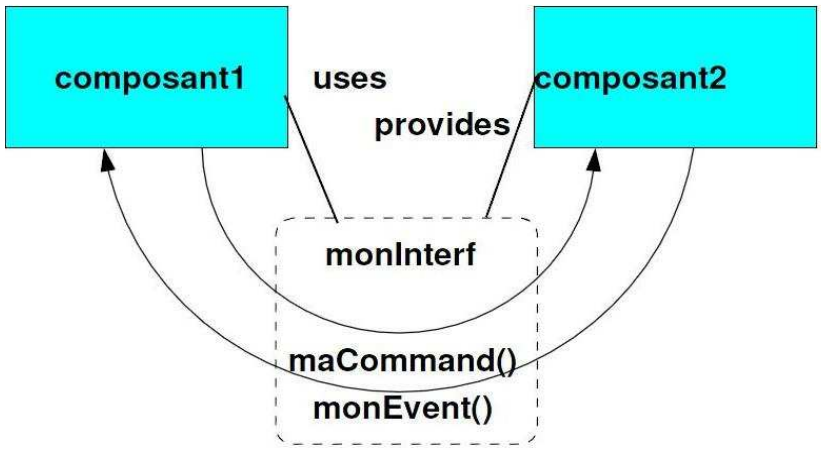


Fig.4.2 Exemple de relation entre composants

En plus NesC supporte aussi le modèle de concurrence, qui est basée sur le concept d'exécution jusqu'à la fin sans interruption ("run-to-completion'') pour les taches sauf exception pour les évènements.

6.1 Compilation

Les capteurs possèdent en général la même architecture et sont programmables via le langage NesC. Cependant plusieurs firmes fabriquent de tels capteurs, ce qui implique quelques différences qui sont palliées par le compilateur de NesC appelé *ncc*.
Pour effectuer une compilation, les fichiers doivent se situer dans le même répertoire contenant aussi un *makefile*.

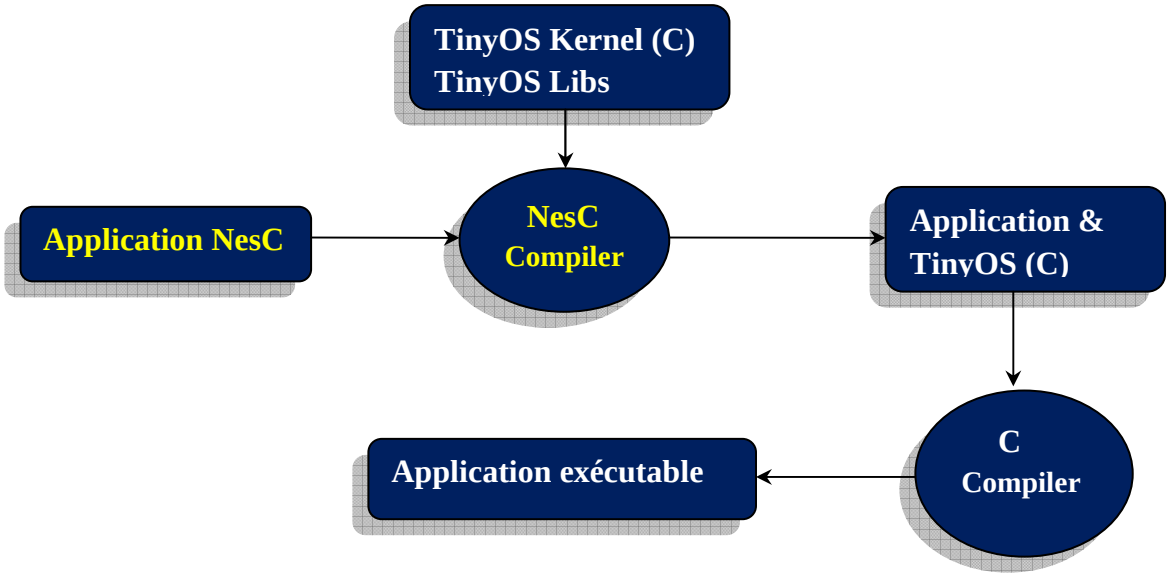


Fig.4.3 Processus de compilation

7. TOSSIM : Simulateur de TinyOS

TOSSIM [53] est un simulateur discret basé sur la programmation par événement et qui a été conçu et désigné pour simuler les réseaux de capteurs qui utilise la plateforme TinyOS.

Le principale but de TOSSIM est de créer une simulation très proche de ce qui ce passe dans ces réseaux dans le monde réel. Construit avec *"make pc"*, la simulation fonctionne en natif sur un ordinateur de bureau ou ordinateur portable. TOSSIM peut simuler des milliers de nœuds simultanément. Chaque mote dans une simulation exécute le même programme TinyOS.

TOSSIM simule le comportement des applications de TinyOS à un niveau très bas. Le réseau est simulé au niveau des bits et chaque interruption dans le système est capturée.

TOSSIM fourni deux modèles de radios pour la communication. Le modèle par défaut est celui « *simple* ». Les paquets sont transmis dans le réseau sans erreur et ils sont reçus par chaque nœud. Avec ce modèle il est ainsi possible que deux nœuds différents peuvent envoyer un paquet en même temps avec la conséquence que ces deux paquets seront alors détruit à cause du chevauchement des signaux. Le deuxième modèle est le modèle « *lossy* ». Dans ce modèle les nœuds sont placés dans un graphe direct formé d'un couple *(a, b)* ce qui signifie qu'un paquet envoyé par le *nœud a* peut être reçu par le *nœud b*. TOSSIM est équipé aussi d'un simulateur graphique *TinyViz*. Cette application est équipée par plusieurs *API plu-gins* qui permet d'ajouter plusieurs fonctions à notre simulateur comme par exemple contrôler les entrées de notre radio ou bien suivre la dépense d'énergie en utilisant un autre simulateur

qui s'appelle *PowerTOSSIM*. Le schéma ci-dessous montre l'architecture de TOSSIM :

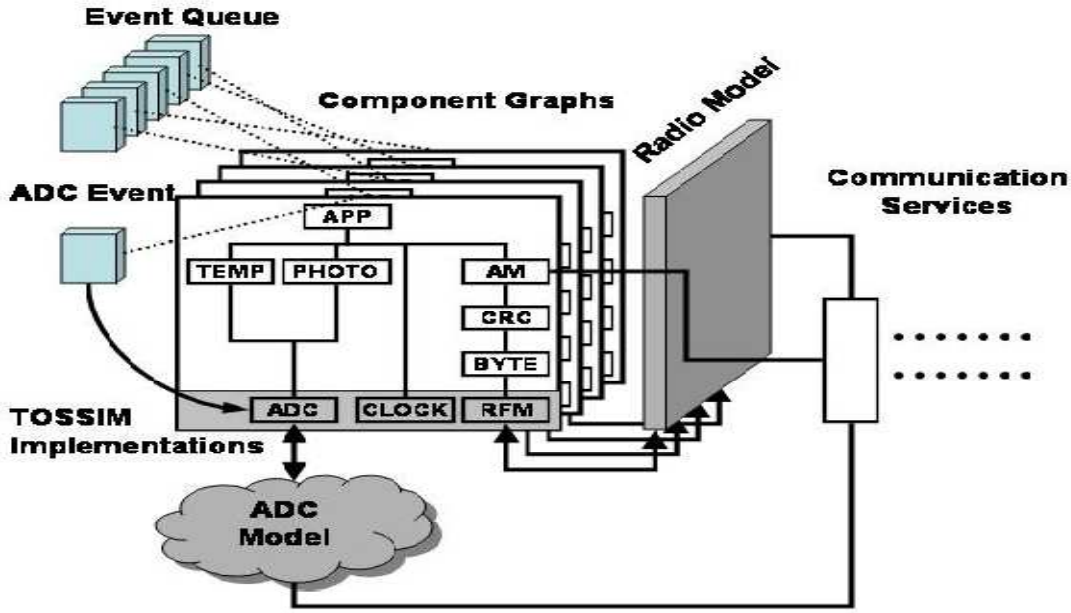


Fig.4.4 Architecture de TOSSIM [54]

7.1 Caractéristiques et avantages de TOSSIM

L'objectif principal de TOSSIM [55] est de fournir une simulation de haute fidélité des applications TinyOS. Pour cette raison, il se concentre sur la simulation TinyOS et son exécution, plutôt que de simuler le monde réel. Au lieu de compiler une application TinyOS sur un mote, les utilisateurs peuvent compiler dans le framework TOSSIM, qui fonctionne sur un PC. Cela permet aux utilisateurs de déboguer, tester et analyser des algorithmes dans un environnement contrôlé et reproductible.

Comme TOSSIM fonctionne sur un PC, les utilisateurs peuvent examiner leur code en utilisant TinyOS débogueurs et autres outils de développement. TOSSIM fournit la sortie *run-time* configurable de débogage, ce qui permet à un utilisateur d'examiner l'exécution d'une demande de différents points de vue sans avoir à recompiler.

7.2 Limitations de TOSSIM

Bien que TOSSIM [55] puisse être utilisé pour comprendre les causes des comportements observés dans le monde réel, il ne saisit pas tous, et ne doit pas être utilisé pour une évaluation absolue. TOSSIM n'est pas toujours la solution de simulation la plus juste; comme toute simulation, il fait plusieurs hypothèses, en se concentrant sur les décisions de certains comportements précis tout en simplifiant d'autres.

L’expérimentation et l’essai des réseaux de capteur est difficile. TOSSIM, un simulateur de TinyOS, permet aux utilisateurs d'exécuter et de tester des algorithmes, les protocoles et applications dans un environnement contrôlé et reproductible. Toutefois, par elle-même une simulation TOSSIM est statique. Au lieu de comportements tels que la modélisation de mouvement ou de lectures du capteur qui change, TOSSIM fournit une base de commande socket API pour d'autres programmes pour le faire. D'une main, ce qui maintient TOSSIM simple et efficace, d'autre part, il met le fardeau de l’écriture complexe du monde réel sur des modèles de l'utilisateur.

8. TinyViz

8.1 Introduction TinyViz

Pour une compréhension moins complexe de l’activité d’un réseau, TOSSIM peut être utilisé avec une interface graphique, TinyViz, permettant de visualiser de manière intuitive le comportement de chaque capteur au sein du réseau.

TinyViz est une application graphique qui donne aperçu de notre réseau de capteur à tout instant, ainsi que des divers messages qu’ils émettent. Il permet de déterminer un délai entre chaque itération des capteurs afin de permettre une analyse pas à pas du bon déroulement des actions.

La classe principale de TinyViz [53] est un fichier *jar*, dans *tools/java/net/tinyos/sim/tinyviz.jar*. TinyViz peut être attaché à une simulation en cours. Il n'est pas réellement un outil de visualisation; au contraire, c'est un framework dans lequel les plug-ins peuvent fournir la fonctionnalité désirée.

8.2 Avantages de TinyViz

Avec TinyViz, les utilisateurs peuvent interagir avec une simulation à travers un panneau graphique GUI, par le réglage d’options. Ces actions peuvent être difficiles à reproduire exactement. En outre, TinyViz peut (comme son nom l'indique) visualiser ce qui se passe dans le réseau. Les utilisateurs peuvent écrire des *plug-ins* TinyViz en Java pour étendre la fonctionnalité de l'interface graphique et de délivrer des commandes à TOSSIM. Toutefois, en raison les utilisateurs doivent pré-compilation leurs plug-ins, ce qui permet seulement une interactivité limitée.

L’utilisation de TinyViz, vous permet facilement de tracer l'exécution d'applications TinyOS, des points d'arrêt lorsque des événements intéressants se produisent, de visualiser des messages radio, et de manipuler la position virtuelle de connectivité radio de motes.

