Table des matières

Liste des figures

Liste des tableaux

Liste d'acronymes

AODV	Ad hoc On-demand Distance Vector routing protocol
API	Application Programming Interface
ATC	Aggregation Time Control
AVI	Absolute Validite Interval
BDCTR	Base de Données de Capteurs Temps Réel
CCA	Clear Channel Assessment
CoAP	Constrained Application Protocol
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CTP	Collection Tree Protocol
DODAG	Destination Oriented Directed Acyclic Graph
DSR	Dynamic Source Routing
EDF	Earliest Deadline First
ETT	Expected Transmission Time
ETX	Expected Transmission Count
FIFO	First In First Out
I-EDF	Implicit Earliest Deadline First
LLNs	Low power and Lossy Networks
LOAD	Lightweight On-demand Ad hoc Distance vector
LOADng	Lightweight On-demand Ad hoc Distance vector–next generation
OF	Objective Function
QoS	Quality-of-Service

REST	REpresentational State Transfer
RM	Rate Monotonic
RPL	Routing Protocol for Low power and Lossy Networks
RSSI	Received Signal Strength Indicator
SGBDR	Système de Gestion de Base de Données Relationnelle
SRT	semantic routing tree
TCP/IP	Transmission Control Protocol/Internet Protocol
TCT	Tolerance Coherency Temporal
TDMA	Time Division Multiple Access
TinyOS	Tiny Operating System
UDGM	Unit Disk Graph Medium Distance Loss
UDP	User Datagram Protocol
WSN	Wireless Sensor Networks

Chapitre 1

Introduction

Un Réseau de Capteurs Sans Fil (RCSF) est un type particulier de réseau Ad-hoc[1], caractérisé par un flux de données continu. Il est composé d'un ensemble de nœuds (capteurs) qui communiquent entre eux via des transmissions sans fil. Les nœuds possèdent la capacité de collecter et de transmettre des données depuis un environnement vers une station de base. Les RCSF ont conquis une diversité de champs d'applications (industriels, militaire, santé, transport, agriculture, etc.). Chaque application possède une méthode d'interrogation spécifique et des méthodes de communication qui dépendent du domaine de l'application.

Dans la littérature, plusieurs approches ont été proposées pour interroger les RCSF. Parmi ces approches, nous citons les approches basées sur les événements, les approches basées sur la durée, et les approches basées sur les requêtes, ainsi que les approches hybrides. Cependant, la divergence de toutes ces approches a suscité la mise en place de nouveaux systèmes d'interrogation déclarative des données de capteurs, appelés aussi Base de Données de Capteurs Sans Fil (BDCSF). Ils possèdent des caractéristiques spécifiques qui les différencient des bases de données classiques. Une BDCSF considère le RCSF comme une grande base de données distribuée dont les capteurs représentent des mini-entrepôts de données et les données des mesures effectives sur l'environnement physique. Les données ne sont pas toujours stockées dans cette base. Elles sont instantanées, transitoires, et fréquemment variables. La structure du flux de données de capteurs peut être assimilée à celle d'une table relationnelle comme dans les SGBD[2] traditionnels et les items à des n-uplets. Cette table virtuelle localisée au niveau de la station de base et les données sont

[1] Un réseau Ad-hoc est un réseau sans fil est capable de s'organiser sans avoir une infrastructure prédéfinie
[2] Système de Gestion de Bases de Données

interrogeables avec un langage relationnel proche de SQL (SQL-like).

L'idée initiale derrière la proposition des BDCSF est de réduire le nombre de communications dans le réseau et de s'intéresser plutôt à un traitement local des données (au niveau des capteurs eux-mêmes), et cela dans le but de réduire la consommation d'énergie dans le réseau et d'augmenter sa durée de vie [Bonnet *et al.*, 2001]. Les BDCSF actuelles telles que COUGAR [Yao and Gehrke, 2002] et TinyDB [Madden *et al.*, 2003a] utilisent de nouvelles techniques de traitement des données telles que l'agrégation, l'indexation et le stockage local. Elles ont intégré la couche de traitement des requêtes dans la couche réseau pour assurer un routage intelligent et un mécanisme efficace de sélection d'itinéraires. Elles proposent aussi de nouveaux types de requêtes et de nouvelles clauses SQL basées sur le temps et les événements, qui s'adaptent à la nature continue du flux de données de capteurs.

1. Contexte

Dans ce travail de thèse, nous nous sommes intéressés plutôt à l'ajout de contraintes temporelles dans les BDCSF. En effet, les applications utilisant les RCSF sont des applications temps réel. Elles nécessitent des délais de transmission de bout en bout respectant les contraintes temporelles définies par le système et l'environnement pour certains événements, afin d'alerter la population d'un risque ou d'un danger. De plus, les données prélevées par les capteurs doivent satisfaire la contrainte de cohérence temporelle définie par ces applications, afin de refléter fidèlement l'état réel de l'environnement et de satisfaire aux règles d'intégrité pour répondre aux besoins des utilisateurs.

Les RCSF doivent fournir des garanties de retard borné pour la transmission des données. Ils doivent aussi assurer la validité temporelle des données pour fournir des données fraîches aux applications, afin de refléter l'état courant de l'environnement.

Les SGBDTR[3] ont montré leur efficacité dans la gestion des données temps réel. Ils peuvent assurer la cohérence temporelle des données et maximiser le nombre des transactions qui respectent leurs échéances. Ils disposent d'algorithmes d'ordonnancement efficaces permettant de respecter les échéances des transactions temps réel [Duvallet *et al.*, 1999]. La mise en place d'une base de données de capteurs temps réel qui hérite de certains fonctions des SGBDTR, comme les algorithmes d'ordonnancement et les contrôleurs d'échéances, corresponde bien à notre contexte de thèse. Cependant, l'environnement des RCSF ainsi que l'architecture actuelle des BDCSF empêchent ces derniers de respecter les contraintes temporelles exigées par les applications temps réel.

[3]Système de Gestion de Bases de Données Temps Réel

2. Problématique

Les BD de capteurs actuelles proposent de nouvelles techniques de traitement des données telles que l'agrégation, l'indexation et le stockage, et ce en intégrant de nouveaux types de requêtes basées sur la durée de vie de la requête et sur les événements. De plus, ces types de requêtes s'adaptent à la nature continue des flux de données de capteurs. Les travaux réalisés pour étudier les contraintes temporelles dans les RCSF se sont concentrés sur la couche MAC, ou sur la couche réseau, en proposant de nouveaux protocoles MAC ou de routage, qui permettent d'améliorer les délais de transmission des données (de bout en bout) dans le réseau. L'approche *base de données* n'a pas été proposée pour résoudre le problème du respect de contraintes temporelles dans les RCSF. En effet, dans la littérature, les bases de données temps réel ont montré leur efficacité pour ordonnancer les tâches temps réel, de manière à respecter leurs échéances ; mais aussi pour assurer la cohérence temporelle de données. Vu les avantages des bases de données temps réel pour respecter les contraintes temporelles, et en se basant sur le principe que le RCSF forme une base de données, nous avons décidé, dans ce travail de thèse, d'étudier les contraintes temporelles dans les RCSF en utilisant l'approche *base de données*. Ainsi, les données prélevées par les capteurs doivent satisfaire la contrainte de cohérence temporelle pour refléter fidèlement l'état réel de l'environnement, et satisfaire aux règles d'intégrité pour répondre aux besoins des utilisateurs. De nombreuses applications utilisant les RCSF sont dites temps réel et exigent des contraintes temporelles sur certaines opérations telles que la validité temporelle des données, le temps de réponse à une demande "utilisateur" et le respect des échéances des transactions. Les RCSF doivent de plus délivrer les résultats exacts à l'utilisateur dans les délais imposés. Cependant, les RCSF déployés actuellement ne répondent pas à ces exigences. La cohérence temporelle d'une BDCSF peut être assurée en respectant les contraintes temporelles des transactions ainsi que la validité temporelle des données. Les problèmes liés aux contraintes temporelles dans les BDCSF peuvent être classés en deux catégories : des problèmes externes liés à l'environnement du RCSF, et des problèmes internes liés à l'architecture et au fonctionnement d'une BDCSF. Les problèmes externes peuvent être résumés dans les points suivants :

- **Contrainte d'énergie** : un capteur sans fil est alimenté par des batteries. Il peut donc épuiser son énergie très vite, au vu du nombre de communications et de traitements de données qu'il exécute. La défaillance d'un capteur peut engendrer des coupures de communications et des pertes ou un retard dans la livraison des données. Le prolongement de l'autonomie de la batterie du capteur est donc l'un des principaux défis dans la conception des RCSF. Ainsi, certaines approches proposent des méthodes telles que le contrôle en mode sommeil [Yutaka *et al.*, 2009], tandis que d'autres approches proposent une nouvelle classe de services sans fil qui fonctionne

à base de capteurs intelligents [Tarannum, 2010].

- **Quantité de données envoyées** : les RCSF génèrent une grande quantité données, qui évolue d'une manière exponentielle avec le nombre de capteurs. Ces données seront transmises à une station de base à travers le réseau. Les données collectées peuvent avoir un certain degré de corrélation ou de redondance. Lorsque le nombre de communications augmente, le temps de traitement des données augmente aussi, provoquant une surcharge temporelle. Des techniques d'agrégation de données et des techniques de traitement des requêtes ont été proposées pour réduire la quantité de données échangées dans le réseau.

- **Problème de synchronisation entre les capteurs** : le problème de synchronisation dans les systèmes distribués était discuté dans plusieurs travaux de recherche [Noh *et al.*, 2008]. Dans les RCSF, chaque capteur est indépendant des autres capteurs. Il possède sa propre horloge et ses notions de temps. Le processus de synchronisation d'horloge est nécessaire pour assurer un contrôle fiable de la validité temporelle des données, ainsi que sur les échéances des transactions. Le mécanisme de synchronisation entre les capteurs est basé sur un message de référence diffusé entre les nœuds, qui permet à chaque nœud de calculer le décalage d'horloge avec ses voisins, et d'ajuster son temps de synchronisation. Cependant, la défaillance d'un nœud ou le changement de sa position peut engendrer des problèmes dans les calculs de synchronisation entre les capteurs. On trouve plusieurs protocoles de synchronisation dans les RCSF tels que TPSN[4] [Ganeriwal *et al.*, 2003] et FTSP[5] [Maroti *et al.*, 2004] qui utilisent une synchronisation émetteur-à-récepteur ou RBS[6] [Elson *et al.*, 2002] qui utilise une synchronisation récepteur-à-récepteur.

- **Taille des RCSF :** l'augmentation du nombre de nœuds dans les RCSF peut atteindre des centaines ou des milliers de nœuds pour certaines applications. Le nombre de nœuds a un impact sur la qualité de service (QdS) du RCSF. En effet, le déploiement des nœuds d'un réseau de grande taille est difficile. Il rend l'auto-organisation des nœuds, ainsi que la tolérance aux défaillances très délicates, pouvant ainsi provoquer une redondance de données et une augmentation des fautes dans le réseau. Les protocoles de routages ainsi que les programmes dédiés aux RCSF doivent être en mesure de gérer ce grand nombre de nœuds. Certains chercheurs ont proposé de mettre en place des nœuds "puits" (sink) pour agréger les données et pour faciliter les communications et assurer la coordination entre les nœuds.

- **Topologie dynamique :** la topologie d'un RCSF peut changer au cours du temps en raison de l'autonomie énergétique limitée des nœuds, pouvant engendrer des

[4]Timing-sync Protocol for Sensor Networks
[5]Flooding Time Synchronization Protocol
[6]Reference Broadcast Synchronization

défaillances au niveau des capteurs ou encore au niveau de la connectivité. L'ajout de nouveaux nœuds pour augmenter la taille du réseau ou pour remplacer des nœuds défaillants imposerait une nouvelle réorganisation du réseau, provoquant ainsi un changement dans la topologie du réseau. Les interférences entre les nœuds, le bruit sur le canal de transmission et le problème d'atténuation peuvent engendrer la perte de données lors de la transmission à travers le RCSF, qui influe directement sur la validité temporelle des données et sur le temps de réponse du réseau. Iino *et al.* ont proposé une méthode d'optimisation heuristique décentralisée, qui permet d'améliorer le nombre de communications réussies entre les capteurs, et ce dans une topologie dynamique [Iino *et al.*, 2010].