Le schéma ci-dessous, donne un aperçu de la façon dont le TinyViz et TOSSIM sont liées :

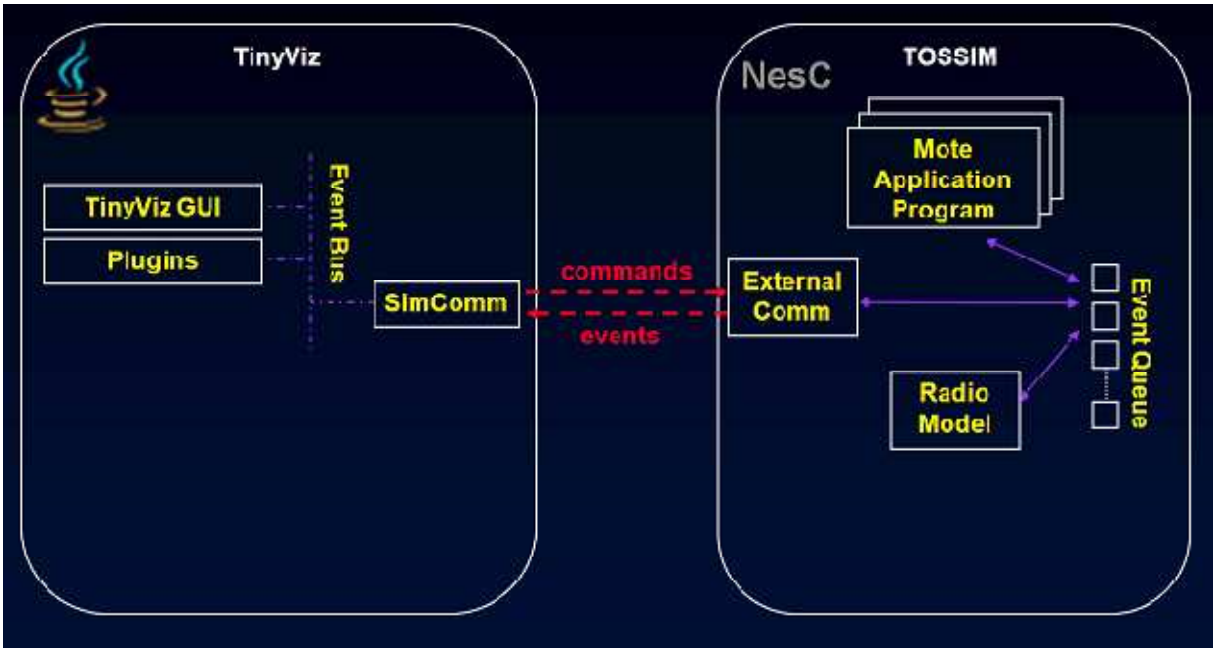


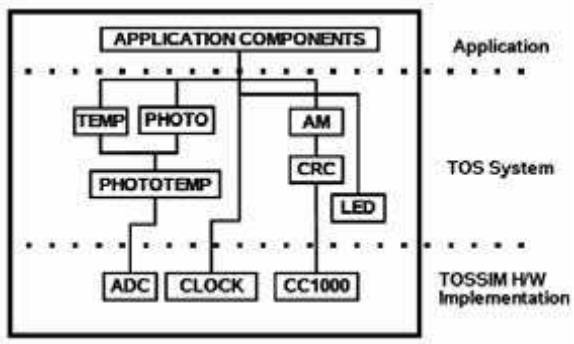
Fig.4.5 TinyViz et TOSSIM

9. PowerTOSSIM

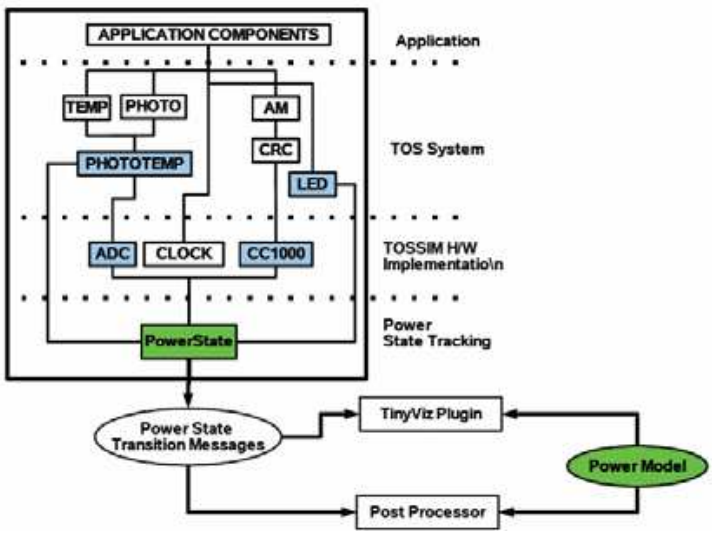
Le simulateur TOSSIM [56] n'a pas la capacité de vérifier le taux d'énergie dissipée pendant l'exécution des applications. Cependant, le besoin de vérifier la consommation énergétique dans un RCSF a un intérêt primordial. L'université de Harvard a conçu le simulateur PowerTOSSIM qui surmonte ce problème. Ce nouveau simulateur est intégré dans TOSSIM. Il permet de générer un fichier de l'extension ***.trace*** qui enregistre les détails de la simulation comme l'énergie consommée dans le réseau.

PowerTOSSIM est l'extension de TOSSIM qui contient un modèle de consommation d'énergie. Pour les valeurs de la consommation, les auteurs se sont basés sur le Mica2. Ils ont supposé connaitre les consommations des différents composants de ce nœud suivant leurs états. Grâce au modèle de simulation basé sur TinyOS, on connaît immédiatement l'état des composants puisque les changements d'états correspondent à des événements dans TinyOS et donc dans TOSSIM. Par conséquent, PowerTOSSIM permet d'avoir des résultats très proches de la réalité sur le plan de la consommation d'énergie.

Enfin, les simulateurs TOSSIM et PowerTOSSIM ne conviennent que pour des applications écrites en TinyOS.



TOSSIM



PowerTOSSIM

Fig.4.6 Architecture de TOSSIM et PowerTOSSIM

Partie II : Implémentation et Résultats

1. Implémentation :

Notre choix s'est porté sur les deux algorithmes de compression de données RLE et K-RLE. Car les contraintes mémoires dans TinyOS (2 ko de mémoire RAM) rendent impossible l'évaluation de l'algorithme S-LZW, c'est pour cette raison que nous allons étudier les performances des deux algorithmes (RLE et K-RLE) en termes de conservation d'énergie et de taux de compression sur le même système pour pouvoir les comparer avec différentes statistiques des données.

Etape1 :

1. Implémentation d'un algorithme qui calcule la température ;
2. Calculer l'énergie consommée par un nœud capteur.

Etape 2 :

1. Implémentation d'un algorithme qui calcule la température avec la compression des données (températures captées) avec les algorithmes de compression RLE et K-RLE avant de les envoyer.
2. Calculer l'énergie consommée par les nœuds capteurs avec utilisation des algorithmes de compressions de données.
3. Calculer le taux de compression et bande passante économisée.

Etape 3 : Test et résultats

Comparaison des résultats de la consommation d'énergie par les nœuds capteurs avec et sans la compression des données (températures) et déduire les impacts de la compression de données sur l'énergie dans les réseaux de capteurs.

Détail des étapes :

1.1 Etape 1 : calcule de température

Nous avons utilisée le tableau de températures réelles construit grâce aux données collectées sur le site Weather Underground [46]. Ces valeurs seront sauvegardées dans un fichier.

L'ADC12 est un convertisseur analogique-numérique 12 bit de haute performance dont les caractéristiques inclus un module de capture de températures. La fonction de transfert du capteur de température est la suivante :

VTEMP = 0.00355 *(TEMPC) + 0.986.

TEMPC : représente la température convertie par CAN ;

VTEMP : représente la vraie température.

Grâce à cette fonction de transfert, nous avons simulé les séquences de mesures de températures en convertissant les archives de Weather Underground [46] en valeurs hexadécimales sur 12 bits telles que les auraient mesurées l'ADC.

La Fig.4.7 représente l'architecture de l'application de détection de Température

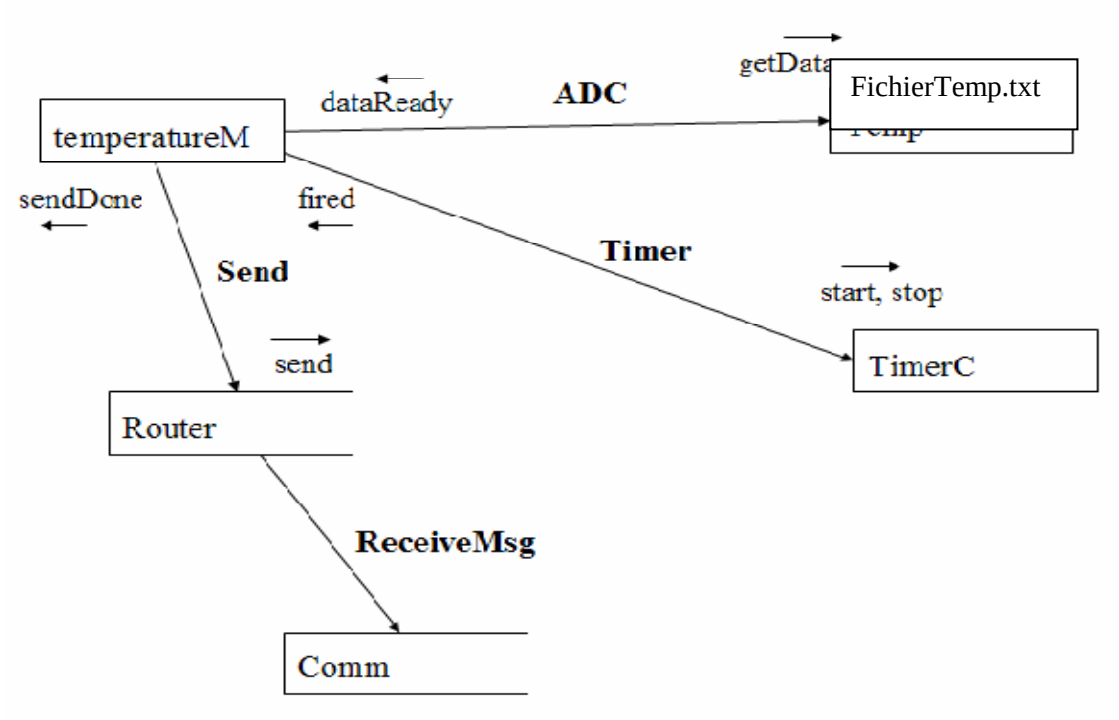


Fig.4.7 Architecture de l'application de détection de Temperature

1.2 Etape 2 : Implémentation de RLE et KRLE

Les algorithmes RLE et K-RLE sont présentés de façon à optimiser le gain de l'énergie dans les réseaux de capteurs. Nous avons simulé un nœud capteur qui compresse les données en appliquant les algorithmes de compression (RLE puis K-RLE) avant de les envoyer vers un autre nœud. Le but de la simulation de ces algorithmes est de calculer le gain de l'énergie consommée par un nœud, que peuvent lui faire gagner, et cela en trois étapes, puis comparer les résultats:

- En envoyant les températures captées sans aucun traitement vers un *puit* ;
- En envoyant les températures captées en les compressant avec l'algorithme de compression RLE ;

- Et enfin en envoyant les températures captées vers un *puit* en les compressant avec l'algorithme de compression K-RLE.

1.2.a Implémentation de RLE

Principe : Si un item *d* apparaît *n* fois consécutivement dans le flux de données d'entrée, nous remplaçons les *n* occurrences avec la simple paire *nd*.

La figure suivante Fig.4.8 montre le diagramme de déroulement de l'algorithme de compression de données RLE.

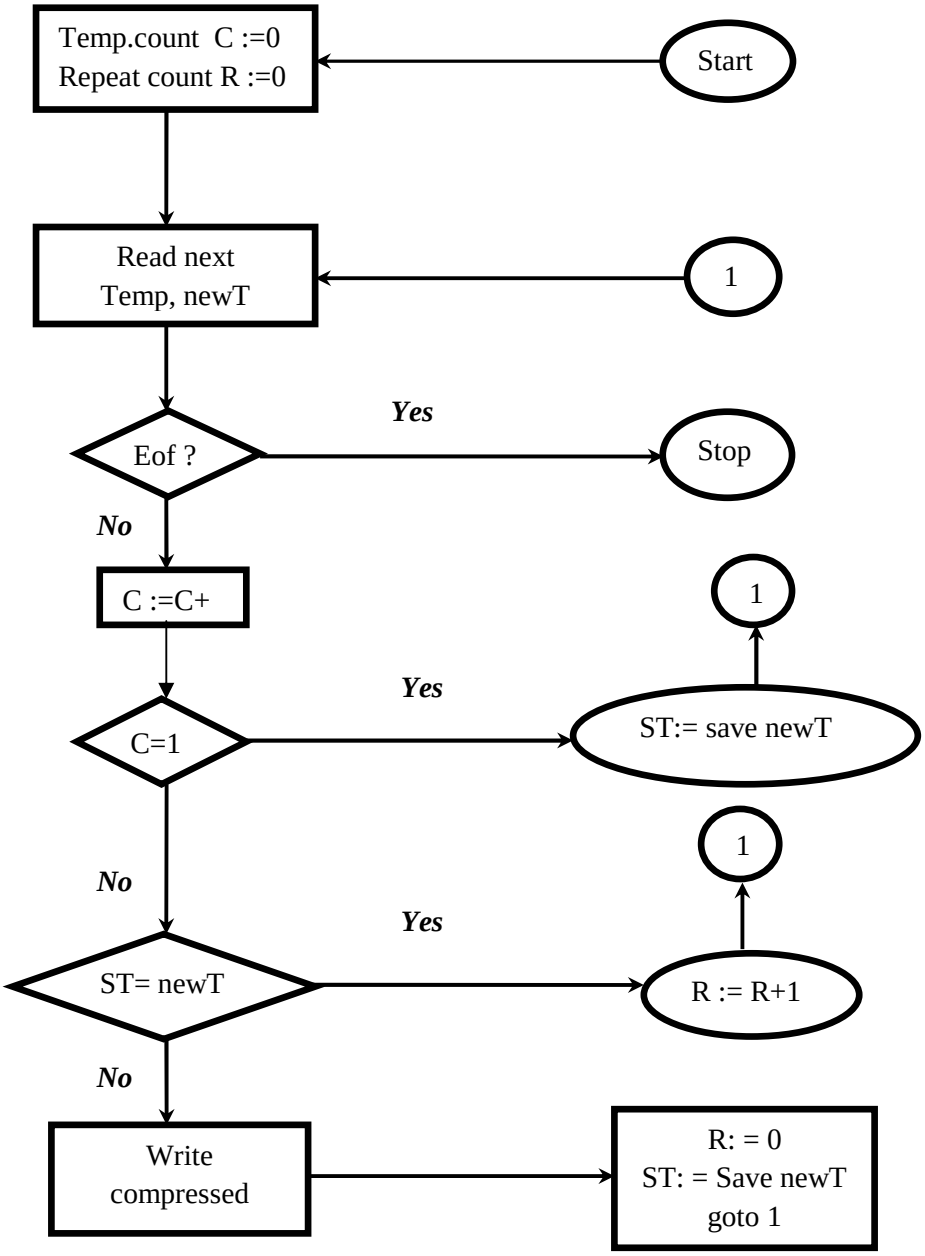


Fig.4.8 Algorithme de compression RLE

Le code RLE : le code de la fonction RLE est :

```
task void processData(){
  int16_t i, nbrepet=1, position=0;
  atomic{
   for (i=0; i<=NBR_MAX; i++){
    if (tab[i]==tab[i+1]){
              while (tab[i]==tab[i+1]){
               nbrepet=nbrepet+1;
               i++;
              }
             tabcompress[position]=nbrepet;
             tabcompress[position+1]=tab[i];
             }
   else{
    tabcompress[position]=tab[i];
             position++;
             i++;
   }
   dbg(DBG_USR1, ">>> temperatures avant la compression =%d\n",tab);
   dbg(DBG_USR1, ">>> temperatures compressées=%d\n",tabcompress);
   }  }
 }
 }
```

1.2.b Implémentation de K-RLE

Principe : Soit *K* un nombre, Si un item *d* ou un item entre *d-K* et *d+K* apparait *n* fois consécutivement dans le flux de données d'entrée, nous remplaçons les *n* occurrences avec la simple paire *nd*.

Rappelons que K est une précision de même unité que le flux de données d'entrée (degrés). Lorsque k est proche de zéro les résultats de RLE et K-RLE sont similaires.

Le Fig.4.9 montre le diagramme de déroulement de l'algorithme K-RLE :

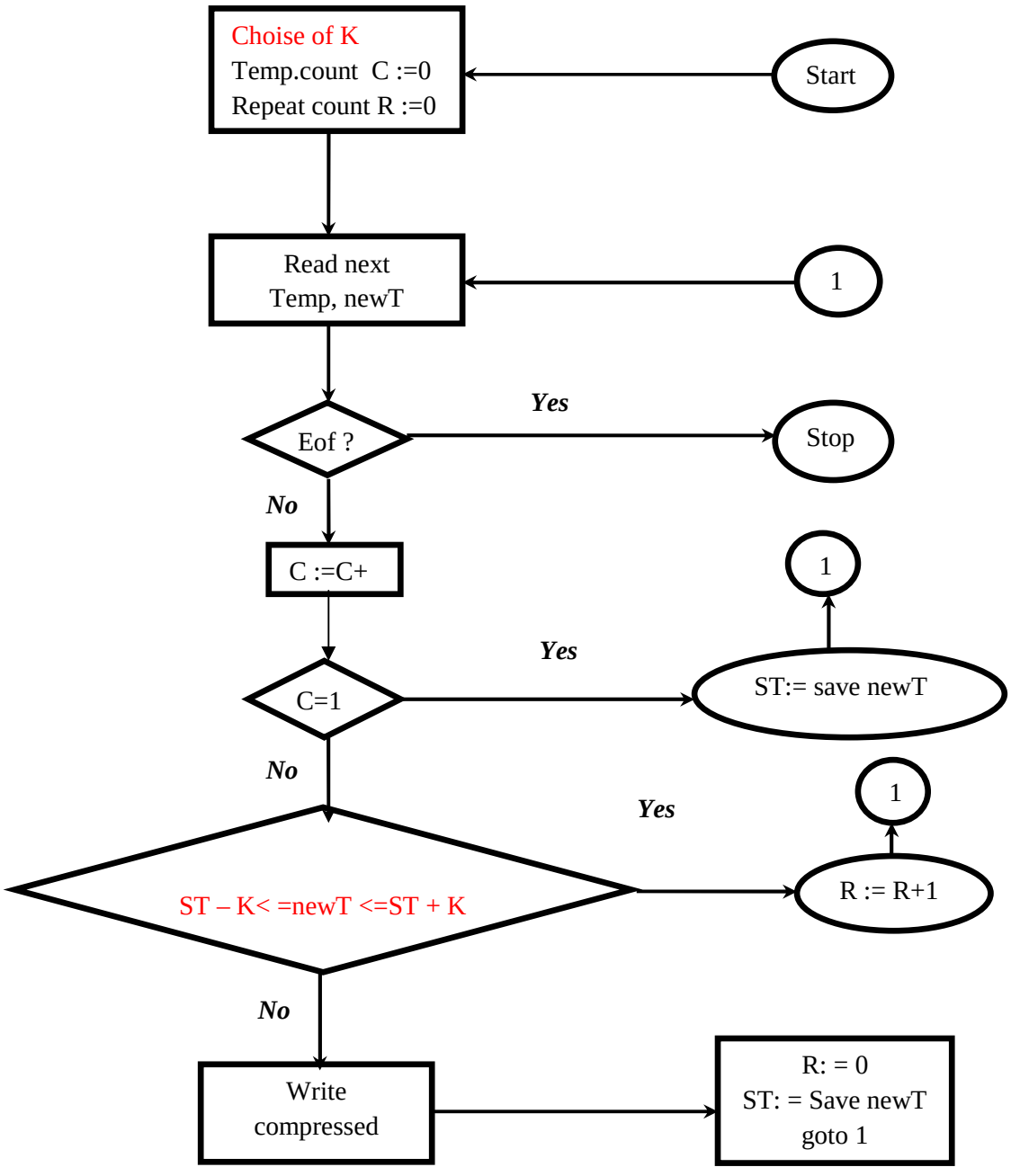


Fig.4.9 Algorithme de compression K-RLE

Le code KRLE : le code de la fonction RLE est :

```
task void processData(){
  int16_t ic, nbrepet=0, position=0;
  uint16_t d;
  atomic{
   while (i<=NBR_MAX){
            d=tab[i];
            while (((d-K)<=tab[i])&&(tab[i]<=(d+K))){
              nbrepet=nbrepet+1;
              i++;
             }
            if (nbrepet>1){
            tabcompress[position]=nbrepet;
            tabcompress[position+1]=d;
            position=position+2;
            nbrepet=0;
          } else{
    nbrepet=0;
    tabcompress[position]=tab[i-1];
    position++;
   }
  }
 }
}
```

1.2.c Communication :

Afin de permettre un flux de communication d'un nœud vers un *puit*, notre implémentation se base essentiellement sur la gestion des événements « émission » et « réception » en respectant le standard de NesC. Une consultation de la fiche technique [52] et divers exemple fournis avec la distribution de tinyOS et plus que nécessaire afin de mettre correctement en place un système de transmission.

Structure des messages :

Le paquet dans TinyOS est envoyé dans une structure appelée TOS_Msg, qui est contenue dans un champ « int8_t data[TOSH_DATA_LENGTH] ». Ce paquet va contenir données à transmettre (compressées dans le cas de RLE et K-RLE ou non compressées). Le message est représenté par une structure de données définie dans le fichier d'entête « IntMsg.h » comme ceci :

```
Enum{
NBR_MAX=23;
};

typedef struct IntMsg {

 float tabCompress[NBR_MAX];  //le tableau de données compressées et convertie

} IntMsg;
```

Envoi de messages :

Le fonction *SendData()* est caractérisée par le faite que c'est une tâche et non pas un événement et par conséquent elle doit être appelée avec l'instruction standard *post call SendData()* et implémentée avec le mot clé *void task.*

```
task void SendData(){
  int16_t k=0;
  IntMsg *message=(IntMsg *)data.data;
  if (sending_packet)
  atomic sending_packet=TRUE;
  for (k=0; k<=position; k++){
   message->tabCompress=tabcompress[k];
  if (call SendMsg.send(1,sizeof(IntMsg),&data)!=SUCCESS){
   sending_packet=FALSE;
   return;
  }
 }
  return;
 }
```

On remarquera dans le corps de la fonction qu'on fait appel à la primitive *SendMsg.send()* qui se charge d'envoyer le message. Le standard NesC fournit aussi une autre primitive *SendMsg.sendDone()* qui se définie comme étant un événement indiquant si une transmission a bien eu lieu ou pas. Les deux primitives sont fortement liées et, par conséquent, elles doivent être implémentées en même temps.

Le code de la primitive *SendMsg.sendDone()* :

```
event result_t SendMsg.sendDone(TOS_MsgPtr msg, result_t success){
 if (sending_packet && msg == &data)
  atomic sending_packet=FALSE;
  return SUCCESS;
 }
```

Réception de message :

Le fonction réception *ReceiveMsg.receive()* est un événement et donc doit être implémentée avec le mot clé *event* :

```
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr recv_packet){
  IntMsg *message=(IntMsg *)(&recv_packet -> data);
  return recv_packet;
 }
```

2. Simulation, évaluation et interprétation :

Pour évaluer les performances des algorithmes de compression de données (RLE et K-RLE), nous avons procédé à la comparaison entre ces deux derniers avec une simulation de transmission de données (température) sans aucune technique appliquée. Pour cela nous avons effectué des simulations avec les mêmes paramètres et métriques.

2.1 Métriques à évaluer :

Pour pouvoir mesurer les performances des algorithmes de compression de données dans les réseaux de capteurs, il est commode de mesurer certaines métriques qui sont :

2.1.1 Consommation énergétique :

Nous nous sommes intéressées essentiellement à la consommation d'énergie des nœuds puisqu'elle constitue un paramètre primordial pour la détermination de la durée de vie d'un RCSF. Nous analysons donc l'impact du mécanisme de compression intégré par les deux algorithmes RLE et K-RLE sur l'énergie consommée par rapport au transfert direct des données (sans compression).

Pour ce faire, nous prenons comme critère l'énergie moyenne consommée par chaque nœud du réseau.

2.1.2 Le taux de compression de données :

Les collisions dans les réseaux de capteurs constituent une des formes de dissipation d'énergie, due au heurtement de deux ou plusieurs paquets envoyés sur le réseau. Les retransmettre par la suite consomme de l'énergie, pour cela nous allons nous intéressé au taux

de compression donner par les deux algorithmes RLE et K-RLE, car sa peut permettre un gain en canal de communication.

2.2 Paramétrage de la simulation :

Avant de lancer les simulations, nous devons ajuster certains paramètres qui sont présentés par le tableau Tab.4.3 :

Paramètres du contexte de la simulation	
Nombres d'itérations (simulations)	**N : les résultats que nous allons présenter sont une moyenne de 5 simulations pour un même scénario**
Taille de paquets de données	**29 octets : c'est le paquet de transmission de TinyOS**

Tab.4.3 Paramétrage du contexte de simulation

3. Résultats et interprétation :

Dans cette partie, nous évaluons les métriques déjà cités et les comparons pour une transmission de données : sans compression et avec compression (RLE et K-RLE).