- **Tolérance aux fautes :** la fiabilité des RCSF est liée à la capacité du réseau à résister aux erreurs qui peuvent survenir pour diverses raisons, tels que le mauvais fonctionnement du matériel, des problèmes logiciels, ou les éventuels risques environnementaux. Lorsqu'un RCSF n'est pas préparé à faire face à de telles situations, les conséquences qui en découlent peuvent être graves dans le contexte des applications critiques. Plusieurs techniques ont alors été proposées afin de détecter les erreurs dans un RCSF telles que l'auto-diagnostic [Harte *et al.*, 2005], la détection groupée [Bhaskar and Sitharama, 2004] et la détection hiérarchique [Paradis and Han, 2007].

Les problèmes internes liés à l'architecture et au fonctionnement peuvent être résumés dans les points suivants :

- **Retards de transmission et/ou de réception** : pendant le processus de collecte, les retards de transmission et/ou de réception des données peuvent conduire au non-respect de la validité temporelle des données. De plus, l'accès concurrentiel entre les requêtes *utilisateur* d'une part, et les requêtes de mise à jour, d'autre part, engendre une surcharge sur la station de base et des retards dans les réponses. Un autre défi est donc de pouvoir faire coexister les mises à jour des capteurs et les requêtes continues tout en respectant la cohérence temporelle des données et les échéances des transactions.

- **Faible puissance de calcul :** un capteur sans fil possède un microcontrôleur avec une faible puissance de calcul et une mémoire limitée, ce qui ne lui permet pas d'exécuter des algorithmes complexes. De ce fait, il est important de réduire le nombre d'opérations pour minimiser la consommation d'énergie du capteur. Par exemple un capteur Mica2 possède 128 KB de mémoire et ne peut stocker que 512 KB.

- **Faible capacité de stockage :** l'incapacité du système pour stocker le flux de données entier suggère l'utilisation de structures de résumés (méthodes d'approximation). Elle ne lui permet pas de stocker des programmes de grandes tailles, qui peuvent

assurer des calculs précis pour des réponses fiables. En plus, le gros volume de données circulant dans le réseau sans tenir compte des données similaires, affaiblit la capacité du système à donner des réponses exactes qui reflètent la réalité.

- **Méthode de transmission des données** : les RCSF utilisent des méthodes de transmission basées sur la diffusion des données (broadcast). Chaque nœud émet son message aux nœuds à portée radio (ses voisins). Les nœuds voisins doivent être à l'écoute du nœud émetteur au moment de la transmission de données. Ceci dépend du protocole MAC utilisé. Dans les RCSF, il existe deux catégories de protocoles MAC : les protocoles MAC synchronisés et les protocoles MAC avec préambule d'échantillonnage. La méthode de diffusion des données utilisant des protocoles MAC synchronisés est plus adaptée aux RCSF, car les nœuds de capteurs doivent se synchroniser entre eux pour se mettre dans un état actif ou dans un état endormi, en même temps. De cette manière, lorsqu'un nœud émet un paquet de données, les nœuds voisins sont activés et à l'écoute du nœud émetteur. Avec les protocoles MAC avec préambule d'échantillonnage, les nœuds s'endorment et se réveillent périodiquement pour écouter le canal de transmission. Tous les nœuds utilisent la même période d'échantillonnage qui sépare deux écoutes consécutives. Avant chaque transmission de données et pour assurer une transmission fiable des données, le nœud émetteur doit savoir le moment où ces voisins se réveillent pour leur envoyer les données, sinon il y aura des pertes de données [LE, 2008].

Dans cette thèse, nos travaux visent les contributions suivantes :

1. Dans la première contribution, nous avons réalisé une étude des bases de données traditionnelles et des bases de données de capteurs pour évaluer leurs performances respectives. Notre objectif de base était de déterminer quelle classe de base donnée fournit les meilleures performances et d'étudier l'impact de quelques contraintes comme le nombre de nœuds, la topologie du réseau, la position des nœuds et le protocole de la couche MAC sur les performances suivantes : (i) le temps de convergence de réseau, (ii) le temps de collecte des données et (iii) le temps de réponse de la base de données à une requête "utilisateur". Nous avons considéré deux scénarios. Le premier scénario est basé sur une approche de traitement des requêtes, qui inclut une base de données de capteurs abstraites. Le deuxième scénario est basé sur une approche de collecte périodique des données, qui inclut un stockage externe dans une base de données traditionnelle.

2. Dans la deuxième contribution, nous nous sommes intéressés à la validité temporelle des données dans les RCSF. Nous avons proposé un nouveau mécanisme de traitement des requêtes qui tient compte de la cohérence temporelle des données. Il permet à chaque nœud du réseau de contrôler la validité temporelle des données afin de ré-

duire la quantité de données invalides délivrée par les capteurs. Nous avons proposé de nouvelles clauses SQL à travers un langage de requêtes déclaratif similaire à SQL, qui permettent à l'utilisateur d'interroger les capteurs, en définissant l'intervalle de validité des données et d'attribuer une date limite pour l'exécution des requêtes. Nous avons également proposé un algorithme qui tient compte des nouvelles clauses SQL et qui assure le contrôle de la validité temporelle des données au niveau de chaque capteur, réduisant ainsi la quantité de données échangées dans le réseau et augmentant la quantité de données valides reçues par la station de base.

3. Dans la troisième contribution, nous avons proposé une nouvelle fonction objective dans le protocole de routage RPL[7] basée sur la latence entre les nœuds de capteurs, pour améliorer non seulement le temps de transmission des requêtes, mais aussi le temps de collecte des données.

4. Notre dernière contribution est la proposition d'une base de données de capteurs temps réel, qui peut garantir les contraintes temporelles dans les RSCF. Notre système est basé sur l'architecture des bases de données de capteurs, dont certains composants sont issus des SGBD temps réel. Il assure aussi une diffusion efficace des requêtes et une collection de données, grâce à la méthode de diffusion de requêtes et de collecte de réponses, basée sur la latence entre les nœuds de capteurs, présenté dans le chapitre 5. Notre système utilise un mécanisme de traitement des requêtes présenté dans le chapitre 5, ainsi que des contrôleurs d'échéances, qui s'exécutent au niveau des nœuds "puits" et sur la station de base pour contrôler les échéances des requêtes et la validité temporelle des données. Il implémente aussi un ordonnanceur temps réel, pour ordonnancer les transactions selon leurs priorités, pouvant ainsi supporter une exécution multi-requêtes.

3. Structure de la thèse

Nous avons organisé cette thèse comme suit : la partie 1 constitue l'état de l'art, dans lequel nous présentons dans le chapitre 2 les BDCSF, leur architecture et leur fonctionnement, leur langage des requêtes et différents systèmes de BDCSF existants. Dans le chapitre 3, nous discutons les contraintes temporelles dans les BDCSF et les différents protocoles temps réel proposés. La partie 2 de cette thèse qui constitue notre contribution, est composée de trois chapitres. Le chapitre 4 présente une étude des bases de données traditionnelles et des BDCSF, afin de pouvoir évaluer leurs performances en termes de respect des contraintes temporelles. Nous étudions également l'impact de quelques contraintes sur les performances du RCSF. Le chapitre 5 présente un nouveau modèle de

[7]RPL : Routing Protocol for Low power and lossy networks

traitement des requêtes que nous proposons pour le maintien de la cohérence des données temporelles dans les RCSF. Ce chapitre présente également une nouvelle version du protocole de routage RPL, qui inclut une nouvelle fonction objectif basée sur la latence entre les différents nœuds dans le réseau. Dans le chapitre 6, nous proposons une BDCSF temps réel basée sur le modèle de traitement de requêtes et sur le protocole de routage RPL modifié cités dans le chapitre 5. Elle inclut également quelques composants des systèmes de gestion de bases de données temps réel. Enfin, le dernier chapitre présente la conclusion et des perspectives.

Partie I

État de l'art

Chapitre 2

Base de données et réseau de capteurs sans fil

Sommaire

2.1 Introduction

Ces dernières années, l'utilisation des capteurs sans fil est en plein essor. Les réseaux de capteurs sans fil (RCSF) ont attiré l'intérêt des industriels. Ils ont conquis une diversité de champs d'applications (militaire, santé, transport, agriculture, etc). Ces capteurs sont souvent organisés en réseau ad hoc, qui n'a pas a priori de topologie définie. Les applications utilisant les capteurs et les réseaux de capteurs sont orientées données, dans le sens où les capteurs acquièrent puis envoient leurs données vers une station de base. Des recherches intensives ont commencé il y a un peu plus d'une dizaine d'années sur la structuration des données de capteurs en bases de données, afin de minimiser le nombre de communications et réduire le niveau de consommation d'énergie des capteurs. Dans ce chapitre, nous étudions les bases de données de capteurs sans fil (BDCSF). Dans la section 1, nous présentons les RCSF, leurs domaines d'applications, ainsi que la gestion de données dans les RCSF telle que l'approche de collecte des données, les traitements de requêtes, le stockage et l'indexation des données. Dans la section 2, nous présentons l'approche base de données dans les RCSF, leurs propriétés, leurs architectures, leurs modèles de données, et leur langage de requête. Par la suite, nous présentons les différents systèmes de BDCSF proposées dans la littérature, et nous comparons ces systèmes.

2.2 Réseau de capteurs sans fil

2.2.1 Présentation

Un réseau de capteurs sans fil est un type particulier de réseau ad-hoc [Egea-Lopez *et al.*, 2005]. Il est composé d'un grand nombre de nœuds (capteurs), déployés dans une zone d'intérêt, pour y collecter des informations telles que la température, l'humidité, la lumière, la pression, etc. Les données collectées sont transmises via des communications sans fil à une station de collecte, appelée Station de Base (SB). Ces données serviront à observer et à étudier certains phénomènes physiques ou à surveiller et à contrôler une zone d'intérêt. Les RCSF ont l'avantage de pouvoir être déployés facilement dans plusieurs environnements avec un faible coût.

Un capteur sans fil est un dispositif électronique de petite taille, capable d'effectuer un certain nombre de tâches tel que la collecte d'informations, la communication avec d'autres capteurs dans le réseau, le traitement d'un signal, la transmission des mesures à la station de base. Il est constitué de nombreux éléments dont les suivants : (1) une unité de traitement, chargée de l'exécution de différents programmes, ainsi que de la gestion des données provenant des autres capteurs, (2) une unité de stockage constituée à la fois

d'une mémoire "flash" pour stocker temporairement les données, ainsi qu'une mémoire vive pour stocker non seulement les valeurs intermédiaires du capteur, mais aussi les données reçues des autres nœuds, (3) un dispositif de détection pour capter les paramètres de l'environnement telles que la température, la pression, la lumière, l'humidité, etc., et également un convertisseur analogique-numérique pour convertir ces mesures en données compréhensibles par l'unité de traitement, (4) une unité d'émission/réception pour envoyer et recevoir les données à travers le réseau, (5) une unité d'alimentation pour fournir l'énergie aux différents composants du capteur.

2.2.2 Domaines d'applications

Les RCSF sont utilisés dans plusieurs domaines d'applications comme celui de la médecine, de l'agriculture, des transports, du contrôle de processus industriels, ou encore de l'armée.

2.2.2.1 Contrôle de l'environnement

Les RCSF peuvent aider à étudier les phénomènes environnementaux complexes comme les séismes, les volcans et les tempêtes. Par exemple, NEAREST[1] est un projet de la commission européenne visant le développement d'un système de détection précoce des Tsunamis locaux dans le Golfe de Cadiz. Des capteurs sismiques et de pression ont été mis en place pour des observations sous marines [Pignagnoli *et al.*, 2011]. On peut citer aussi le réseau de détection sismique de l'observatoire de Grenoble (réseau Sismalp), lancé en 1987, qui représente le seul réseau sismologique couvrant l'ensemble des Alpes occidentales françaises. Son objectif est de surveiller la sismicité régionale pour mieux en comprendre la sismotectonique et pour estimer le risque sismique [Guyoton, 1991].

2.2.2.2 Domaine militaire

Les RCSF sont déployés pour surveiller et pour détecter les intrusions non autorisées dans une zone protégée. Le système de surveillance, basé sur des capteurs sans fil, doit fournir au centre de commandement des informations précises, et ce avec une confidentialité acceptables sur la position des objets se déplaçant dans une certaine zone géographique, pour les localiser. Il doit de plus fournir les informations dans un temps de latence acceptable afin d'agir au bon moment et de prendre les précautions nécessaires.