3.1 Le taux de compression :

3.1.a Le taux de compression de données par les algorithmes RLE et K-RLE

Dans cette partie, nous allons utiliser la formule du taux de compression en pourcentage pour estimer les performances des différents algorithmes de compression :

$$\tau\,(\%) = 100 * \left(1 - \left(\frac{\text{taille\_compressee}}{\text{taille\_originale}}\right)\right)$$

Nous comparons dans un premier temps RLE et K-RLE.

L'algorithme K-RLE introduit un nouveau paramètre qui est K. Nous allons donc dans un premier temps définir un cas pratique où l'utilisateur considère qu'une différence de 2° entre les différentes températures n'est pas critique (K = 2°). Lorsque K = 0°, il n'y a aucune différence entre RLE et K-RLE. K-RLE devient tout simplement RLE, c'est pour cette raison que un K proche de zéro a été choisi, soit K = 0,5° et les comparer par rapport à RLE.

La Fig.4.10 montre qu'il y'a une très grande différence entre RLE et 2-RLE en ce qui concerne le taux de compression.

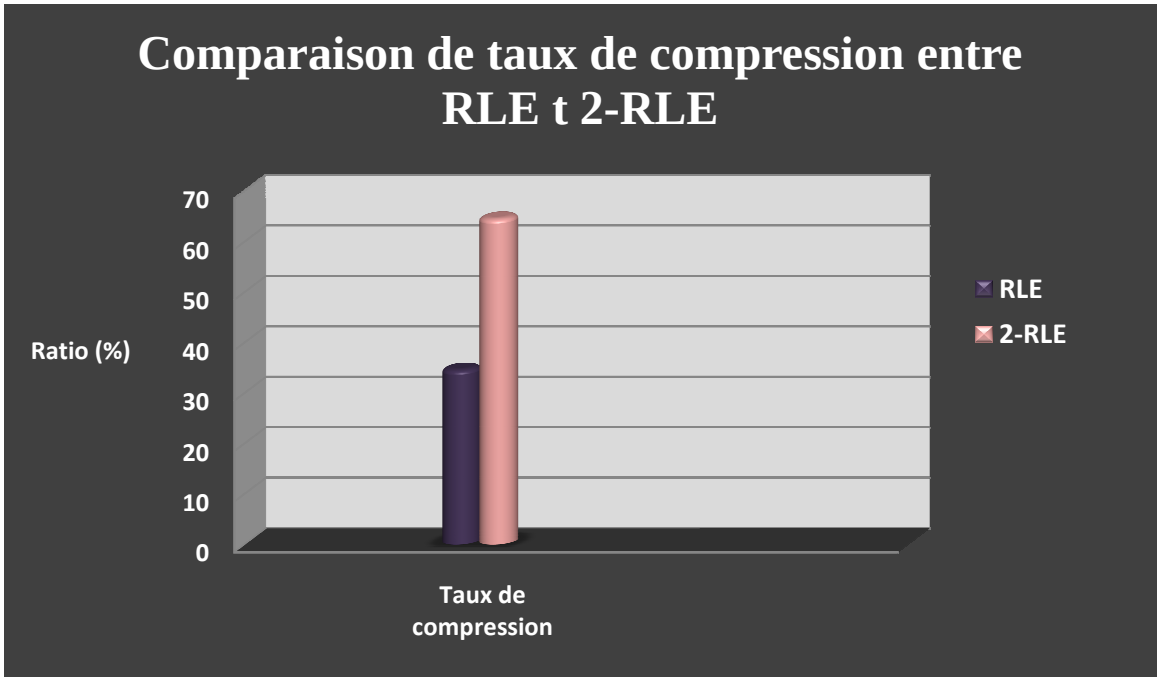


Fig.4.10 La comparaison du taux de compression de RLE et K-RLE

2-RLE offre de bien meilleur résultats que l'algorithme RLE. Tandis que le meilleur taux de compression de RLE et de 34,6%, le taux moyen pour 2-RLE est 64,5%.

La Fig.4.11 suivante montre que 2-RLE fournit de meilleurs résultats que RLE. 2-RLE offre des taux de compression meilleur que RLE, comme le montre les résultats précédents. Cependant, la comparaison entre 1/2-RLE et RLE montre que les résultats des deux algorithmes sont les mêmes. De là, nous pouvons conclure en disant que, tandis que K-RLE peut améliorer le taux de compression de RLE de manière considérable, le choix de K reste un facteur essentiel.

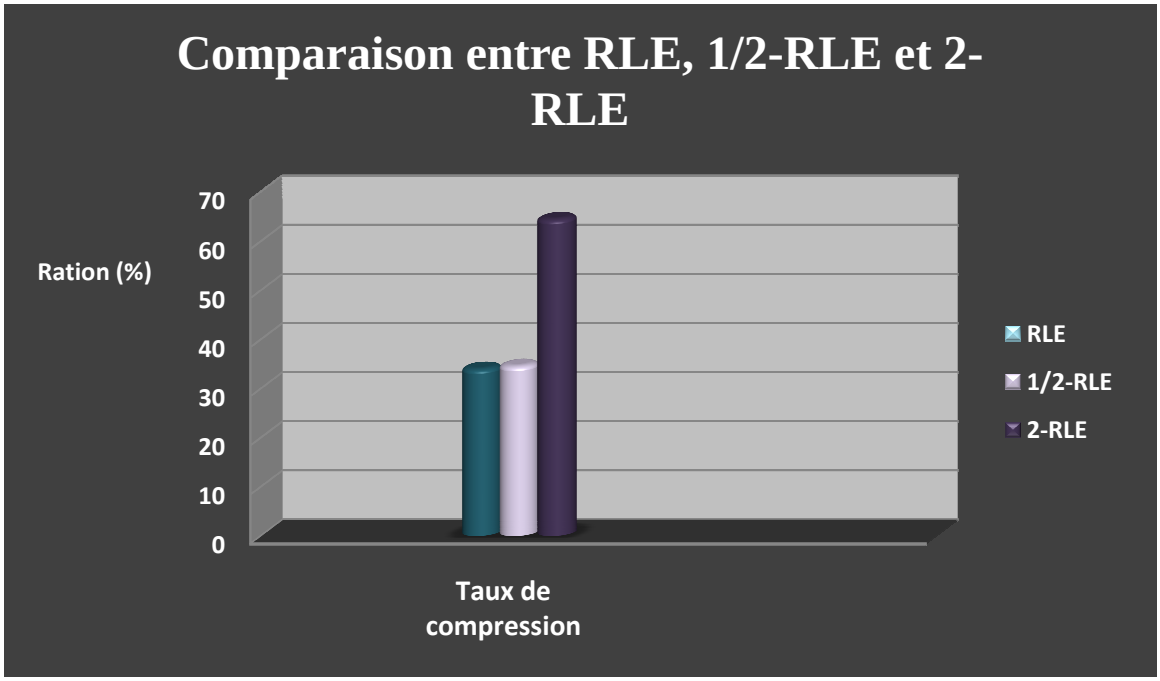


Fig.4.11 La comparaison du taux de compression de RLE, 1/2-RLE et 2-RLE

Nous pouvons continuer à accroître le taux de compression de K-RLE en augmentant la valeur de K, mais il faut bien noter que l'efficacité de l'algorithme K-RLE se fait au détriment de la qualité des données, car plus K est grand, plus la compression est efficace mais plus la différence entre les données originales et les données décompressées est importante, ce qui est une particularité des algorithmes de compression avec pertes. Cet algorithme se comporte comme JPEG qui est aussi ajustable et où l'efficacité de compression et la qualité des données évoluent de manières antagonistes.

Cependant, K-RLE ne perd pas les données en tant que tel mais plutôt les modifie avec une précision de K.

3.2 Consommation énergétique :

Pour avoir des tests en consommation d'énergie, nous avons eu recours au *fichier trace* généré par le simulateur PowerTOSSIM.

Pour évaluer l'énergie totale E_{Totale} consommée, nous allons prendre en compte l'énergie de la compression $E_{compression}$ (cpu), et l'énergie de la transmission $E_{Transmission}$ (radio):

$$E_{Totale} = E_{Compression} + E_{Transmission}$$

Les cas de figures suivantes peuvent se présenter :

3.2.a Consommation d'énergie par un nœud

Dans cette section, nous allons évaluer et comparer la consommation énergétique d'un nœud capteur, sans compression, et en utilisant les deux algorithmes RLE et K-RLE. La fig.4.12 détaille la consommation énergétique d'un nœud, sans compresser les données, en les compressant avec RLE, puis avec K-RLE.

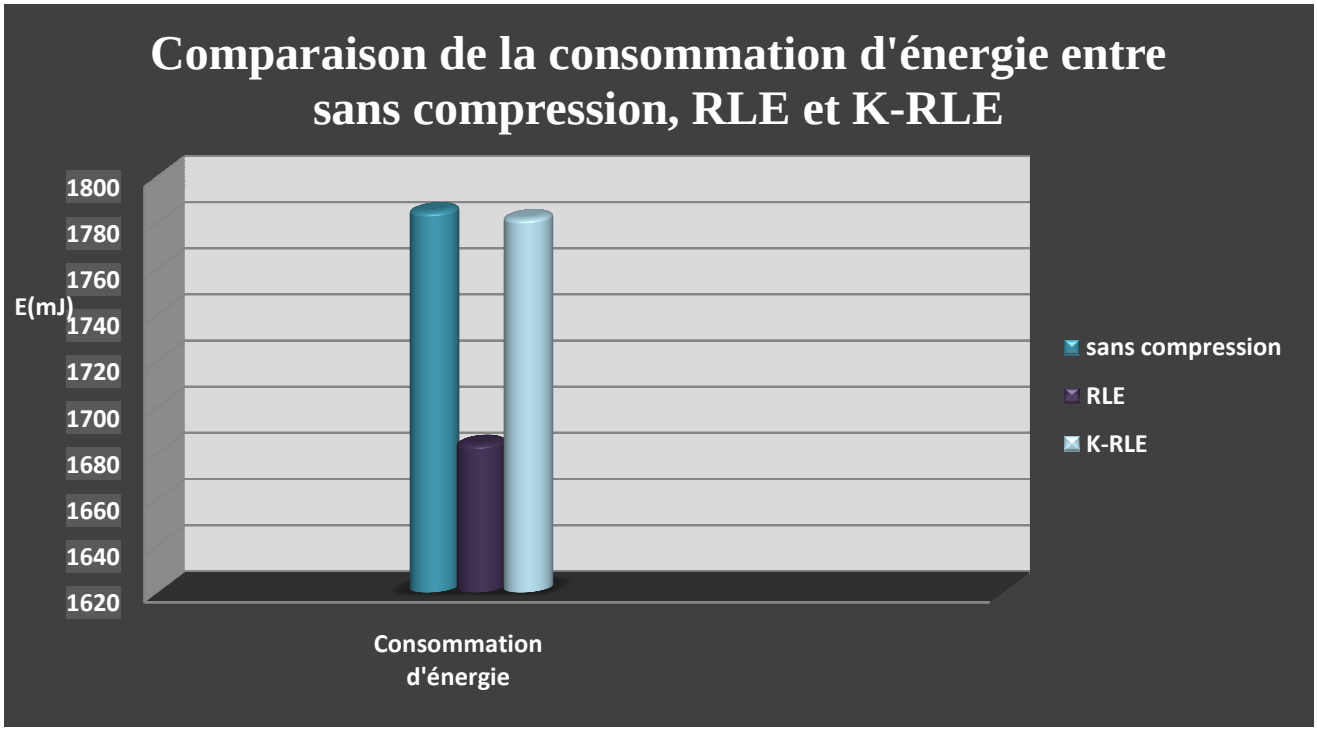


Fig.4.12 Comparaison de la consommation d'énergie entre sans compression, RLE et K-RLE

Nous constatons que la compression de donnée peut être utile dans certains cas, et peut permettre un gain d'énergie, et contrario dans d'autre cas. En effet, l'algorithme RLE étant simple, consomme le moins d'énergie et donc permet un gain énergétique au dispositif par rapport à un envoi de données sans compression. Par contre l'algorithme K-RLE consomme presque la même quantité d'énergie qu'avec la transmission sans compression.

3.2.b Consommation d'énergie par la CPU et le tranceiver d'un nœud

Dans cette partie nous allons comparer la consommation énergétique des algorithmes du coté cpu et tranceiver (radio).

Nous observons dans la figure ci-dessous, que RLE permet, à la radio de consommer moins d'énergie, cela est du à la réduction de la quantité de données envoyées, et à la cpu et cela est

du à la simplicité de RLE. Contrario K-RLE consomme plus d'énergie (radio et cpu) que RLE.