[1] Integrated observations from NEAR shore sourcES of Tsunamis: towards an early warning system

2.2.2.3 Domaine de l'agriculture

Les RCSF peuvent améliorer le domaine de l'agriculture, en offrant notamment un support de gestion précis des ressources d'eau, un suivi du développement des maladies, ou encore une prédiction du moment adéquat de la récolte [Bechkit *et al.*, 2011].

2.2.2.4 Applications liées à la santé

Les RCSF peuvent améliorer plusieurs services liés à la santé tels que la surveillance à domicile des personnes âgées et des malades dans les hôpitaux. Par exemple, le projet STAR (Système Télé-Assistant Réparti) [Haiying *et al.*, 2004] consiste à concevoir une plateforme dédiée à la surveillance de personnes souffrant d'arythmies cardiaques. Il permet d'acquérir et d'analyser les signaux ECG [2] en temps réel, et de les envoyer à travers une passerelle connectée à Internet depuis la personne vers un centre de traitement médical ou un hôpital.

2.2.2.5 Systèmes de transports intelligents

Les RCSF sont également utilisés par les systèmes de transport intelligents (STI). Ils sont exploités dans des applications comme la gestion de flottes de véhicules et la surveillance du trafic. En effet, des capteurs vidéo, des capteurs à boucle, des capteurs acoustiques, des capteurs de pression et des radars sont posés temporairement sur certains axes routiers, pour fournir des informations sur le trafic (la vitesse, la densité, le taux d'occupation, les bouchons et les accidents) [Breheret *et al.*, 2000]. L'ensemble des mesures collectées sont utilisées par certaines applications de sécurité comme la gestion des congestions pour prévenir des accidents.

2.2.3 Gestion des données dans les RCSF

La gestion des données dans les RCSF inclut plusieurs processus tels que la collecte, le traitement et le stockage des données. Ces processus sont souvent accompagnés d'autres fonctions comme l'agrégation ou la compression de données, etc., selon la nature et la quantité de données à gérer, afin d'améliorer la qualité de service dans le réseau.

[2] Électrocardiographie : une représentation graphique de l'activité électrique du cœur.

2.2.3.1 Collecte des données

La collecte de données est une fonction principale dans les RCSF. Elle consiste à transmettre les données prélevées par des capteurs vers la station de base. Les nœuds forment un arbre de recouvrement et utilisent un protocole de routage pour envoyer leurs données vers la racine représentant la station de base. Les données collectées au niveau de la station de base peuvent ainsi être traitées immédiatement pour répondre aux besoins de l'utilisateur. Elles peuvent également être stockées dans une base de données pour être analysées par la suite. On peut distinguer plusieurs méthodes de collecte de données dans les RCSF :

- **Une collecte périodique** : les capteurs transmettent périodiquement leurs mesures à la station de base selon une période fixée par l'utilisateur ou l'application, afin d'observer et de contrôler des phénomènes dans des zones d'intérêt.
- **Une collecte événementielle** : la collecte des données est déclenchée suite à un événement tel qu'un dépassement du seuil de température fixé par le superviseur ou la détection d'intrusions dans une zone à surveiller. Les données sont transmises en continu vers la station de base, de manière à ce que l'utilisateur intervienne le plus vite possible.
- **Une collecte à la demande** : les capteurs répondent aux requêtes de l'utilisateur par les données demandées. L'utilisateur saisit sa demande sous forme d'une requête, qui sera transmise à la station de base, puis diffusée à travers le RCSF. Chaque capteur reçoit la requête, génère par la suite une réponse pour l'envoyer à la station de base, qui la transmet à son tour à l'utilisateur.
- **Une collecte hybride** : elle combine plusieurs méthodes de collecte des données. Par exemple, on peut combiner une collecte périodique avec une collecte événementielle. Les capteurs envoient périodiquement des informations sur une zone surveillée. Lorsque les capteurs détectent un événement, ils suspendent la transmission périodique, pour envoyer en continu les données détectées à la station de base, afin d'alerter les superviseurs.

On peut aussi classer la collecte des données dans les RCSF selon la quantité et de la nature des données collectées :

- Une collecte dans laquelle les données sont transmises individuellement vers la station de base lorsque les mesures sont importantes.
- Une collecte basé sur l'agrégation de données. Ce processus permet de fusionner plusieurs données provenant de plusieurs capteurs au niveau des nœuds "puits", afin d'éliminer les transmissions redondantes et de réduire la quantité de données transmises à travers le réseau [Ozdemir and Xiao, 2009]. L'agrégation de données

permet de minimiser la latence de communication entre les nœuds et la station de base, et de réduire la consommation d'énergie dans le réseau.

2.2.3.2 Traitement des requêtes dans les RCSF

Le processus de traitements des requêtes est basé sur les requêtes de l'utilisateur. Il représente un processus de collecte des données à la demande. L'utilisateur envoie ses requêtes au RCSF pour interroger les données des capteurs, ou pour interroger les données historiques qui sont stockées dans des bases de données centralisées. L'utilisateur saisit une requête dans un langage similaire au langage SQL. La requête est transmise à la station de base, puis elle est diffusée à travers le réseau. Il existe deux modes de diffusion de requêtes : le premier mode est une diffusion globale pour tous les nœuds du réseau et le deuxième mode est une diffusion sélective pour les nœuds qui contribuent au résultat de la requête. Chaque nœud reçoit la requête, et génère une réponse. Mais parfois le traitement d'une requête est réparti sur plusieurs nœuds, afin de générer le résultat final. Les réponses sont envoyées par la suite à la station de base, puis à l'utilisateur. Cependant, la transmission des résultats et la communication entre les nœuds nécessitent beaucoup d'énergie. De ce fait, un traitement local des données (au niveau de nœuds) est nécessaire pour réduire la consommation d'énergie dans le réseau, tout en utilisant des fonctions telles que l'agrégation et la compression des données [Hoeller *et al.*, 2010]. Les performances du processus de traitements par requêtes sont principalement liées au protocole de routage utilisé, qui lui, est responsable de la diffusion des requêtes et de la collecte des réponses [Krishnakumar, 2010]. Toutefois, un protocole de routage intelligent au niveau de la couche réseau pourrait offrir un mécanisme efficace de sélection d'itinéraires et réduire le nombre de transmissions de données. L'intégration de la couche de traitement des requêtes à la couche réseau conduit ainsi à une gestion intelligente des ressources afin de fournir une QdS qui satisfait les besoins de l'utilisateur.

2.2.3.3 Approche "base de données de capteurs"

L'approche base de données de capteurs considère le RCSF comme une grande bases de données distribuée et les capteurs comme des mini-entrepôts de données. La structure du flux de données de capteurs peut être assimilée à celle d'une table relationnelle comme dans les SGBD traditionnels et les items à des n-uplets arrivant en ligne. Cette table sera virtuelle au niveau de la station de base ou au niveau du nœud, et les données sont interrogeables avec un langage relationnel proche de SQL (SQL-like).

L'approche "base de données de capteurs" a été proposée, d'une part pour faciliter l'acquisition de données de RCSF, et d'autre part pour réduire le nombre de communications entre les

capteurs, afin de minimiser la consommation d'énergie et d'augmenter la durée de vie du réseau. Dans cette approche, le traitement des requêtes ainsi que le stockage de données sont locaux. L'utilisateur peut utiliser des requêtes sélectives sur un ensemble de nœuds pour sélectionner des données précises, mais aussi des requêtes globales pour interroger l'ensemble du réseau. L'approche "base de données" inclut plusieurs fonctions de traitement de données telles que l'agrégation et le filtrage de données, afin de réduire la consommation d'énergie dans le réseau. Plus de détails sur les bases de données de capteurs sont donnés dans la section 2.

2.2.3.4 Stockage des données

Les RCSF utilisent deux approches de stockage des données : (i) une approche d'entreposage et (ii) une approche distribuée. L'approche distribuée inclut un stockage sur tous les nœuds du réseau, et un stockage centrée-donnée (data-centric) concernant une partie des nœuds.

- **Approche d'entreposage** : dans cette approche, le stockage de données peut être effectué sur un serveur central ou dans des bases de données externes. Les données collectées depuis le RCSF sont envoyées périodiquement vers la station de base, où elles sont traitées. Par la suite, elles sont stockées dans une base de données distante que l'utilisateur peut interroger à travers des requêtes SQL pour récupérer des données. Il est à noter que le traitement des requêtes et l'accès au réseau de capteurs sont des processus bien séparés [Bonnet *et al.*, 2001]. Cette approche d'entreposage est bien adaptée pour répondre à des requêtes prédéfinies sur des données historiques. Par contre, elle est incapable de gérer un grand nombre de données pouvant créer un goulot d'étranglement sur le serveur central. Les nœuds situés près de la station de base restent à l'écoute des autres nœuds pour transmettre leurs réponses à la station de base. Ils ont ainsi une charge du travail supérieur à d'autres nœuds situés loin de la station de base. Ils peuvent épuiser leur énergie rapidement, ce qui peut engendrer des coupures de communications entre la station de base et le reste du réseau. Dans cette approche, le traitement des données est centralisé au niveau de la station de base, ce qui augmente le nombre de communications dans le réseau. Mais la transmission sans fil des données nécessite plus d'énergie que les traitements. Ainsi, selon une étude faite par [Jason *et al.*, 2000], la transmission d'un bit dans le réseau de capteurs consomme l'équivalent de l'exécution de 800 à 1000 instructions. La conséquence est que, la durée de vie du réseau diminue rapidement.

- **Approche distribuée** : dans une approche distribuée, le traitement ainsi que le stockage des données se sont effectués dans les capteurs eux-mêmes. Le stockage des données peut aussi être réalisé dans des bases de données distantes. Cette approche est basée sur une distribution globale des méta-données bien définie avec une

topologie du réseau bien maintenue [Sarkar, 2012], et dont les données sont stockées dans un serveur de bases de données et dans les capteurs eux-mêmes. En effet, les capteurs n'envoient pas leurs données périodiquement à la station de base, sauf s'ils reçoivent une requête avec les données demandées par l'utilisateur. Généralement, l'utilisateur ne connaît pas l'emplacement des données pertinentes. On distingue deux classes de stockage des données : un stockage général qui concerne tous les nœuds du réseau et un stockage "data-centric". La requête de l'utilisateur sera alors diffusée sur l'ensemble des capteurs du réseau. Un plan d'exécution est réparti sur les différents nœuds pour tenir compte de la charge de travail des requêtes, ainsi que pour déterminer les données qui doivent être extraites du réseau. Les requêtes de longue durée engendrent un long temps de réponse. De ce fait, seules les données pertinentes sont extraites et certaines données agrégées, permettant ainsi de minimiser le nombre de communications et de mieux exploiter les capacités de calcul des capteurs. Cette approche permet ainsi de réduire la consommation d'énergie dans le réseau puisque les nœuds ne sont activés que lorsqu'il y a une demande de données. Par contre, cette approche ne peut pas assurer l'équilibrage des charges entre les nœuds.

- **Approche distribuée centrée-donnée (data-centric)** : cette approche consiste à stocker les données collectées d'une région ou d'un type approprié de données dans des nœuds performants. Le réseau utilise les tables de hachage géographique pour bien répartir la charge sur tous les nœuds. Il est à noter qu'une table de hachage géographique possède deux fonctions : la première fonction permet de déposer des données sous la forme (clé, valeur). La clé représente la valeur de hachage de la donnée. Elle est utilisée pour distinguer les types de données, les emplacements de stockage et les événements. La deuxième fonction permet de récupérer la valeur de la donnée en se basant sur la clé. Toutes les données détectées et qui sont associées à un événement, seront ainsi envoyées au capteur le plus proche de cet événement. Les requêtes qui sont à la recherche de cet événement sont alors acheminées vers ce nœud. Cette approche assure une diffusion sélective qui dépend des données demandées par l'utilisateur, sans avoir interrogé l'ensemble des capteurs dans le réseau. Son avantage est de répartir la charge de travail sur tous les nœuds et de réduire la consommation d'énergie dans le réseau.

2.2.3.5 Indexation des données

Des méthodes d'indexation sont également proposées pour gérer les données issues d'un RCSF en temps réel. Par exemple, la méthode PoTree proposée dans [Noël and Servigne, 2004], la méthode PasTree proposée dans [Noël, 2006], et la méthode StH proposée dans

[Servigne and Noël, 2008]. Ces méthodes consistent à indexer les bases de données spatio-temporelles en mémoire vive [Noël and Servigne, 2005]. Le modèle PoTree consiste à utiliser des arbres pour structurer les données. Cette méthode permet de réduire la taille de la structure de données et de faciliter sa mise à jour. Le modèle PasTree est une amélioration du modèle PoTree. Il supporte le phénomène d'agilité des capteurs et est présenté sous la forme d'un arbre d'indexation permettant l'accès aux données par le biais des identifiants des capteurs. Elle contient une couche d'abstraction pour différencier les données collectées des données concernant le changement de position des capteurs. Enfin, le modèle STH (spatio temporal/heat) est basé sur le même principe que PasTree, mais il est dédié à la gestion de la saturation de la mémoire de la base de données.

2.3 Bases de données de capteurs sans fil

2.3.1 Présentation

Les bases de données de capteurs sans fil (BDCSF) constituent une abstraction des RCSF. Elles considèrent le RCSF comme une base de données relationnelle, composée de capteurs, dont chacun produit un ou plusieurs tuples de données, appelé source (ou table relationnelle). Les tables de la BDCSF sont virtuelles et représentent des vues (ensembles de mesures) générées par l'ensemble des capteurs. L'accès à ces tables virtuelles est assuré à travers des requêtes SQL, qui présentent des opérations d'interrogation ou de collecte des données organisées d'une manière qui permet une récupération efficace des données. La conception d'une BDCSF doit assurer une abstraction complète du réseau, car l'utilisateur n'est pas censé connaître la topologie réelle du réseau (cf. Figure 2.1).

Les bases de données de capteurs partagent la même terminologie que les bases de données classiques, excepté le fait que les données de la BDCSF sont générées d'une manière dynamique (cf. Tableau 2.1).

Terme	**Description**
Modèle de données	Un modèle qui décrit de façon abstraite la représentation sémantique des données
Un tuple	Un groupe de données composé de plusieurs valeurs d'attributs relatives à un objet
Une source	Le capteur qui génère un tuple
Une table	Une collection de tuples de même type
Un opérateur	Une fonction qui prend une ou plusieurs tables en entrée et génère une table en sortie
Une requête	Une composition d'opérateurs

Tableau 2.1: Terminologie des bases de données de capteurs

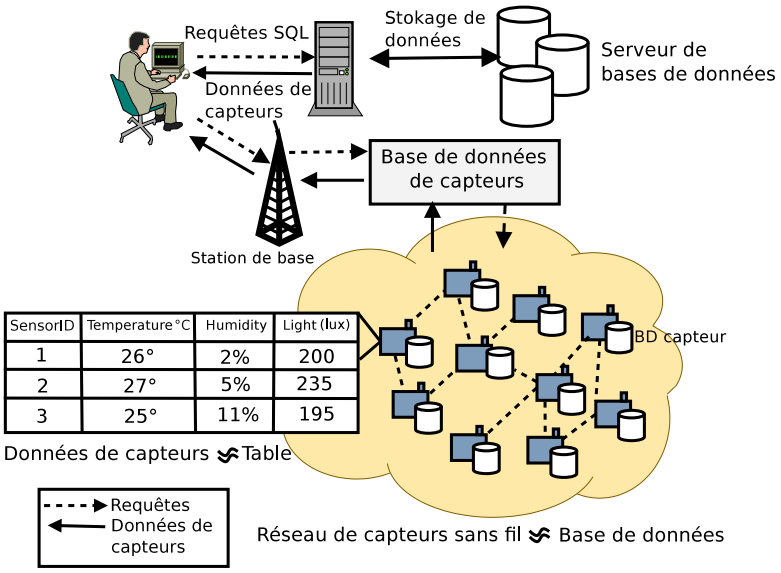


Figure 2.1: Un modèle de réseau avec une bases de données de capteurs sans fil

Les BDCSF doivent avoir certaines propriétés pour que les données des capteurs reflètent l'état actuel de l'environnement. Parmi ces propriétés, nous citons :

- **La persistance :** une base de données de capteurs doit être persistante, indépendamment de la situation du RCSF (en raison de la topologie qui change fréquemment, ou de défaillances des nœuds).

- **La cohérence :** la requête de l'utilisateur doit atteindre le nœud de destination pour obtenir les données demandées. Lorsque la topologie du réseau change, la table de routage doit être mise à jour et la requête sera redirigée vers une autre source de données, afin de maintenir une base de données cohérente et pour satisfaire les demandes des utilisateurs.

- **L'équilibrage de charge :** le stockage des données de capteurs doit être distribué sur plusieurs nœuds dans le RCSF. Un stockage concentré sur un nœud unique peut provoquer une surcharge et une perte de données. En outre, le processus de traitement des requêtes doit être équilibré sur plusieurs nœuds, afin de réduire la charge de travail du RCSF.

- **L'extensibilité :** le stockage des données et la capacité de traitement des requêtes augmente avec l'augmentation du nombre de nœuds. Le RCSF doit tenir compte de ces contraintes. Cependant, les nœuds de capteurs ont des capacités de calcul et d'énergie limitées. Pour ces raisons, il est nécessaire de réduire le nombre de communications, pour conserver l'énergie dans le réseau et améliorer la latence des requêtes dans le RCSF.

- **La concurrence :** une base de données de capteurs peut être utilisée par plusieurs utilisateurs, en même temps. Elle doit soutenir les multiples accès de requêtes, tout en respectant les délais et la validité temporelle des données.
- **L'accès sécurisé aux données :** les bases de données de capteurs doivent assurer et sécuriser l'accès aux données des capteurs, afin de garantir la sécurité du système.

2.3.2 Différence avec les bases de données classiques

Les BDCSF sont différents des systèmes de bases de données distribuées traditionnels par les points suivants :

- Le traitement des données dans les RCSF est fortement lié à l'environnement du réseau. Cet environnement est composé d'un ensemble de capteurs caractérisés par un espace de stockage, une capacité de calcul limitée et une faible autonomie. Au contraire, les systèmes de bases de données distribuées traditionnels sont installés sur des machines performantes qui n'ont souvent aucune contrainte d'énergie ou de calcul.
- Les bases de données traditionnelles peuvent contenir des données statiques, *i.e.* des données qui ne sont que rarement modifiées. Les données historiques qui stockent les dernières mises à jour sur les données peuvent aussi être considérées comme des données statiques. Par contre, les systèmes de gestion de données pour RCSF utilisent des données dynamiques (flux continus de données) et les réponses aux requêtes sont souvent approximatives.
- Les techniques de traitement des requêtes dans les systèmes de bases de données distribuées traditionnels ne s'adaptent pas aux RCSF : les bases de données traditionnelles utilisent des fonctions de verrouillage pour synchroniser l'accès simultané de plusieurs utilisateurs à la même donnée. Ces fonctions ne conviennent pas aux flux de données continues et aux requêtes de longue durée. Les techniques d'optimisation des requêtes dans les systèmes de bases de données distribuées traditionnels sont basées sur des modèles à coût fixe et sur des informations statiques. Les systèmes de gestion de données pour RCSF utilisent un plan de requêtes basé sur le coût de la consommation d'énergie et sur des données dynamiques.
- Les systèmes de gestion de données pour RCSF ont des opérateurs supplémentaires pour fixer la durée de vie et le cycle d'exécution d'une requête.
- Les tables d'une base de données traditionnelle sont stockées sous forme de fichiers sur une unité de stockage. Les tables dans un réseau de capteurs sont virtuelles. Elles

représentent des vues relationnelles générées par le réseau de capteurs [Govindan *et al.*, 2002].

- Le modèle de coût pour les bases de données traditionnelles est différent du modèle de coût pour les bases de données de capteurs. Dans une base de données traditionnelle, le coût d'un plan de requêtes est estimé sur la base du nombre d'enregistrements accessibles, le nombre d'accès au disque, ou le nombre de résultats. Dans les RCSF, le coût d'un plan de requêtes est estimé sur la base de l'énergie. Pour calculer l'énergie nécessaire pour traiter un plan de requêtes, il est nécessaire de calculer : (i) l'énergie nécessaire pour envoyer une donnée sur chaque flux, ainsi que (ii) la fréquence des données par flux.

2.3.3 Architecture et fonctionnement

Un système de BDCSF possède une architecture similaire à celle d'une architecture de système de gestion de bases de données (SGBD). Elles partagent certains composants comme le processeur de requêtes, l'optimiseur de requêtes et des fonctionnalités telles que l'analyse et la transformation des requêtes pour qu'elles soient exécutables par le RCSF. L'architecture d'une base de données de capteurs diffère d'un système à un autre, mais en général, elle est composée d'éléments de base, qui sont les suivants (cf. Figure 2.2):

- **Analyseur de requêtes :** il vérifie la syntaxe et la sémantique de la requête. La vérification syntaxique consiste à vérifier si l'orthographe de la requête ainsi que d'ordre des mots SQL correspondent à la structure syntaxique de langage SQL et à vérifier que les attributs et les relations utilisés dans la requête existent déjà dans le dictionnaire du système. La vérification sémantique consiste à vérifier la cohérence entre les opérations demandées dans la requête.

- **Processeur de requêtes :** il est responsable de la conversion des requêtes reçues de l'utilisateur et écrites en langage SQL en un plan d'exécution de requêtes efficace, qui peut être exécuté par le réseau de capteurs. Ce plan de requêtes sera envoyé dans le réseau, pour récupérer des données demandées par l'utilisateur.

- **Optimiseur de requêtes :** lorsque la station de base reçoit la requête de l'utilisateur, l'optimiseur de requêtes est invité à générer un plan d'exécution de requêtes optimisé, pour accéder aux données requises par la requête. Le plan de requêtes spécifie les détails précis de l'exécution de la requête. Il doit être exécuté avec un minimum d'opérations, d'énergie et en un temps de traitement minimal.

- **Protocole de routage :** il définit des stratégies efficaces pour transmettre des données à partir de la station de base aux capteurs et recueillir des données à partir

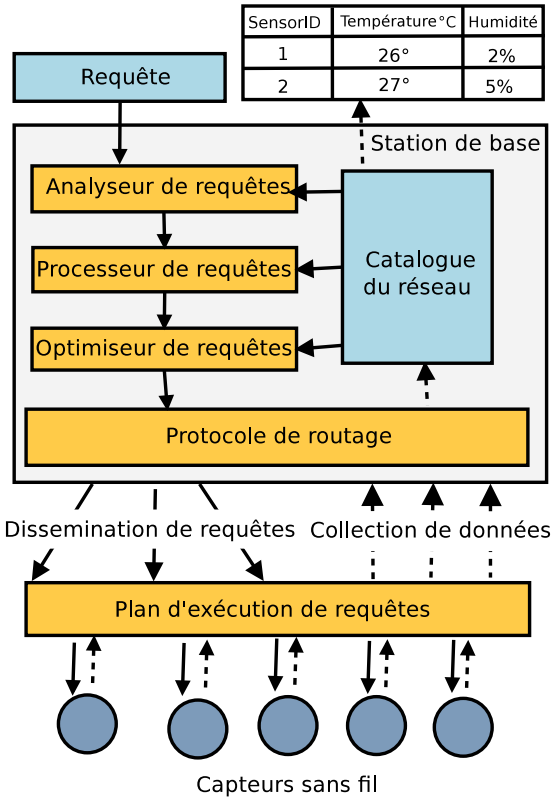


Figure 2.2: Architecture d'une base de données de capteurs sans fil

des nœuds de capteurs à la station de base.

- **Le catalogue du réseau :** il représente le catalogue et le gestionnaire de schémas pour le processeur de requêtes. Il contient les différents attributs et les relations utilisés dans les requêtes.

2.3.4 Langage des requêtes

Les systèmes de bases de données de capteurs fournissent des langages de requêtes similaires au langage SQL. La grammaire du langage est différente d'un système à un autre. Par exemple, nous trouvons les opérateurs pour l'agrégation telles que la somme (`SUM`), la moyenne (`AVERAGE`) ou le minimum et le maximum (`MIN, MAX`), le regroupement (`GROUP BY`), la sélection (`SELECT`) et la jointure (`JOIN`). Il existe d'autres opérateurs adaptés au contexte des réseaux de capteurs. Ils sont utilisés pour déclencher des évènements (`TRIGGER ACTION`). Le résultat de ces opérateurs est une table relationnelle, qui peut servir d'entrée à d'autres opérateurs.