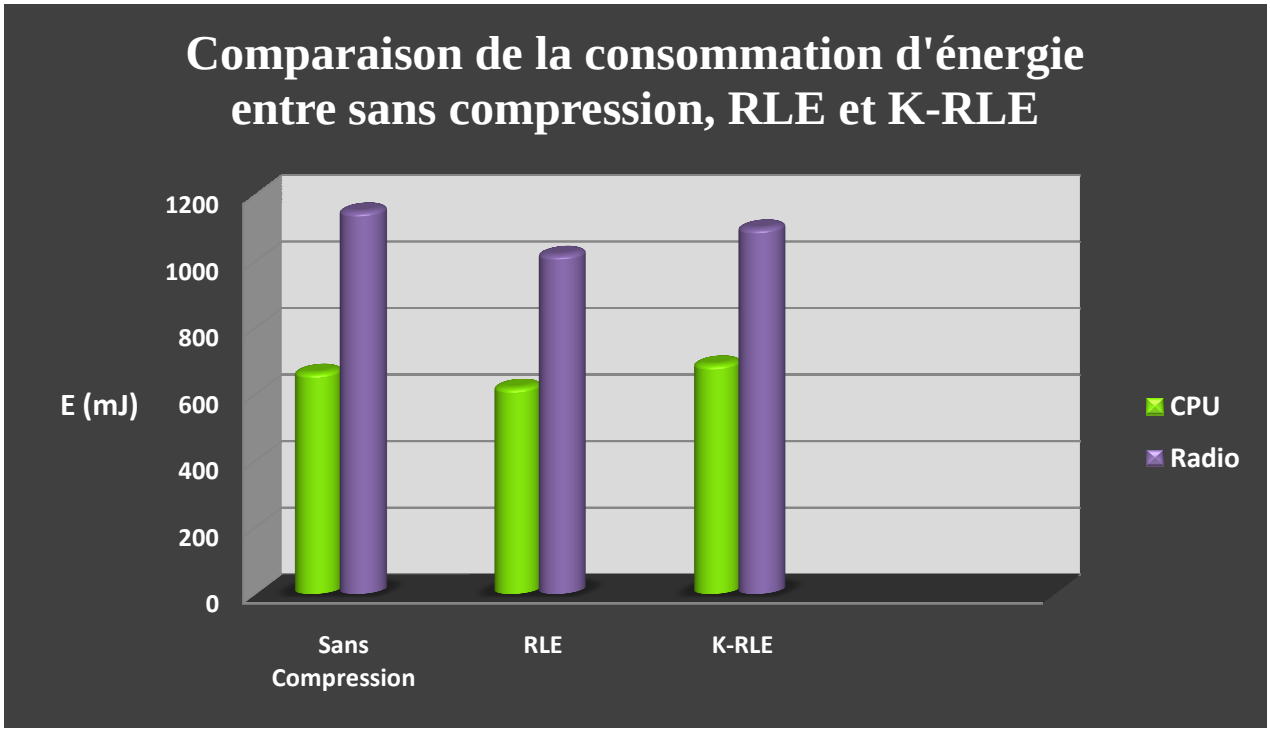


Fig.4.13 Comparaison de la consommation d'énergie entre sans compression, RLE et K-RLE (radio et CPU)

K-RLE consomme autant d'énergie que la transmission sans compression mais il reste un algorithme efficace en taux de compression.

3.2.c La variation de la consommation énergétique d'un nœud dans le temps

La comparaison de la consommation énergétique dans le temps, pour un nœud capteur, et cela dans le cas d'utilisation du meilleur algorithme de compression en terme de consommation d'énergie qui est RLE, et la transmission des données sans compression nous donne les résultats suivants :

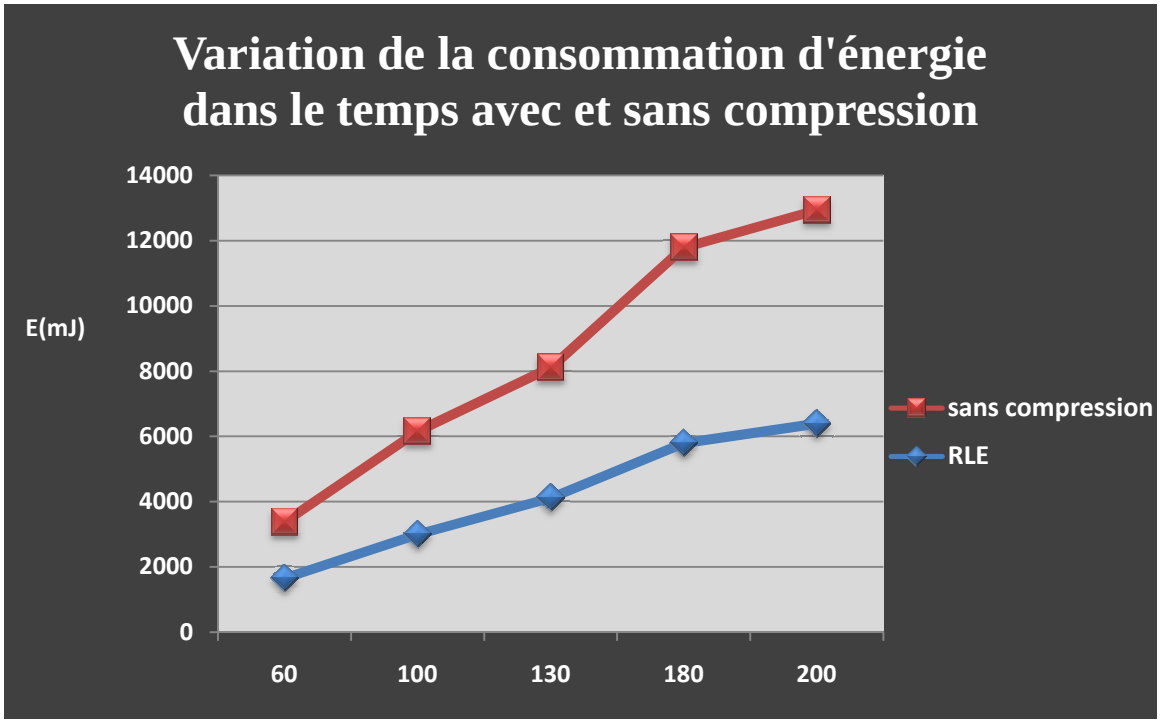


Fig.4.14 Variation de la consommation d'énergie dans le temps avec et sans compression

Nous constatons que la consommation d'énergie par le nœud dans le cas de la compression augmente d'une façon constante dans le temps, contrairement à la consommation qui augmente très rapidement dans le cas de l'envoi sans compression, et cela peut mener à l'épuisement du nœud très tôt.

L'application de l'algorithme de compression RLE peut permettre une durée de vie plus longue pour le nœud capteur.

4. Comparaison des performances des deux techniques « Compression de Données » et « Mise en Veille »

Après avoir évalué et comparer les résultats de consommation d'énergie de la compression de données des deux algorithmes RLE et K-RLE, nous allons faire une comparaison entre la technique de « compression de données » et la technique de « Mise en veille ». Pour ce faire, nous allons comparer les résultats obtenues précédemment avec ceux de la mise en veille obtenus par [57].

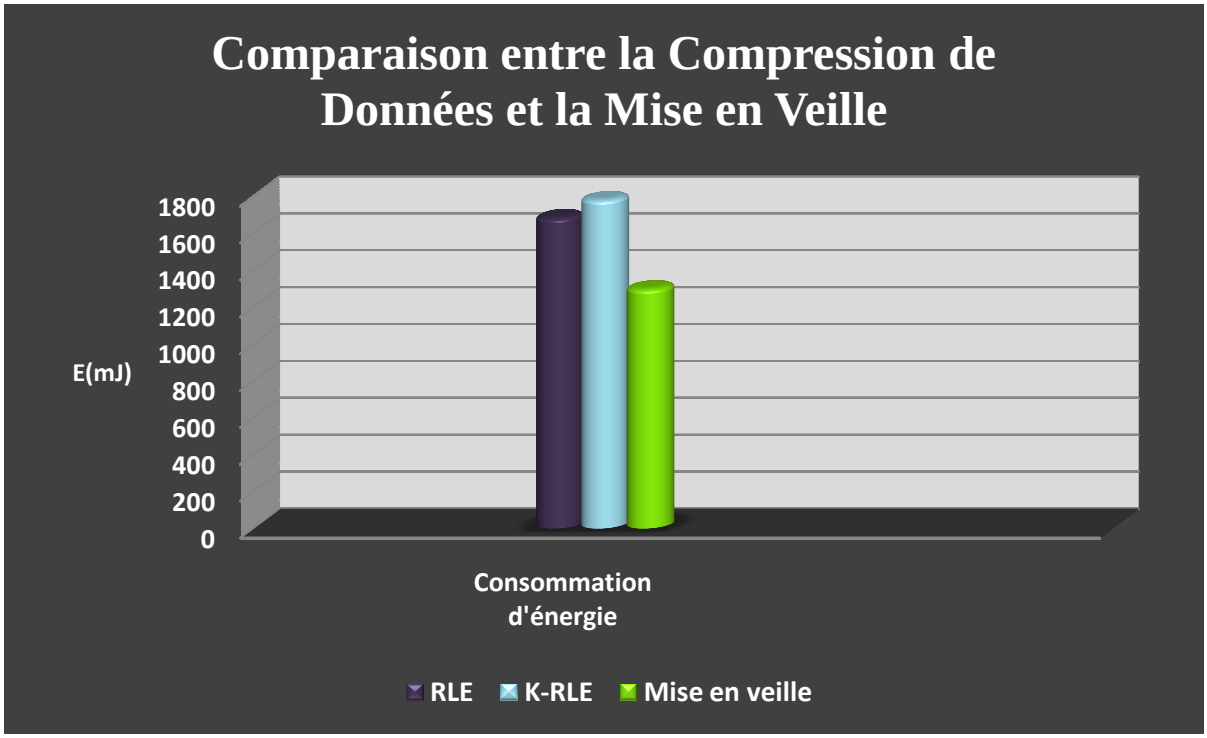


Fig.4.15 Comparaison entre la compression de données et de la mise en veille

La technique de mise en veille consomme moins d'énergie que la compression de données, cependant la mise en veille de la radio d'un capteur pendant un moment peut engendrer des pertes de paquets envoyés dans le réseau [57]. A étendre, on peut voir la combinaison des deux techniques.

5. Synthèse sur K-RLE :

K-RLE est un algorithme de compression inspiré de RLE (introduit un nouveau paramètre K), très simple et rapide, mais consomme plus d'énergie et modifie les données. Mais il reste un algorithme efficace en taux de compression (65% en moyen). Réduire le volume d'information à envoyer de 64 %, revient à économiser 64 % de la bande passante, ce qui réduit la charge du réseau et par conséquent le nombre de collisions qui est l'un des facteurs de dissipation de l'énergie. On peut donc s'attendre à une baisse de la consommation en utilisant cette technique dans le cas d'un réseau à forte charge.

6. Conclusion :

Dans ce chapitre nous avons présenté l'implémentation ainsi que l'évaluation des deux algorithmes de compression de données RLE et K-RLE. Le système d'exploitation TinyOS est utilisé. Avec une programmation entière en NesC, et une simulation avec TOSSIM et PowerTOSSIM.

Par la suite, nous avons comparé les performances de RLE et K-RLE sur le plan de la consommation d'énergie et le taux de compression. Nous avons constaté, la compression de données dans certains cas (RLE) associée une économie d'énergie par rapport à une transmission sans compression. K-RLE inspiré de RLE, est un algorithme avec perte dont le niveau de perte dépend du paramètre K, consomme plus d'énergie que RLE, mais donne de meilleurs taux de compression, ce qui nous permet de gagner en canal de communication.

Cette simulation a permis de mettre en lumière l'antagonisme entre l'efficacité de compression et la qualité des informations pour les algorithmes de compression de données.

Enfin, nous avons fait une comparaison vis-à-vis de la consommation d'énergie entre les deux techniques de conservation d'énergie, la *compression de données* et la *mise en veille.*

Conclusion et Perspectives

Conclusion

Les réseaux de capteurs, nouvelle thématique de recherche innovante, vient enrichir le domaine des réseaux et télécommunications. La dimension de ce nouveau type de réseau capable de collecter des informations sur un environnement et de les transmettre à une station collectrice, va permettre de surveiller différentes zones qui peuvent être hostiles à l'homme. La pluridisciplinarité de ce domaine pousse à la convergence des compétences de plusieurs laboratoires qui ont parfois des centres d'intérêts différents mais qui se rejoignent sur cette thématique. La particularité de ces nouveaux systèmes distribués émergents est la capacité réduite des nœuds en terme de calcul, de mémoire, ou d'énergie. Cette dernière caractéristique est le point critique dans ces réseaux. Les recherches dans ce domaine se focalisent donc dans la majorité des cas sur la gestion énergétique.

Pour optimiser la collecte et le transfert de données dans la zone de déploiement de ce réseau de capteurs, nous avons étudié la technique de compression de données permettant de réduire la taille des informations transmises tout en respectant la cohérence des données et nécessitant une faible consommation énergétique. Les deux algorithmes de compression de données étudiées, RLE et K-RLE, comparés à une transmission de données sans compression, ont montré leur efficacité en terme de taux de compression et de consommation énergétique, mais qui dépendent de la variation de températures. Nous avons implémenté et simulé un algorithme de compression de données avec perte, s'inspirant de l'algorithme RLE, en introduisant un nouveau paramètre K.

Cette proposition implémentée est une nouvelle technique permettant d'utiliser un réseau de capteurs avec des qualités supérieures en terme de performance et de consommation d'énergie.

Les différentes problématiques abordées et la multitude d'applications qui peuvent y être rattachées ou encore la pluridisciplinarité de cette thématique procure à ce domaine un avenir prometteur.

Perspectives

Les applications basées sur des observations sans contraintes temporelles permettent l'optimisation du canal de communication en utilisant des techniques de compression de données. La caractérisation du paramètre K de l'algorithme K-RLE reste encore une voie intéressante à explorer.

Nous avons aussi fait une comparaison entre les deux techniques de conservation d’énergie, la compression de données et la mise en veille. Il serait peut être intéressant de fusionner les deux techniques pour obtenir un meilleur gain énergétique.

Bibliographie

[1] Cristian Duran-Faundez (2009), « Transmission d'images sur les réseaux de capteurs sans fil sous la contrainte de l'énergie », Université Henri Poincaré, Nancy 1, Thèse.

[2] Rahim KACIMI (2009), « Techniques de conservation d'énergie dans les réseaux de capteurs », Université de Toulouse, Thèse.

[3] Lyes KHELLAD, Nadjib BADACHE (Février 2004), « Les réseaux de capteurs : état de l'art », N° LSI-TR0304, Université USTHB.

[4] Vijay Raghunathan, Curt Schurgers, Sung Park, and Mani B. Srivastava. Energy-aware wireless microsensor networks. IEEE Signal Processing Magazine, 19(2): 40_50, March 2002.

[5] Medetonhan Shambhalla Eugène William PAMBA CAPO-CHICHI (Mars 2010), « Conception d'une architecture hiérarchique de réseau de capteurs pour le stockage et la compression de données », L'UFR des Science et Technique de l'université de FRANCHE-COMTÉ, Thèse.

[6]Arampatzis, Th., J. Lygeros et S. Manesis (2005). A survey of applications of wireless sensors and wireless sensor networks. In : *Proceedings of the 2005 IEEE International Symposium on Intelligent Control, Mediterrean Conference on Control and Automation*.

[7] Arora, A., P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora et M. Miyashita(2005). A line in the sand: A wireless sensor network for targer detection, classification, and tracking. *IEEE Computer Networks*

[8] Mainwaring, Alan, Joseph Polastre, Robert Szewczyk, David Culler et John Anderson (2002). Wireless sensor networks for habitat monitoring. In: *2002 ACM International Workshop on Wireless Sensor Networks and Applications. WSNA '02*. Atlanta GA.

[9] Naumowicz, Tomasz, Robin Freeman, Andreas Heil, Martin Calsyn, Eric Hellmich, Alexander Brandle, Tim Guilford et Jochen Schiller (2008). Autonomous monitoring of vulnerable habitats using a wireless sensor network. In: *Workshop on Real-World Wireless Sensor Networks, REALWSN'08*. Glasgow, Scotland.

[10] Biagioni, Edoardo S. et K. W. Bridges (2002). The application of remote sensor technology to assist the recovery of rare and endangered species. *International Journal of High Performance Computing Applications*.

[11] Schulz, Jens, Frank Reichenbach, Jan Blumenthal et Dirk Timmermann (2008). Low cost system for detecting leakages along artificial dikes with wireless sensor networks. In: *Workshop on Real-World Wireless Sensor Networks, REALWSN'08*. Glasgow, Scotland.

[12] Doolin, David M. et Nicholas Sitar (2005). Wireless sensors for wildfire monitoring. In: *Proceedings of SPIE Symposium on Smart Structures & Materials/ NDE 2005*. San Diego, California.

[13] Salles, Nicolas, Nicolas Krommenacker et Vincent Lecuire (2008). Performance study of IEEE 802.15.4 for industrial maintenance applications. In: *IEEE International Conference on Industrial Technology, ICIT2008*. Chengdu, China.

[14] Willig, A., M. Kubisch, C. Hoene et A. Wolisz (2002). Measurements of a wireless link in an industrial environment using an ieee 802.11-compliant physical layer. *IEEE Transaction on Industrial Electronics*.

[15] Gregory J. Pottie and W. J. Kaiser. Wireless integrated network sensors. Communications of the ACM, 43(5) :51_58, 2000.

[16] Wei Ye, John Heidemann, and Deborah Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. IEEE/ACM Transactions on Networking, 12(3) :493_506, 2004.

[17] J. A. Paradiso and T. Starner. Energy scavenging for mobile and wireless electronics. Pervasive Computing, IEEE, 2005.

[18] François INGELREST, Olivier ROY, Martin VETTERLI, « Traitement du Signal pour Réseaux de Capteurs : Échantillonnage et Communications », École Polytechnique Fédérale de Lausanne (EPFL), Suisse

[19] Ted Herman, Sébastien Tixeuil, « *Un algorithme TDMA réparti pour les réseaux de capteurs », University of Iowa, Department of Computer Science LRI - CNRS UMR 8623, INRIA Projet Grand Large, Universit´e Paris-Sud XI*

[20] Gérard Chalhoub, *« Routage et MAC dans les réseaux de capteurs sans fil »*, UNIVERSITE BLAISE PASCALE COLE DOCTORALE SCIENCES POUR L'INGENIEUR DE CLERMONT-FERRAND Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes, 8 Novembre 2010

[21] Gérard Chalhoub, *« MaCARI : Une méthode d'accès déterministe et économe en énergie pour les réseaux de capteurs sans fil»*, UNIVERSITE BLAISE PASCALE COLE DOCTORALE SCIENCES POUR L'INGENIEUR DE CLERMONT-FERRAND Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes, Thèse 2011

[22] Jérémie Decouchant et Olivier Alphand, *« Techniques de compression distribuées dans les réseaux de capteurs sans fils »*, Ensimag - Grenoble INP, Travaux d'étude et de Recherche, 11 mai 2011.

[23] Claude Castelluccia et Aurélien Francillon, « Protéger les réseaux de capteurs sans fil », INRIA Grenoble-Rhône-Alpes.

[24] Vijay Raghunathan, Saurabh Ganeriwal, and Mani Srivastava. Emerging techniques for long lived wireless sensor networks. IEEE Communications Magazine, 44(4) :108_114, April 2006.

[25] N. Kimura and S. Latifi. A survey on data compression in wireless sensor networks. In International Conference on Information Technology : Coding and Computing, volume 2, pp.8_13, 2005.

[26] Thèse - VAN DEN BOSSCHE Adrien - Proposition d'une nouvelle méthode d'accès déterministe pour un réseau personnel sans _l à fortes contraintes temporelles, Spécialité : Génie Informatique, Automatique et Traitement du Signal, le 06 Juillet 2007.

[27] (2009) Bluetooth SIG - The Official Wibree website. http ://www.wibree.com/

[28] (2009) The ZigBee Alliance website. [Online]. Available : http ://www.zigbee.org/

[29] H. Labiod, editor. Réseaux mobiles ad hoc et réseaux de capteurs sans fil. Hermes Science Publications - Traité IC2, série Réseaux et télécommunications, ISBN 2-7462-1292-7, Avril 2006.

[30] The Sensor Museum. http://www.btnode.ethz.ch/Projects/SensorNetworkMuseum/

[31] (2008) The Crossbow website. [Online]. Available : http://www.xbow.com

[32] 1995-2009 The Texas Instruments website. http://focus.ti.com/

[33] Trevor Armstrong. Wake-up based power management in multi-hop wireless networks, 2005. Term Survey Paper, University of Toronto, available at http://www.eecg.toronto.edu/ trevor/Wakeup/index.html.

[34] P. Muhlethaler, 802.11 et les réseaux sans fil. Eyrolles, 2002

[35] G. P. Halkes, T. Van Dam, and K. G. Langendoen, "Comparing energy-saving MAC protocols for wireless sensor networks," Mobile Networks and Applications, no. 10, pp. 783–791, 2005.

[36] N. Kimura and S. Latifi. A survey on data compression in wireless sensor networks. In International Conference on Information Technology : Coding and Computing, volume 2, pp.8_13, 2005

[37] F. Marcelloni and M. Vecchio. A simple algorithm for data compression in wireless sensor networks. IEEE Communications Letters, Vol. 12, No. 6, pp. 411_413, June 2008.

[38] Jérémie Decouchant, Olivier Alphand. « Techniques de compression distribuées dans les réseaux de capteurs sans fils », Travaux d'Etude et de Recherche Ensimag - Grenoble INP, 11 mai 2011

[39] MM. S.Maadi, Y. Peneveyre, et C. Lambercy. *« Compression de données avec pertes »*, Services de téléinformatique.

[40] Adel CHOUHA, *« Traitement et Transfert d'images Par Réseau de Capteurs sans Fil »*, Université Hadj Lakhder – Batna, Thése 2011

[41] MM. S.Maadi, Y. Peneveyre, et C. Lambercy. *« Compression de données sans pertes »*, Services de téléinformatique.

[42] D. Huffman. A method for the construction of minimum-redundancy codes. Proceedings of the IRE, Vol. 40, No. 9, pp. 1098_1101, Sept. 1952.

[43] PEREIRA Vincent - LEPRETTE Franck - HACAULT Vincent, *« Compression de données »*, Décembre 2004

[44] T. A. Welch. *A technique for high-performance data compression*. Computer, 17(6) :8_19, 1984.

[45] C. M. Sadler and M. Martonosi. Data compression algorithms for energy-constrained devices in delay tolerant networks. In Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys 2006), pp. 265_278, Boulder, Colorado, USA, 2006.

[46] The Weather Underground website. [Online]. Available:https://english.underground.com.

[47] F. Busque, S. Virally, J.S. Corbeil, D.A. Guzman, F. Beaudet, M. LEDUC, V. Couderc Vincent, P. Leproux, G. Huss, N. Gobout, S. Lacroix - Utilisation de la variance d'Allan pour caractériser la stabilité en puissance de sources pulsées, Journées Nationales d'Optique Guidée, Poster, Lannion, France, octobre 2008.

[48] Emmanuel Conchon « *Définition et mise en oeuvre d'une solution d'émulation de réseaux sans fil »*. Institut National Polytechnique De Toulouse. Octobre 2006

[49] Gianni A. Di Caro « *Analysis of simulation environments for mobile ad hoc networks»*. Technical Report No. IDSIA-24-03. Dalle Molle Institute for Artificial Intelligence, Galleria 2, 6928 Manno, Switzerland. 2003

[50] Philip Levis and David Gay, « *TinyOS Programming»* , July 16, 2009

[51] David Gay, David Culler, Philip Levis, *« nesC Language Reference Manual »*, 2002, [online] Disponible http ://nescc.sourceforge.net/papers/nesc-ref.pdf

[52] (2004) The TinyOS Community Forum. http ://www.tinyos.net/

[53] Philip Levis and Nelson Lee, *«TOSSIM: Un simulateur pour les réseaux TinyOS »*,TinyOS 1.1.0, Septembre 2003

[54] Philip Levis, Nelson Lee, Matt Welsh, and David Culler, *« TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications »*.