2.3.5 Types de requêtes

Dans les RCSF, les requêtes représentent des demandes d'une station de base aux capteurs pour envoyer leurs données prélevées [Rahman and Hussain, 2007]. Elles peuvent être classées en fonction de critères tels que la fréquence des réponses qui peut être périodique, à la demande, etc. ou la quantité de données, les événements ou la durée de vie [Jun Zheng, 2009].

2.3.5.1 Fréquence des requêtes

- **Requêtes périodiques** : elles consistent à interroger périodiquement un ou plusieurs capteurs selon un intervalle de temps fixé par l'utilisateur pour, par exemple, surveiller une zone géographique. Ce type de requêtes peut être lancé depuis un instant donné jusqu'à un temps logique infini [Jun Zheng, 2009].

- **Requêtes historiques** : elles sont utilisées pour interroger des données historiques stockées dans une station de base ou dans un nœud du réseau. Cependant, les capteurs ayant une capacité de stockage limitée, la plupart des données sont stockées dans des bases données distantes.

- **Requêtes instantanées** : elles consistent à interroger une seule fois un ou plusieurs capteurs à un instant donné, pour donner une vue instantanée sur une zone surveillée par le réseau.

2.3.5.2 Quantité de données

- **Requêtes de sélection** : elles permettent l'extraction des données de capteurs qui satisfont les conditions de la clause <WHERE> dans une requête. Ces requêtes sont efficaces pour extraire des mesures sélectives de certains capteurs. Par exemple, la requête "SELECT température, humidity FROM sensors WHERE location in Region and sensor_ID=123", signifie qu'on demande la valeur de la température et de l'humidité à extraire depuis le capteur localisé dans une région et dont l'identifiant est 123.

- **Requêtes d'agrégation** : ces requêtes sont appliquées quand les capteurs doivent envoyer une grande quantité de données à travers le réseau. Elles permettent d'éviter la redondance d'informations provenant de plusieurs sources. Elles utilisent des fonctions telles que MIN, MAX, SUM, COUNT et AVG pour manipuler ces données.

2.3.5.3 Événements et durée de vie

- **Requêtes basées sur des événements** : ces requêtes sont déclenchées dès qu'un événement remplit les conditions de la requête. Par exemple, dans le cas d'un incendie, si la température atteint un degré supérieur à 100, les capteurs envoient de nouvelles mesures de température toutes les cinq secondes. On écrit alors la requête : "`ON EVENT température > 100 SELECT nodeid, température FROM sensors EVERY 5 seconds`". Ces requêtes peuvent réduire considérablement la quantité d'énergie utilisée dans le réseau.

- **Requêtes basées sur la durée de vie** : ces requêtes sont définies pour fixer une durée de vie pour les requêtes à longue durée d'exécution, car ces dernières n'ont pas de limite pour leur temps d'exécution. Les systèmes de traitement de requêtes tels que TinyDB et Cougar implémentent ce type de requêtes. TinyDB utilise la clause `<LIFETIME <valeur>>` et Cougar utilise les clauses `<DURATION>` et `<EVERY>` pour définir la durée de vie de la requête.

2.3.5.4 Précision des réponses

- **Requêtes de précision** : elles sont moins fréquentes, mais utiles pour les applications critiques telles que la surveillance des systèmes de santé. Elles expriment le besoin d'extraire des données précises de capteurs et supposent une communication fiable pour assurer une livraison de données précises.

- **Requêtes approximatives** : elles reflètent la tolérance de l'erreur sur les données en intégrant un seuil d'erreur (souvent suivi d'un seuil de confiance). Par exemple, la requête suivante : "SELECT nodeid, température FROM sensors ERROR 2 CONFIDENCE 95 %", indique que la différence entre la valeur de l'erreur déclarée par chaque capteur, c.à.d. la valeur absolue de la quantité (valeur réelle - valeur détectée), doit être délimitée par un seuil d'erreur égal à 2, avec un seuil de confiance de 95 %. La tolérance d'une certaine marge d'erreur peut réduire considérablement l'énergie dépensée dans la propagation du résultat [Trigoni *et al.*, 2007].

2.3.5.5 Nature de la recherche

Parmi les requêtes basées sur la nature de la recherche, nous citons :

- **Requêtes spatiales** : elles concernent les données produites dans un emplacement. Par exemple, la requête "`SELECT température FROM sensor WHERE location`

= (50,60)", cherche la température d'un capteur situé dans un point du plan de coordonnées (50,60).

- **Requêtes temporelles** : elles concernent les données produites à une certaine date. Par exemple, la requête "SELECT température FROM sensor WHERE date = '12/12/2012'", recherche la température de chaque capteur à la date du '12/12/2012'.

- **Requêtes spatio-temporelles** : elles concernent les données produites à une date et dans un emplacement donnés. Par exemple, la requête "SELECT température FROM sensor WHERE date = '12/12/2012' and location = (50,60)", recherche la température d'un capteur localisé aux coordonnées (50,60) et à la date du '12/12/2012'.

2.3.6 Systèmes de BDCSF

2.3.6.1 SINA

SINA [Shen *et al.*, 2001] est un intergiciel qui permet aux applications *capteurs* d'émettre des requêtes à travers le réseau et de collecter les réponses. SINA s'exécute sur chaque nœud du réseau. Il fournit une organisation des informations collectées et une surveillance des événements. Pour assurer l'efficacité énergétique et l'évolution des opérations, les nœuds du réseau sont regroupés en *clusters* (groupes). SINA considère le réseau de capteurs comme une collection de fiches de calcul. Chaque fiche contient l'ensemble des attributs de chaque capteur et chaque attribut est considéré comme une cellule. Au départ, la fiche technique de chaque capteur contient un petit nombre d'attributs prédéfinis (alimentation, taux de transfert, sensibilité, processeur, mémoire etc.). Ces fiches peuvent être demandées par d'autres nœuds pour créer de nouvelles cellules. Chaque nouvelle cellule transformée doit avoir un nom unique et devient par la suite l'attribut d'un nœud. L'architecture de SINA comprend les fonctionnalités suivantes (cf. Figure 2.3) :

- Regroupement hiérarchique : les nœuds sont regroupés sous forme de *clusters* en fonction de leur niveau de puissance et de leur position. Le processus d'agrégation est appliqué d'une manière récursive pour former une hiérarchie de groupes. Chaque *cluster* doit avoir un responsable qui s'occupe du filtrage, de la fusion, de l'agrégation des données et des calculs périodiques. Si le chef de *cluster* rencontre des problèmes, le processus de regroupement sera initié et un nouveau chef de *cluster* sera élu.

- Identification des capteurs par attribut : avec un grand nombre de capteurs, il est difficile de contrôler chaque nœud. Srisathapornphat *et al.* [Srisathapornphat *et al.*, 2001] proposent d'identifier les nœuds cibles en utilisant des paires d'attributs. Par exemple, la définition [type=température, location=NE, température=103] décrit

tous les capteurs de température situés dans le secteur nord-est avec une lecture de température de 103°C.

- Détection de l'emplacement des capteurs : le système de positionnement global (GPS) fournit des informations de positionnement absolu des capteurs. Un seul sous-ensemble de capteurs peut être équipé d'un récepteur GPS pour que les autres nœuds puissent récupérer leurs positions approximatives dans la zone. SINA utilise aussi les dispositifs de poursuite optique qui donnent une grande précision sur la localisation des nœuds. La figure 2.3 présente un modèle de réseau de capteurs avec l'intergiciel SINA. L'intergiciel SINA avec les traits en pointillés sur la figure 2.3, permet aux applications *capteurs* d'émettre des requêtes et des commandes vers les capteurs, puis de recueillir les réponses et les résultats par la suite.

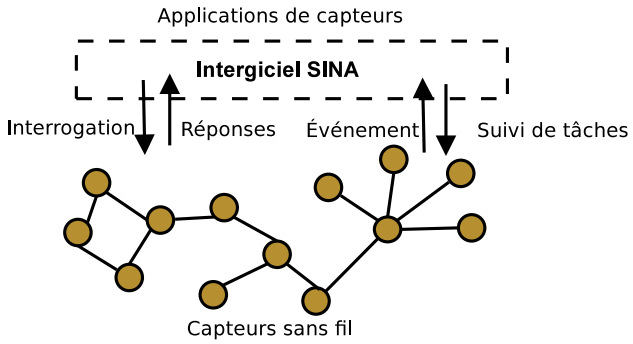


Figure 2.3: Un modèle de réseau de capteurs avec l'intergiciel SINA

SINA utilise trois méthodes pour effectuer la collecte d'informations :

- Les opérations d'échantillonnage : cela permet de réduire la grande quantité de réponses. Les nœuds doivent prendre des décisions autonomes pour répondre ou non aux demandes d'informations. SINA calcule la probabilité d'une réponse à partir du nombre de réponses dans chaque groupe de capteurs.

- Les opérations auto-orchestrées : dans les réseaux possédant un minimum de nœuds, l'exactitude du résultat final dépend des réponses de tous les nœuds dans le réseau. Pour réduire les risques de collision, SINA implémente une approche qui consiste à retarder l'envoi des réponses pendant un laps de temps pour chaque nœud.

- Les opérations de diffusion de calcul : les algorithmes utilisés pour la collecte d'informations sont limités par la capacité de communication des nœuds. Chaque capteur n'est censé communiquer qu'avec ses voisins. La logique d'agrégation des informations est programmée avec le langage SQTL (*Sensor Query and Tasking Language*) [Chaiporn *et al.*, 2000] et diffusée auprès des capteurs. Un des inconvénients est que cette logique est difficile à implémenter dans des réseaux de taille importante.

SINA utilise un langage de requête qui s'appelle "SQTL", qui est conçu pour être flexible et compact [Srisathapornphat *et al.*, 2001]. Il interprète les requêtes déclaratives entre les applications de capteurs et l'intergiciel SINA, qui offre un accès au matériel et aux primitives de communication. Il fournit une gestion appropriée des événements pour les applications des réseaux de capteurs. SINA prend en charge trois types d'événements : (i) les événements générés lorsqu'un message est reçu par un capteur, (ii) les événements déclenchés périodiquement par une minuterie et (iii) les événements provoqués par l'expiration d'un temporisateur. Voici un exemple de modèle de déclaration de requêtes [Srisathapornphat *et al.*, 2001] qui permet de calculer la moyenne des températures des clusters formant le réseau :

Requête 1 (La grammaire du langage de requêtes de SINA).

```
SELECT avg(getTemperature())
AS avgTemperature
FROM CLUSTER-MEMBERS
```

2.3.6.2 Cougar

Le projet Cougar [Yao and Gehrke, 2002] a été réalisé à l'Université Cornell (USA). Il présente une infrastructure dédiée à la gestion des données de capteurs. Cougar est basé sur des capteurs pré-programmés et sur une entité centrale à laquelle les données sont agrégées et stockées pour tester des techniques de traitement des requêtes sur les réseaux de capteurs sans fil [Fung *et al.*, 2002]. Cougar considère le réseau de capteurs comme une grande bases de données distribuée où les données sont assimilées à une table relationnelle. Les attributs de cette table sont les attributs des capteurs et les valeurs sont les mesures. Les nœuds sont organisés en groupes (*clusters*), dont chacun possède un chef de *cluster*, qui joue le rôle d'un point d'agrégation de données [Makhoul, 2008]. L'architecture de Cougar est composée des éléments suivants (cf. Figure 2.4) :

- Le proxy *Requête* qui est une petite application associée à chaque capteur, entre sa couche réseau et sa couche application, pour interpréter et exécuter les requêtes échangées entre le capteur et la station de base [Jun Zheng, 2009]. Le proxy *Requête* est responsable des communications entre les capteurs, la station de base et les requêtes *utilisateur* (cf. Figure 2.4). Il regroupe et élimine les données non pertinentes.