[55] Samuel BENNY KUMMARY, « Visualization of Sensor network application in simulated environments », B.E., Andhra University, India, 2005, thése 2009

[56] BENCHAIRA Djawhara, BENCHIKH Ahlem, *« Sécurité de la dissémination de données dans un réseau de capteur sans fil »*, Ecole Nationale Supérieure d'Informatique (ESI), Oued-Smar Alger, 2009

[57] MAIZ Zina, *« Les impacts de la technique de mise en veille sur l'énergie dans les réseaux de capteurs sans fil »*, Master2, Université Mouloud MAMMERI Tizi-OUZOU, 2011

Annexes

Annexe A : le système d’exploitation TinyOS

Annexe B : le langage de programmation NesC

Annexe C : l’interface graphique TinyViz

Annexe D : le simulateur PowerTOSSIM

Annexe A : Le système d'exploitation TinyOS

A.1. Présentation générale de TinyOS

TinyOS est un système d'exploitation open source conçu pour les réseaux de capteurs par l'université américaine de BERKELEY. Le caractère open source permet à ce système d'être régulièrement enrichi par une multitude d'utilisateurs. Sa conception a été entièrement réalisée en NesC, langage orienté composant syntaxiquement proche du C. Il respecte une architecture basée sur une association de composants, réduisant ainsi la taille du code nécessaire à sa mise en place. Cela s'inscrit dans le respect des contraintes de mémoires qu'observent les capteurs pourvus de ressources très limitées dues à leur miniaturisation.

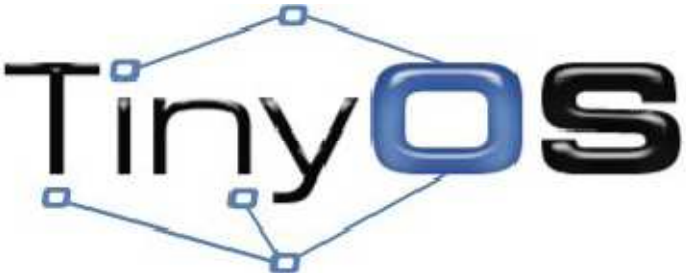


Fig. A-1 : Cigle du système d'exploitation TinyOS.

Pour autant, la bibliothèque de composants de TinyOS est particulièrement complète puisqu'on y retrouve des protocoles réseaux, des pilotes de capteurs et des outils d'acquisition de données. Un programme s'exécutant sur TinyOS est constitué d'une sélection de composants systèmes et de composants développés spécifiquement pour l'application à laquelle il sera destiné (mesure de température, taux d'humidité…).

TinyOS s'appuie sur un fonctionnement évènementiel, c'est-à-dire qu'il ne devient actif qu'à l'apparition de certains évènements. Le reste du temps, le capteur se trouve en état de veille, vu les faibles ressources énergétiques des capteurs, garantissant ainsi une durée de vie maximale. Ce type de fonctionnement permet une meilleure adaptation à la nature aléatoire de la communication sans fil entre capteurs.

A.2. Caractéristiques de TinyOS

TinyOS a été créé pour répondre aux caractéristiques et aux nécessités des RCSF telles que :

- **Taille réduite** : TinyOS a une empreinte mémoire très faible puisqu'il ne prend que 4 Ko de mémoire libre et 300 à 400 octets dans le cadre d'une distribution minimale.

- **Applications orientées composants**: Un programme s'exécutant sur TinyOS est constitué d'une sélection de composants qui peut être utilisée telle quelle ou bien adaptée à une application précise (mesure de température, du taux d'humidité, etc.). A cette fin, TinyOS fournit une réserve de composants systèmes utilisables au besoin. Parmi les plus fréquents, on cite ceux concernant les entrée/sorties, les timers, etc.

TinyOS utilise un Langage de Description d'Architecture ou ADL6 afin de définir quels sont les composants impliqués dans la création de l'application ainsi que la manière dont ils sont reliés. Cette liaison entre composants repose sur la notion d'interface.

- **Programmation orienté évènement**: Le plus gros avantage de TinyOS est qu'il est basé su un fonctionnement événementiel, c'est à dire qu'il ne devient actif qu'à l'apparition de certains évènements. Le reste du temps, le capteur se trouve en état de veille afin de garantissent une durée de vie maximale aux faibles ressources énergétiques du capteur. Ce fonctionnement évènementiel (event-driven) s'oppose au fonctionnement dit temporel (time-driven) où les actions du système sont gérées par une horloge donnée.

- **Non Préemptif:** Le caractère préemptif d'un système d'exploitation précise si celui-ci permet l'interruption d'une tâche en cours. TinyOS ne gère pas ce mécanisme de préemption entre les tâches. Autrement dit, une tâche ne peut pas interrompre une autre tâche. Ce mode de fonctionnement permet de bannir les opérations pouvant bloquer le système et donne la priorité aux interruptions matérielles (i.e. les événements peuvent interrompre les tâches). TinyOS est donc basé sur une structure à deux niveaux de planification :
  - Les évènements : ils sont utilisés pour réaliser des processus urgents et courts.
  - Les tâches : les tâches sont pensées pour réaliser une plus grande quantité de traitements et elles ne sont pas critiques dans le temps. Les tâches sont exécutées complètement, mais l'initialisation et la terminaison d'une tâche sont des fonctions séparées. Les tâches ne peuvent pas prendre de paramètre en entrée.

- **Pas de temps réel** : Lorsqu'un système est dit « temps réel » celui ci gère des niveaux de priorité dans ses tâches permettant de respecter des échéances données par son environnement. Dans le cas d'un système strict, aucune échéance ne tolère de dépassement contrairement à un système temps réel mou. TinyOS se situe au delà de ce second type car il n'est pas prévu pour avoir un fonctionnement temps réel.

A.3. Equipements supportés par TinyOS

TinyOS peut être implémenté sur un PC capteur (ATMega8, AVRMote, Mica, Rene2, MSP430, Telos). Au delà de cette liste, il est possible d'implémenter tout type de plateforme embarquée physique en redéveloppant les bibliothèques nécessaires à la prise en compte des entrées sorties nécessaires. Citant comme exemple le résultat d'une thèse mettant en œuvre TinyOS sur un dispositif Freescale MC13192-EVB (semi-conducteur utilisé pour évaluer des plateformes) sur un réseau ZigBee**.**

A.4. Allocation de la mémoire

Il est très important d'aborder la façon avec laquelle un système d'exploitation gère la mémoire, d'autant plus lorsque ce système travaille dans un environnement aussi restreint. TinyOS occupe un espace mémoire faible répartie en :

- Pile : sert de mémoire temporaire au fonctionnement du système notamment pour l'empilement et le dépilement des variables locales.
- Variables globales : réservent un espace mémoire pour le stockage de valeurs pouvant être accessible depuis des applications différentes.

- Mémoire libre : pour le reste du stockage temporaire.

TinyOS possède une mémoire fixe. En effet, il interdit les allocations dynamiques ainsi que celles se produisant à l'exécution. De plus, les pointeurs de fonctions n'existent pas. Pour cela, TinyOS s'appuie sur le graphe de composants précédemment décrit afin de déterminer la taille de chaque composant et ainsi établir statiquement leurs liaisons à la compilation. Par ailleurs, il n'existe pas de mécanisme de protection de la mémoire sous TinyOS, ce qui rend le système particulièrement vulnérable aux corruptions de la mémoire.

A.5. Allocation de ressources

Le choix d'un ordonnanceur détermine le fonctionnement global du système et le dotera de propriétés précises telles que la capacité à fonctionner en évènementiel. L'ordonnanceur TinyOS se compose de :

– 2 niveaux de priorités (bas pour les tâches, haut pour les évènements).
– 1 file d'attente FIFO (disposant une capacité de 7).

A l'appel d'une tâche, celle-ci va prendre place dans la FIFO en fonction de sa priorité (plus elle est grande, plus le placement est proche de la sortie). Dans le cas où la file d'attente est pleine, la tâche dont la priorité est la plus faible est enlevée de la file FIFO. Lorsque la file est vide, le système met en veille le dispositif jusqu'au lancement de la prochaine interruption.

A.6. Guide d'installation :

Deux principales versions de TinyOS sont disponibles : la version stable (1.1.0) et la version en développement (2.0.2) qui nécessite l'installation de l'ancienne version pour fonctionner. TinyOS peut être installé sur Windows (2000 et XP), GNU/Linux, Mac OS ou sur un capteur. Nous avons procédé à l'installation de la première version de TinyOS sur Windows Vista et ubuntu 10.10.

Annexe B : Le langage de programmation NesC

B.1. Présentation générale de NesC

NesC est une extension du langage de programmation C. Il est conçu pour incarner les concepts structurant et le modèle d'exécution de TinyOS. Les composants sont les éléments de base pour former une application NesC. Chaque composant correspond à un élément matériel (LED, timer, ADC …) et peut être réutilisé dans différentes applications.

Les composants NesC fournissent ou utilisent des interfaces bidirectionnelles qui définissent d'une manière abstraite les interactions entre deux composants. L'utilisation des mots clés **use** et **provide** au début d'un composant permet de savoir respectivement si celui-ci fait appel à une fonction de l'interface ou redéfini son code. Il est à noter que tous les composants NesC doivent posséder l'interface StdControl car celle-ci est utilisée pour initialiser, démarrer et arrêter les composants.

Les composants NesC présentent des similarités avec des objets. Les états sont encapsulés et on peut y accéder par des interfaces. En NesC, l'ensemble des composants et leurs interactions sont fixés à la compilation pour plus d'efficacité. Ce type de compilation permet d'optimiser l'application pour une exécution plus performante. En langage objet, cette phase est réalisée lors de l'exécution ce qui rend celle ci plus lente.

B.2. Implémentation d'une application NesC

Pour implémenter une application NesC, il faut avoir connaissance sur la structure et le fonctionnement des composants et des interfaces qui la constituent. Cette partie permet de bien expliquer ces notions. Il est néanmoins recommandé de faire recours aux leçons au niveau du tutorial TinyOS qui englobe tous les besoins de programmation NesC en accédant à **/opt/tinyos-1.x/doc/tutorial**

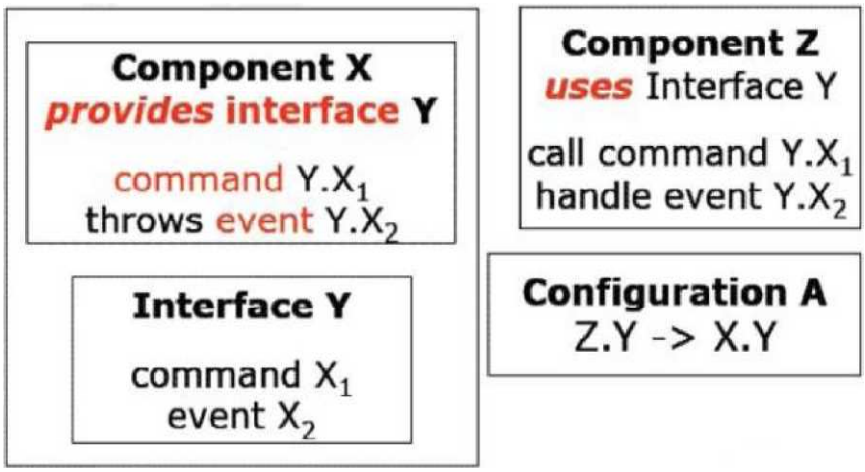


Fig. B-1 : Architecture générale d'une application NesC

B.2.1. Les interfaces

Une interface déclare deux types de fonctions: commandes et événements. Afin de distinguer ces fonctions, leurs en-têtes sont précédés des mots-clés respectifs ***event*** ou ***command***.

Les commandes font typiquement des appels du haut vers le bas (des composants applicatifs vers les composants plus proches du matériel). Tandis que les événements remontent les signaux du bas vers le haut.

Pour appeler une commande, il faut utiliser le mot-clé **call**. Par exemple:

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

Pour signaler un événement, il faut utiliser le mot-clé **signal**. Par exemple:

signal Send.sendDone(&msg1, SUCCESS);

Le modèle mémoire fixé par TinyOS n'autorise pas les pointeurs de fonctions. Afin de proposer un mécanisme alternatif, NesC utilise des interfaces paramétrées. Celles-ci permettent à l'utilisateur de créer un ensemble d'interfaces identiques et d'en sélectionner une seule à appeler grâce à un identifiant. Par exemple : interface **SendMsg [uint8_t id]**

B.2.2. Les composants

Il existe deux types de composants : les configurations et les modules.

B.2.2.1. Les configurations

Elles permettent de décrire les composants composites, i.e., des composants composés d'autres composants. Elles relient les interfaces utilisées par certains composants aux interfaces offertes par d'autres composants. Une configuration est donc constituée de modules et/ou d'interfaces ainsi que de la description des liaisons entre ces composants. Il existe trois possibilités de connexion:

- End-point1 = End-point2
- End-point1 -> End-point2
- End-point1 <- End-point2 (équivalent à : endpoint2 -> endpoint1)

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

Il est à noter que la configuration Main est obligatoirement présente dans la configuration décrivant l'ensemble de l'application car son rôle est de démarrer l'exécution de l'application.

B.2.2.2. Les modules

Ce sont les éléments de base de la programmation. Ils permettent de fournir les codes des applications NesC. Pa ailleurs, il est à noter que le modèle d'exécution proposé par NesC repose sur les tâches et les gestionnaires d'interruption. Donc, les modules permettent aussi d'implémenter ces tâches.

Une tâche est un ordonnancement FIFO utilisée pour réaliser un travail qui nécessite beaucoup de calculs. Elle peut être postée par une commande ou un événement. C'est un élément de contrôle indépendant défini par une fonction retournant **void** et sans arguments :

task void NomTask() { ... }

Les tâches sont lancées en les préfixant par post:

post NomTask();

B.3. Compilation d'une application NesC

Les fichiers de NesC portent l'extension *.nc*. Par ailleurs, le compilateur de NesC est appelé ncc. Pour effectuer la compilation, les fichiers sources doivent se situer dans le même répertoire contenant aussi un makefile de la forme :

COMPONENT= nom de l'application
include ../Makerules

Ce Makefile permet de compiler le composant en spécifiant en paramètre la plateforme sur laquelle doit fonctionner l'application. Par exemple, pour un capteur de type mica2, la commande permettant de compiler l'application sera : make mica2. Le compilateur ncc offre aussi la possibilité de pouvoir compiler l'application pour l'utiliser sur un simulateur de TinyOS. Dans ce cas, la commande sera : **make pc**. Cette commande génère un exécutable **main.exe** dans l'arborescence **/repertoire_courant/build/pc**.

B.4. Exemple illustratif d'une application NesC

On va donner l'exemple universel « Bonjour » ou « Hello » pour mieux illustrer la structure d'une application NesC.

Hello.nc

```
configuration Hello {
}

implementation {
components Main, HelloM,
SingleTimer, LedsC ;
Main.StdControl --->
HelloM.StdControl;
Main.StdControl – –> SingleTimer;
HelloM.Timer --->
SingleTimer.Timer ;
HelloM.Leds ---> LedsC ;

}
```

HelloM.nc (3)

```
module HelloM {
provides {
interface StdControl;
}
uses {
interface Timer; interface
Leds;
}
}
```

HelloM.nc (2)

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

HelloM.nc (3)

```
event event Timer.fired(){
call Leds.redOn();
call Leds.greenOn();
call Leds.yellowOn();
return SUCCESS;
}
} // implementation
```

Fig. B-2 : Exemple illustratif d'une application NesC.

Annexe C : L'interface graphique TinyViz

C.1. Présentation générale de TinyViz

TinyViz est fourni avec TinyOS. Il s'agit d'une interface graphique programmée en langage JAVA. Elle permet de représenter un RCSF émulé grâce au simulateur TOSSIM.

Pour plus d'informations sur l'utilisation de TinyViz, aller à:
http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson5.html

Une fois TinyViz est lancé, on peut visualiser une fenêtre comme celle illustrée dans la figureC.1. Dans la partie gauche de cette figure, on distingue les capteurs qui sont déplaçables dans l'espace. Quant à la partie droite, on distingue les commandes permettant d'intervenir sur la simulation:

- **On/Off**: met en marche ou éteint un capteur.
- **Delay**: permet de sélectionner la durée au bout de laquelle se déclenche le timer.
- **Play**: permet de lancer la simulation où de la mettre en pause
- **Bouton de grilles**: affiche un quadrillage sur la zone des capteurs afin de pouvoir les situer dans l'espace.
- **Clear**: efface tous les messages qui avaient été affichés lors de la simulation.
- **Stop**: arrête la simulation et ferme la fenêtre.

Pour lancer une application, il faut régler le **Delay** souhaité entre chaque application, choisir les plugins de visualisation que l'on souhaite, et, appuyer sur **Play**. La simulation démarre.

Chaque onglet contient un plugin qui permet de visualiser la simulation de façon plus ou moins détaillée. Par exemple, en activant le plugin **Debug Messages**, tous les messages de type **Debug** apparaîtront dans l'onglet correspondant. Le plugin **Radio Links** permet de visualiser graphiquement par des flèches, les échanges effectués entre les capteurs. Plus précisément, si un capteur envoie un broadcast, il sera repéré par un cercle. Par contre, s'il envoie un message direct (unicast) alors le lien de communication sera repéré par une flèche.

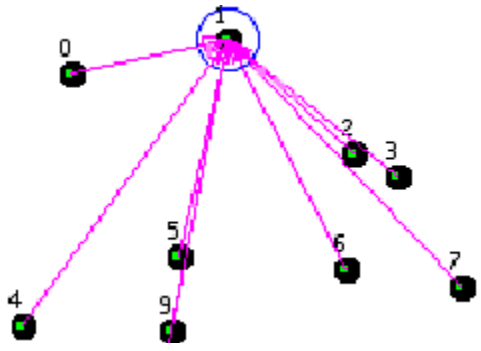


Fig. C-2 : Echange de messages entre les nœuds.

Annexe D: PowerTOSSIM

D.1. Présentation générale

Le simulateur TOSSIM n'a pas la capacité de vérifier le taux d'énergie dissipée pendant l'exécution des applications. Cependant, le besoin de vérifier la consommation énergétique dans un RCSF a un intérêt primordial. L'université de Harvard a conçu le simulateur PowerTOSSIM qui surmonte ce problème. Ce nouveau simulateur est intégré dans TOSSIM. Il permet de calculer le total d'énergie consommée par chaque composant constituant l'architecture de TOSSIM (LED, radio, CPU, etc.).

Pour simuler ces composants (voir figure VI-5), on fait appel au module PowerState. Ce dernier engendre des messages de transition d'états d'énergie (power state transition messages) pour chaque composant. Ces messages peuvent être combinés avec un modèle d'énergie pour générer en détail les consommations d'énergie. Pour se faire, un fichier programmé en langage python, intitulé « postprocess.py » est utilisé.

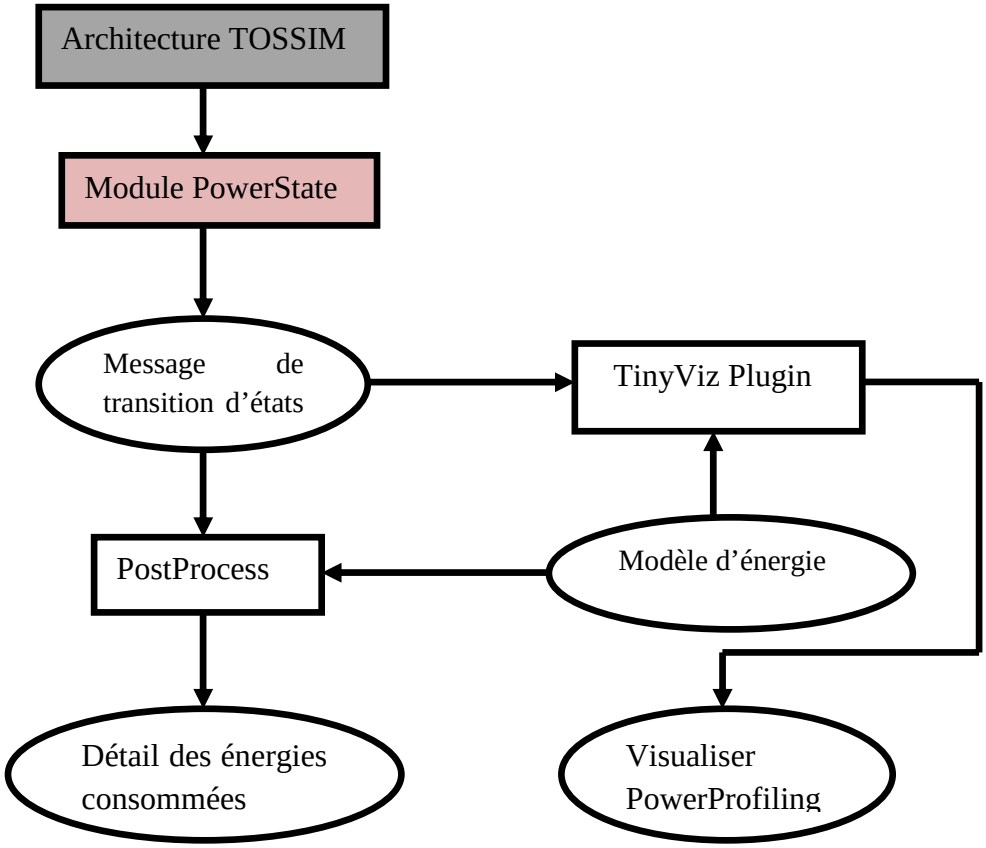


Fig. D-1: Architecture de PowerTOSSIM.

D.2. Lancer PowerTOSSIM

A- Pour récupérer l'énergie consommée par les nœuds du réseau, il faut passer par ces étapes:

1- Accéder à l'application à simuler et la compiler en tapant: **make pc**
2- Taper **export DBG=power**

3- Exécuter **main.exe** en choisissant le temps de simulation avec **–t** et le nombre de nœuds d réseau avec **–p**. Une trace de simulation est enregistré dans un fichier dont l'extension est **.trace** .Pour se faire, taper **: /build/pc/main.exe -t=60 -p 10 > NomApp.trace** (Le temps est égal à 60 secondes et le nombre de noeuds à 10)

4- Exécuter **postprocess.py** sur la trace de simulation en spécifiant les paramètres –sb et --em **/opt/tools/scripts/PowerTOSSIM/postprocess.py –sb=0 –em /opt/tools/scripts/Power TOSSIM/mica2_energy_model.txt NomApp.trace**

Le paramètre *–sb* spécifie si les noeuds sont attachés à un autre nœud (i.e. embarqué). En outre, le paramètre *–em* spécifie le modèle d'énergie.

Pour plus de détail sur l'utilisation d'autres paramètres de PowerTOSSIM, exécuter **postprocess.py –help**

5- Le résultat enregistre l'énergie totale utilisée par chaque composant sur chaque nœud. Il est sous la forme suivante :

Mote 0, cpu total: 719.503906
Mote 0, radio total: 1235.255862
Mote 0, adc total: 0.000000
Mote 0, leds total: 571.570576
Mote 0, sensor total: 0.000000
Mote 0, eeprom total: 0.000000
Mote 0, cpu_cycle total: 0.000000
Mote 0, Total energy: 2526.330344
.
.
Mote 9, cpu total: 635.394462
Mote 9, radio total: 1090.990102
Mote 9, adc total: 0.000000
Mote 9, leds total: 504.416514
Mote 9, sensor total: 0.000000
Mote 9, eeprom total: 0.000000
Mote 9, cpu_cycle total: 0.000000
Mote 9, Total energy: 2230.801078

6- Pour ne pas perdre ce résultat, il est commode de le sauvegarder dans un fichier texte. Pour se faire, Ajouter dans l'instruction de l'étape 4:
/opt/tools/scripts/PowerTOSSIM/postprocess.py –sb=0 –em /opt/tools/scripts/Power TOSSIM/mica2_energy_model.txt NomApp.trace > Result.txt

7- Pour avoir un résultat d'énergie plus détaillé, ajouter le paramètre **–detail** dans l'instruction de l'étape 4. Le résultat est enregistré automatiquement dans des fichiers textes dont le nombre est égal au nombre de nœuds simulés. Autrement dit, chaque fichier contient le détail de la consommation énergétique d'un seul nœud du réseau.

B- Pour récupérer l'état de l'horloge lors de la transmission et de la réception de paquets, il faut passer par ces étapes:

1- Accéder à l'application à simuler et la compiler en tapant: **make pc**
2- Taper **export DBG=clock**

Pour afficher des messages en parallèle avec l'horloge, taper **export DBG=clock, usr1**

3- Exécuter **main.exe** en tapant **: /build/pc/main.exe -t=60 -p 10 > NomApp.trace**

4- Accéder au fichier **NomApp.trace**
Il contient des lignes sous la forme suivante :

Moment d'envoi du paquet
Heure : Minute : Seconde

2: CLOCK: event handled for mote 2 at **0:0:36.47777400** (347634 ticks).
2: CLOCK: Setting clock interval to 218 @ 0:0:36.47777400
2: j'envoie le paquet de données à la destination 42 ///DBG usr1
.
.
42: j'ai reçu le paquet de données de la source 2 ///DBG usr1
.
.
.
.
.

Moment de réception du paquet. Délai de propagation du paquet=36, 60979650-36, 47777400=0, 13202250 secondes

42: CLOCK: event handled for mote 42 at **0:0:36.60979650** (902286 ticks).
42: CLOCK: Setting clock interval to 231 @ 0:0:36.60979650