- Le composant *Frontal* qui joue le rôle de passerelle entre l'interface graphique et le réseau de capteurs (cf. Figure 2.4). Il exécute les tâches suivantes : (i) il envoie les requêtes qu'il reçoit de l'interface graphique vers le *proxy requête* qui les traite,

(ii) il reçoit les résultats des chefs de *clusters*, en gardant une trace des requêtes en cours d'exécution pour l'interface graphique, (iii) il fournit les tuples nécessaires aux requêtes et envoie par la suite les résultats à l'interface *utilisateur*. Il peut émettre aussi des tuples de données vers une base de données *MySQL* installée sur un serveur central.

- L'interface graphique qui permet à l'utilisateur d'interroger le réseau de capteurs via des requêtes qui lui permettent de visualiser la topologie du réseau (cf. Figure 2.4). Les résultats des requêtes sont affichés à l'utilisateur sous forme d'un tableau.

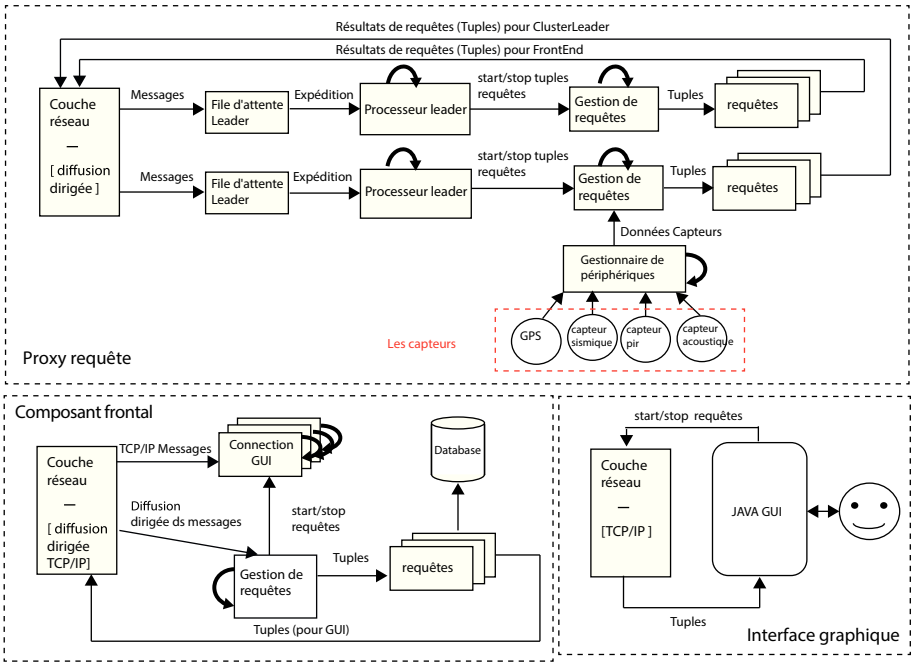


Figure 2.4: Architecture du système Cougar

Cougar [Fung *et al.*, 2002] définit un capteur en tant qu'objet de type abstrait, ADT (Abstract Data Type) pour tous les capteurs de même type. Les bases de données *objets-relationnelles* supportent les données de type abstrait. En effet, un objet *ADT* dans une base de données de capteurs correspond à un capteur physique dans le monde réel [Bonnet *et al.*, 2001]. L'interface publique d'un capteur correspond aux fonctions spécifiques de traitements du signal. L'abstraction permet à l'utilisateur de ne pas connaître la topologie du réseau de capteurs, la manière de collecter et de traiter les données et la façon d'envoyer les résultats.

Cougar regroupe les capteurs sous forme de *clusters*. L'agrégation des données permet d'économiser l'énergie et de réduire le flux de communication entre les nœuds. Chaque nœud contient une liste de tous les nœuds à qui il doit envoyer les données. Le nœud responsable de l'agrégation des données demande les tuples nécessaires au gestionnaire

de périphériques, qui lui transmet les mesures provenant des capteurs. Le nœud agrégat combine les données locales avec les données agrégées partiellement, reçues par d'autres capteurs. Ensuite, il envoie le résultat au chef de *cluster* qui calcule la moyenne des résultats agrégés et les compare à un seuil défini par l'utilisateur. Si la moyenne calculée est supérieure au seuil, alors le chef de *cluster* envoie les résultats à la passerelle (Frontal), qui seront présentés par la suite à l'utilisateur sous forme d'un tableau.

Cougar utilise un langage de requêtes dont le syntaxe est similaire à celle de SQL. Ce langage assure la continuité des requêtes pour fournir des résultats à l'utilisateur. Le canevas de requête suivant illustre la grammaire de ce langage (cf. Requête 2).

Requête 2 (La grammaire du langage de requêtes de Cougar)**.**

```
SELECT <select-list>
FROM <Sensordata>
[WHERE <predicate>]
[GROUP BY <attributes>]
[HAVING <predicate>]
DURATION <time-interval>
EVERY <time-span>
```

Dans la requête 3, les clauses `<WHERE>`, `<SELECT>`, `<GROUP BY>` et `<HAVING>` ont la même signification que dans le standard SQL, c'est-à-dire : la clause SELECT permet de récupérer les données, la clause `<WHERE>` permet de filtrer les n-uplets en imposant une condition à remplir, la clause `<GROUP BY>` regroupe les capteurs selon leurs attributs et la clause `<HAVING>` agit comme le filtre `<WHERE>`, mais sur les opérations résultant des regroupements. Les clauses `<DURATION>` et `<EVERY>` sont spécifiques au langage SQL de Cougar. Ainsi, la clause `<DURATION>` spécifie la durée de vie de la requête et la clause `<EVERY>` détermine le cycle d'exécution de la requête.

2.3.6.3 TinyDB

TinyDB est un système d'exécution de requêtes sur un réseau de capteurs, intégré au système d'exploitation TinyOS [Mazzer and Tourancheau, 2008]. Il a été développé à l'université de Californie. Il s'agit d'un système de gestion de bases de données destiné à l'extraction d'informations à partir d'un réseau de capteurs ayant des ressources limitées [Madden *et al.*, 2003b]. TinyDB collecte les données de capteurs, les filtre les agrège et les achemine vers un serveur central.

Dans l'utilisation de TinyDB, on distingue trois parties (cf. figure 2.5) : l'utilisateur, le

serveur et le réseau de capteurs. L'utilisateur peut se connecter au réseau à travers la station de base via un port série et les capteurs peuvent communiquer entre eux à travers le réseau sans fil. La machine de l'utilisateur doit contenir l'application TinyDB pour gérer les requêtes *utilisateur*. Elle les analyse, les décompose, puis les envoie aux nœuds du réseau à travers la station de base. Dès que celle-ci reçoit les résultats des requêtes, elle les transmet à l'utilisateur.

Le système TinyDB possède deux sous-systèmes : une application pour les réseaux de capteurs et une interface graphique pour l'utilisateur en Java. L'application TinyDB s'exécute sur chaque capteur. Elle est composée des parties suivantes (cf. Figure 2.5) [Li *et al.*, 2008] :

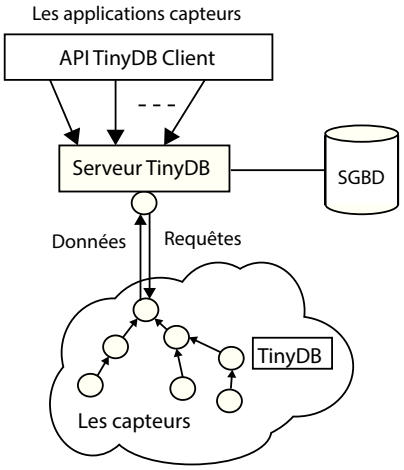


Figure 2.5: Architecture du système TinyDB

1. Un catalogue et un gestionnaire de schéma : pour le suivi de l'ensemble des attributs, ou des types de mesures à lire (lumière, température, humidité) , ou les propriétés du capteur tel que son identifiant et sa position. Le schéma du capteur décrit sa capacité à contenir des attributs comme une table virtuelle.

2. Un processeur de requêtes : le processeur de requêtes utilise le catalogue du capteur pour extraire les valeurs des attributs locaux, recevoir les données de capteurs voisins, agréger l'ensemble des données, les filtrer et les acheminer vers les nœuds parents.

3. Un gestionnaire de mémoire : TinyDB peut gérer l'espace mémoire d'une manière dynamique et stocker les données de façon compacte pour économiser l'espace de stockage.

4. Un gestionnaire de topologies réseaux : TinyDB gère de manière efficace la connectivité des nœuds. Il maintient les tables de routage et veille à ce que les données et les résultats des requêtes soient bien acheminés à travers le réseau.

TinyDB possède deux types d'interface *client* [Madden *et al.*, 2003a]. Une interface basée sur un langage de requête SQL, appelé TinySQL et une autre interface graphique constituée d'un ensemble d'applications et de classes Java.

TinyDB fournit un langage déclaratif simple (TinySQL) et une interface graphique pour la collecte et l'agrégation de données [Madden *et al.*, 2005]. Le langage de requêtes TinySQL comprend la sélection, la projection, la détermination du taux d'échantillonnage, l'agrégation, les déclencheurs, la durée de vie d'une requête, la mise à jour et les jointures simples [Patil and Patil, 2011]. La grammaire du langage de requêtes TinySQL est présentée dans la requête 3.

Requête 3 (La grammaire du langage de requêtes de TinyDB)**.**

```
SELECT select-list [FROM sensors]
WHERE predicate
[GROUP BY gb-list]
[HAVING predicate]
[TRIGGER ACTION command-name[(param)]]
[EPOCH DURATION time]
```

Dans le canevas de la requête 4, les clauses `<WHERE>`, `<GROUP BY>` et `<HAVING>` ont la même signification que pour le standard SQL. Les clauses `<TRIGGER ACTION>` et `<EPOCH DURATION>` sont spécifiques au langage TinySQL. La clause `<TRIGGER ACTION>` permet de déclencher une action telle qu'une alarme et la clause `<EPOCH DURATION>` définit la durée d'un intervalle d'échantillonnage.

TinyDB comprend un déclencheur d'actions [Madden *et al.*, 2003a]. Une action est déclenchée lorsque le résultat satisfait la clause `<WHERE>` de la requête (cf. Requête 5).

Requête 4 (Exemple de requête de TinyDB)**.**

```
SELECT temp FROM sensors
WHERE temp > thresh
TRIGGER ACTION SetSnd(512)
EPOCH DURATION 512
```

La requête 5 correspond à une alarme déclenchée à chaque fois que la température dépasse un seuil fixé par l'utilisateur. SetSnd(512) indique que la durée sonore de l'alarme est de 512 ms.

TinyDB possède un mécanisme d'agrégation qui permet de combiner les messages du

réseau, ce qui réduit le nombre de messages à envoyer. Il utilise aussi un système de gestion de méta-données qui prend en charge les optimisations de l'agrégation. Le mécanisme d'agrégation est représenté par une structure arborescente [Makhoul, 2008], où la racine est unestation de base et les feuilles de l'arbre constituent tous les nœuds du réseau de capteurs. Le canevas de la requête 6 présente une requête d'agrégation [Patil and Patil, 2011]. La requête 6 retourne les identifiants de nœuds, leur luminosité et leur température moyenne lorsque la mesure de la lumière est inférieure ou égale à 100, pour une période de cinq minutes.

Requête 5 (Exemple de requête d'agrégation avec TinyDB).

```
SELECT idCluster, AVG(light), AVG(temp)
FROM sensors
WHERE light<=100
GROUP BY idCluster
EPOCH DURATION 5min
```

L'implémentation actuelle de TinyDB souffre de quelques faiblesses : (i) il subsiste des erreurs relatives aux fonctions d'agrégation : elles retournent les mêmes valeurs pour différentes requêtes agrégées, (ii) il manque des fonctions de filtrage : la clause HAVING n'est pas implémentée dans le langage de requêtes, (iii) il existe une perte considérable de messages dans le réseau [Kofoed, 2007]. Bien que TinyDB puisse stocker les résultats des requêtes dans une base de données PostgreSQL, cette option ne fonctionne que pour les requêtes non agrégées.

2.3.6.4 Antelope

Antelope est un système de gestion de bases de données spécifique aux réseaux de capteurs à ressources limitées. Il fonctionne au-dessus du système d'exploitation Contiki [Tsiftes and Dunkels, 2011]. Il propose un stockage des données dans le capteur lui-même et fournit un ensemble d'opérateurs relationnels permettant la création dynamique et l'interrogation des bases de données. Antelope met en œuvre une abstraction du stockage en utilisant un système de fichiers, appelé Coffee [Tsiftes *et al.*, 2009], qui permet de réduire la complexité des systèmes de stockage.

Antelope est composé de huit éléments (cf. Figure 2.6) : (1) un processeur basé sur le langage AQL (Antelope Query Langage) pour l'analyse des requêtes, (2) un contrôleur de confidentialité pour vérifier que les requêtes sont autorisées, (3) un noyau qui contient la logique de la base de données et les coordonnées de l'exécution des requêtes, (4) une machine virtuelle pour exécuter les requêtes, (5) l'indice abstrait qui contient la logique

d'indexation, (6) le processus d'indexation pour construire des index à partir des données existantes, (7) l'abstraction des ressources de stockage à travers le système de fichier *Coffee*, qui permet des opérations sur des fichiers stockés sur une mémoire flash externe et (8) le transformateur de résultats pour améliorer leur présentation à l'utilisateur.
Antelope utilise le langage de requête AQL défini pour construire et interroger les bases de données dans les réseaux de capteurs. AQL n'est pas un sous-ensemble de SQL, mais les deux langages partagent des éléments syntaxiques. Il exclut toutes les fonctionnalités SQL complexes comme les extensions procédurales, les déclencheurs et les transactions [Tsiftes and Dunkels, 2011].

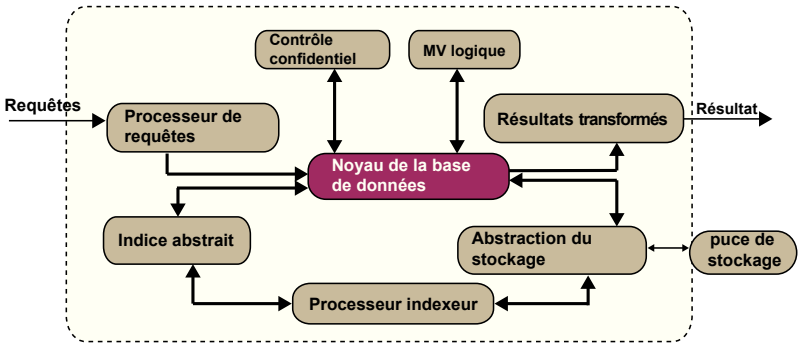


Figure 2.6: Architecture du système Antelope

L'analyseur AQL traduit les opérations d'une représentation textuelle en données de calcul abstraites et permet l'exécution de plusieurs requêtes simultanément. La création d'une base de données avec AQL passe par trois étapes : la création de la relation, puis de ses attributs et enfin de ses index (cf. Figure 2.7).

```
CREATE RELATION sensor;
CREATE ATTRIBUTE id DOMAIN INT IN sensor;
CREATE ATTRIBUTE name DOMAIN STRING(20) IN sensor;
CREATE ATTRIBUTE position DOMAIN LONG IN sensor;
CREATE INDEX sensor.id TYPE INLINE;
CREATE INDEX sensor.position TYPE MAXHEAP;
```

Figure 2.7: Création d'une base de données avec le langage AQL

Dans le canevas de requêtes (cf. Figure 2.7), nous présentons les étapes de création d'une base de données avec Antelope en utilisant les clauses CREATE RELATION, CREATE ATTRIBUTE et CREATE INDEX. Une fois la base de données créée, l'utilisateur peut l'interroger en utilisant les opérations listées dans la figure 2.8.

```
Opération        | Usage
-------------------------------------------
INSERT           | Insérer un tuple dans une relation
REMOVE           | Supprimer les tuples correspondant à une condition
SELECT           | Sélectionner les tuples et les attributs du projet
JOIN             | Joindre deux relations selon une condition
CREATE RELATION  | Créer une relation vide
REMOVE RELATION  | Supprimer une relation et tous les index associés
CREATE ATTRIBUTE | Ajouter un attribut à une relation
CREATE INDEX     | Créer un index d'attribut
REMOVE INDEX     | Supprimer un index d'attribut
```

Figure 2.8: Les opérations dans le langage AQL

Le noyau de la base de données est appelé par le module d'exécution des requêtes pour évaluer les opérations AQL. Il est adapté aux systèmes embarqués utilisant souvent un ordonnancement coopératif pour assurer aux processus une exécution jusqu'à la fin sans préemption.

Antelope utilise trois algorithmes d'indexation pour optimiser l'exécution des requêtes dans les bases de données en délimitant l'ensemble des tuples à traiter [Tsiftes and Dunkels, 2011] : (i) *MaxHeap Index* qui réduit la consommation d'énergie et l'utilisation de la mémoire, (ii) *Inline Index* qui assemble les données de capteurs et d'autres données et (iii) *Hash Index* qui stocke les paires « clé-valeur » dans la mémoire vive pour les relations utilisées dans les requêtes.

2.3.6.5 Corona

Corona est un système de traitement des requêtes pour les RCSF utilisant des capteurs SunSPOT [Khoury *et al.*, 2010], développé à l'école d'informatique de l'Université de Sydney. Il fournit un langage déclaratif de type SQL pour formuler les requêtes. Corona introduit la notion de fraîcheur de données qui permet à l'utilisateur d'obtenir des données fraîches à partir du RCSF et il est capable d'exécuter plusieurs applications simultanément. Il utilise des requêtes découplées qui peuvent fonctionner avec des périodes d'activation beaucoup plus longues et il regroupe les lectures du capteur autour de valeurs fréquentes, dans un stockage local sur les nœuds, pour réduire les efforts de communication et améliorer la durée de vie du réseau.

Le système Corona comporte trois composantes (cf. Figure 2.9):

- **Le processeur de requêtes :** ce programme est installé sur les nœuds de capteurs.

Il inclut un planificateur qui permet d'ajouter ou de supprimer les tâches dans la file d'attente, ainsi que de maintenir celles qui sont actives. Les tâches sont exécutées suivant leurs dates de déclenchement. L'utilisateur peut fixer le nombre et la période d'exécution des tâches. Le processeur de requêtes s'occupe aussi de la synchronisation et du maintien des horloges des tâches. Il supporte les différents opérateurs de requêtes tels que la projection, la sélection, l'agrégation, la jointure et l'opération de regroupement.

- **Le système de contrôle :** il fonctionne sur l'ordinateur de l'utilisateur et il est chargé par l'analyse et l'optimisation des requêtes. Il les traduit sous forme d'opérations relationnelles pour les transmettre par la suite au RCSF. Il fournit le temps global pour les nœuds du réseau. Il recueille les données en provenance du réseau et les affiche à l'utilisateur.
- **L'interface graphique :** elle est utilisée pour la saisie des requêtes et pour l'affichage des résultats.

Figure 2.9: Architecture du système Corona

Corona fournit un langage de requêtes de type SQL, qui supporte les opérateurs fondamentaux des requêtes SQL comme la sélection, la projection, la jointure, l'agrégation et l'opération de regroupement. Il permet aussi aux utilisateurs d'interroger les capteurs pour avoir les données détectées comme la lumière, la température, ainsi que les informations sur le capteur comme l'identifiant, le parent du nœud et le temps. L'utilisateur peut spécifier les heures de début de requête, son époque et sa durée de vie. La requête 7 illustre la syntaxe de ce langage.

Requête 6 (La grammaire du langage de requêtes de Corona).

```
SELECT <attributes>
FROM  <SENSORS>
[ WHERE <condition>]
[ GROUP BY <attributes>]
[ HAVING <condition >]
[ START (IN <relative_time> |
         AT <absolute_time>
[ ON DATE <date>]) ]
[ EPOCH  <relative_time>]
[ RUNCOUNT (<integer> | FOREVER ) ]
```

Dans la Requête 7, la clause <SELECT> spécifie les attributs demandés par l'utilisateur. La clause <FROM> identifie la source des données. La clause <WHERE> spécifie les conditions à appliquer pour filtrer et agréger les données. La clause <GROUP BY> spécifie comment les données doivent être regroupés. La clause <HAVING> spécifie les conditions à appliquer après l'agrégation des données. La clause <START IN> définit le retard relatif avant l'exécution d'une requête. La clause <START AT> définit l'heure de début d'exécution d'une requête. La clause <ON DATE> spécifie la date d'exécution d'une requête. La clause <EPOCH> définit le temps qui sépare deux exécutions de requête. La clause <RUNCOUNT> définit le nombre d'exécutions de la requête.

Corona fournit un mécanisme de traitement de requêtes multi-tâches, qui permet l'exécution concurrente des requêtes. De nombreuses applications peuvent accéder en même temps au réseau et obtenir des informations, ce qui réduit le nombre d'infrastructures utilisées. Corona utilise aussi un mécanisme qui permet de ne pas réactiver un capteur par le processeur de requête durant une période. Il utilise une clause SQL nommé "fraîcheur" pour fixer ce temps. Durant cette période, le capteur utilise ses précédentes lectures (qui se trouvent dans sa mémoire cache). La contrainte de fraîcheur aide à optimiser les requêtes, en fixant le décalage horaire entre les dates de début d'exécution des nouvelles requêtes, afin de maximiser la durée de vie du réseau.

2.3.6.6 MaD-WiSe

MaD-WiSe (Management of Data in Wireless Sensor networks) [Amato *et al.*, 2010] est un système de gestion de flux de données dans les RCSF. Il considère le RCSF comme une base de données distribuée, et supporte les différents aspects liés à la conception des systèmes de bases de données. Il fournit un langage de requêtes nommé "MW-SQL", qui

permet d'interroger le RCSF, de recueillir, de filtrer et de gérer les données des capteurs. MW-SQL utilise le concept de source de données pour spécifier les données à récupérer du réseau de capteurs. Il inclut également des agrégats temporels les clauses "`EVERY`" et "`EPOCH`". La clause "`EVERY`" est utilisée pour spécifier le taux de données à récupérer au cours de chaque période d'acquisition. La clause "`EPOCH`" définit l'intervalle de temps entre deux périodes de collecte des données.

Le système MaD-WiSe est constitué d'un ensemble de modules qui s'exécutent dans les nœuds de capteurs et d'autres modules qui s'exécutent au niveau de l'ordinateur de l'utilisateur (le client) (cf. Figure 2.10) [Amato *et al.*, 2010].

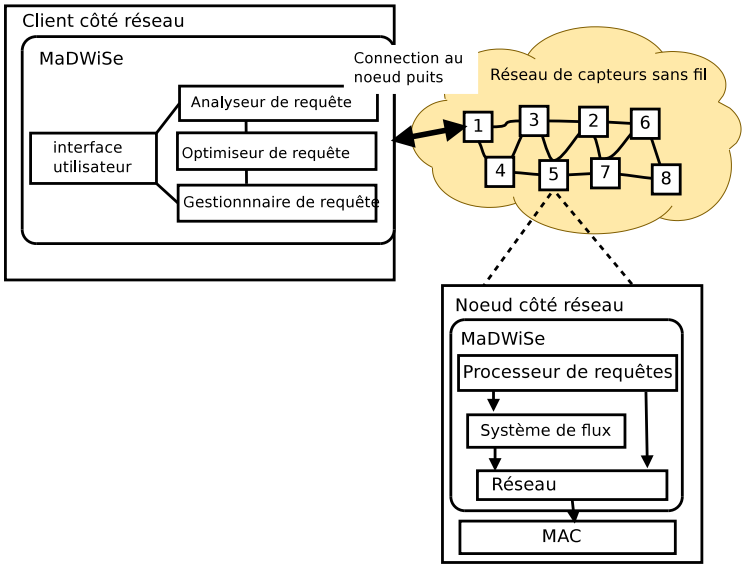


Figure 2.10: Architecture du système MaD-WiSe

- **Sous-système coté "client"** (ordinateur de l'utilisateur) : il contient un analyseur, un optimiseur et un gestionnaire de requêtes. L'analyseur de requêtes traduit les requêtes SQL-Like en plan d'exécution de requêtes initial. L'optimiseur de requêtes génère un plan d'exécution de requêtes optimisé du plan initial. Il organise les nœuds impliqués dans l'exécution de la requête, les opérations à exécuter, l'activation du transducteur et les communications radio, afin de réduire la consommation d'énergie dans le réseau. Le gestionnaire de requêtes diffuse le plan d'exécution optimisé dans le réseau et reçoit par la suite les réponses des capteurs.

- **Sous-système coté "nœud de capteur"** : il contient la couche réseau, la couche système et la couche de traitement de requêtes. La couche de traitement de requêtes implémente le processeur de requêtes du système. Il peut être programmé par le client afin de participer à l'exécution des requêtes distribuées. La couche de flux de système offre des mécanismes d'abstraction pour l'accès aux données au moyen de flux de données. Il peut être considéré comme un système de fichiers dans un

réseau de capteurs, mais dans cet environnement, les données ne sont pas statiques, car les données de capteurs sont générées en flux continus. Le système de flux offre des fonctionnalités pour créer ou supprimer des flux ainsi que lire et écrire des enregistrements de/vers des flux existants. La couche réseau offre à la fois la connexion et des services de communication orienté connexion. Il implémente une application basée sur un protocole qui assure une efficacité énergétique pour les services orientés connexion.

MaD-WiSe utilise un langage de requête nommé "MW-SQL", qui permet aux utilisateurs de formuler des requêtes SQL, de manipuler, de filtrer et d'organiser des séquences de données, générées par les capteurs. Il utilise le concept de source de données pour présenter les réponses aux utilisateurs. La requête 8 présente un exemple de requête MW-SQL :

Requête 7 (La grammaire du langage de requêtes de MaD-WiSe)**.**

```
SELECT select-list
FROM source
[ WHERE condition ]
[ EPOCH samples [ SAMPLES ] ]
[ EVERY rate ]
```

Dans la requête 7, la clause <SELECT> définit les données à récupérer du réseau telles que la température, l'humidité etc. La clause <FROM> définit la source des données. La clause <WHERE> spécifie les conditions sur les données à récupérer du réseau. La clause <EPOCH> définit le nombre d'échantillons consécutifs. La clause <EVERY> est utilisée pour spécifier le taux d'échantillonnage. MW-SQL utilise d'autres concepts tels que l'agrégation spatiale et temporelle, la sélection topologique, la création et le nommage de sources.

MaD-WiSe définit trois types de flux de données pour modéliser les différents modes d'accès. Le premier type concerne les flux de données de capteurs, qui représentent les données acquises par les capteurs. Le deuxième type concerne les flux de données à distance, qui représente les données envoyées d'un nœud source à un nœud destination. Le troisième type concerne les flux locaux, qui représentent des flux de données générés par l'exécution d'opérations locales et qui sont envoyées en entrée à d'autres opérations locales. La modélisation des flux de données fournit un modèle de coût de requêtes optimal. Ce modèle distingue les différents types de flux et génère un plan d'exécution de requêtes qui réduit la consommation d'énergie dans le RCSF.

MW-SQL utilise une algèbre de requêtes inspirée de l'algèbre relationnelle. Cependant, les opérateurs de l'algèbre relationnelle ne sont pas adaptés à la nature des réseaux de capteurs. Cette algèbre supporte un ou deux flux en entrée et un flux en sortie, avec

certains opérateurs et des paramètres spécifiques. MaD-WiSe définit trois opérateurs de base (la sélection, la projection et l'union). Il peut prendre également tout type de flux en entrée ou en sortie. Il fournit une définition spéciale aux opérateurs de jointure (opérateurs utilisés pour relier les tuples en entrée à différents flux), les agrégats spatiaux (utilisés pour les données agrégées, générées par les différents flux) et pour les agrégats temporels (utilisés pour les données agrégées qui arrivent séquentiellement dans les flux).

MW-SQL utilise une approche d'optimisation algébrique basée sur des règles de transformation. Il transforme le plan de requête par un autre, avec un coût en énergie faible. Il applique l'ordre des opérateurs, qui seront affectés aux règles de transformation. Le plan de requêtes final est obtenu après quelques transformations successives, basé sur les flux de données de capteurs.

2.3.7 Comparaison entre systèmes de BDCSF

Dans cette section, nous comparons différents systèmes et bases de données de capteurs. Tout d'abord, nous donnons une comparaison des plates-formes et des dispositifs nécessaires pour mettre en œuvre les bases de données de capteurs. Deuxièmement, nous comparons leur architecture et enfin, nous comparons leurs fonctionnalités.

2.3.7.1 Plate-formes et dispositifs

Les systèmes de bases de données de capteurs récents utilisent les systèmes d'exploitation TinyOS ou ContikiOS, à l'exception du système Corona qui s'exécute à l'aide du logiciel Sun SPOT SDK. TinyDB consiste en une application composée de 140 fichiers et de 25000 lignes [Kellner, 2010]. Elle utilise une mémoire de 60 KO, qui représente à-peu-près la mémoire utilisée dans les capteurs MICAz (64 KO). On ne peut pas tester TinyDB sur des capteurs de type TelosB ayant une mémoire de 48 KO. Cougar utilise des nœuds de type Sensoria, MaD-WiSe utilise des nœuds de type MICAz. Antelope utilise des nœuds de type Tmote Sky (cf. Tableau 2.2).

2.3.7.2 Langage de requêtes

Les systèmes SINA, TinyDB, Cougar, Antelope, Corona et Mad-WISE utilisent un langage de requêtes de type SQL. Seul Antelope fournit un langage de requêtes utilisé à la fois pour formuler les requêtes et pour créer des bases de données [Tsiftes and Dunkels, 2011].

	SINA	Cougar	TinyDB	Antelope	MaD-WiSe	Corona
Année de création	2000	2002	2004	2011	2010	2010
Système d'exploitation	Pas de système	Linux 2.4&Sensoria	TinyOS	ContikiOS	TinyOS	Pas
Simulateur	Glomosim	Sim Tracker	Tossim	Cooja	Tossim	SunSPOT SDK
Type de capteur	Simulator Node	Sensoria WINSNG 2	Mica2	TelosB	Micaz	Sun SPOT
Mémoire de capteur	Inconnu	32 MBytes	128 kBytes	48 kBytes	128 kBytes	2 MBytes
Language SQL-Like	SQTL	SQL-Like	SQL-Like	AQL	MW-SQL	SQL-Like
Interface utilisateur	Pas d'interface	FrontEnd GUI	TinyDB GUI	Cooja interface	SensorViz	Corona GUI

Tableau 2.2: Plate-formes et dispositifs des bases de données de RCSF

2.3.7.3 Architecture des systèmes

Chaque système de bases de données de capteurs a sa propre architecture, mais la plupart de ces systèmes partagent certains composants comme l'interface graphique pour la saisie de requêtes et pour afficher les résultats, le module de traitement de requêtes, y compris l'analyseur et le mécanisme d'optimisation. Cependant, certains systèmes tels qu'Antelope comprennent des composants spécifiques comme le module d'abstraction de stockage et de processus d'indexation pour les données des capteurs.

2.3.7.4 Interface graphique

Cougar et TinyDB fournissent une interface graphique, qui permet à l'utilisateur de saisir des requêtes, pour spécifier les données à extraire à partir du réseau. Les réponses aux requêtes sont affichées par la suite, sous forme de tableaux. SINA ne fournit pas une interface graphique aux utilisateurs, mais il offre une interface procédurale pour saisir des scripts utilisateurs. Antelope ne fournit pas une interface graphique, mais il utilise une application réseau qui s'appelle NetdB. Cette interface permet à l'utilisateur de saisir ses requêtes en ligne de commande. Corona fournit une interface graphique permettant aux utilisateurs la construction des requêtes sans avoir besoin de connaître le langage SQL. Il possède aussi des fonctionnalités permettant la visualisation des résultats des requêtes. MaD-WiSe fournit une application sous forme d'interface graphique nommée SensorViz. Elle permet aux utilisateurs d'interagir avec le réseau de capteurs, en envoyant des requêtes et de visualiser les résultats de la requête par la suite.

2.3.7.5 Agrégation des données

Cougar, TinyDB et Antelope supportent les fonctions d'agrégation telles que COUNT, MAX, MIN et SUM. Le système SINA utilise les fonctions d'agrégation pour former les groupes de capteurs (clusters) et pour faciliter les opérations au sein de réseaux de capteurs [Jaikaeo *et al.*, 2000]. Corona supporte aussi l'agrégation, afin de réduire la grande quantité de données transmises à travers le réseau. MaD-WiSe supporte l'agrégation spatiale et temporelle, en utilisant des opérateurs globaux tels que la moyenne, le maximum et le minimum.

2.3.7.6 Optimisation multi-requêtes

L'optimisation multi-requêtes permet de réduire la communication dans le RCSF. Elle réduit la consommation d'énergie dans le réseau dans le cas d'un accès multi-utilisateurs. Le système SINA ne supporte pas l'optimisation des requêtes. Il utilise un environnement d'exécution pour les capteurs qui s'exécute sur chaque nœud. Il est responsable de l'envoi des messages, de l'examen de tous les messages arrivés et de toutes les opérations appropriées aux actions des capteurs [Jaikaeo *et al.*, 2000]. Cougar et Mad-WISE supportent le processus d'optimisation des requêtes, mais ils ne supportent pas l'optimisation multi-requêtes [Dawborn and Khoury, 2010]. Quant à TinyDB, Antelope et Corona, ils supportent le traitement multi-requêtes, mais ils ne tiennent pas compte de l'optimisation multi-requêtes [Madden *et al.*, 2005] [Tsiftes and Dunkels, 2011].

2.3.7.7 Requêtes basées sue la durée de vie

Cougar et TinyDB utilisent la clause `<DURATION>` dans leurs requêtes qui spécifie la durée de vie de la requête. Sina inclut l'expiration des événements à travers la clause `<EXPIRE>` dans le langage SQLT qui définit les événements qui se produisent quand un temporisateur a expiré. Antelope ne supporte pas la spécification de la durée de vie des requêtes. Avec Corona, les requêtes peuvent avoir une date de démarrage spécifique représentée par les clauses `START IN <relative time>` ou `START AT <absolute time>` ou `START ON DATE <date>` et la période entre l'activation et la durée de vie de la requête représentée par la clause `<EPOCH>`. MaD-WiSe utilise un opérateur d'agrégation temporel qui inclut un intervalle de temps appelé `<EPOCH>`, qui détermine la période durant laquelle les données de capteurs sont agrégées.

2.3.7.8 Requêtes basées sur les évènements

Cougar et Antelope ne supportent pas les requêtes basées sur les évènements. TinyDB supporte les événements pour déclencher la collecte des données. Ces événements sont générés de manière explicite, soit par une autre requête, soit par une partie de niveau inférieur du système d'exploitation [Madden *et al.*, 2005]. TinyDB utilise le prédicat "`ON EVENT`" pour définir un évènement dans une requête. SQTL, le langage de script de Sina, supporte trois types d'événements et utilise des prédicats pour définir ces événements : (i) `RECEIVE`, pour les événements générés quand un message est reçu par un capteur, (ii) `EVERY`, pour les événements déclenchés périodiquement par un temporisateur et (iii) `EXPIRE`, pour les événements déclenchés par l'expiration du temporisateur [Jaikaeo *et al.*, 2000]. Corona supporte les processus multi-tâches. Il permet l'exécution de plusieurs requêtes simultanément, mais il ne peut pas capturer facilement des événements spatio-temporels [Alsboui *et al.*, 2012]. Avec MaD-WiSe, les auteurs ne mettent pas en œuvre les requêtes basées sur l'événement, mais ils prévoient d'utiliser une approche événementielle pour traiter des données de capteurs [Amato *et al.*, 2010].

2.3.7.9 Synchronisation

TinyDB utilise un protocole de synchronisation sur chaque nœud qui définit le début et la fin de chaque période pour les différents nœuds [Madden *et al.*, 2005]. Cougar, ainsi que Sina et Antelope n'appliquent pas la synchronisation entre les capteurs pendant le traitement des résultats [Yao and Gehrke, 2002]. Corona utilise des requêtes de synchronisation (SYN) pour resynchroniser le réseau. Mad-WiSe synchronise l'activité radio du réseau, pour transférer des données de la source vers la destination.

2.3.7.10 Exécution multi-requêtes

SINA, Cougar et Mad-WiSe ne supportent pas l'exécution de plusieurs requêtes en parallèle, contrairement à TinyDB qui la supporte, mais il ne peut pas les ordonnancer avec les différentes propriétés de synchronisation [Chipara *et al.*, 2006]. Il peut équilibrer la charge de travail entre les nœuds, ce qui augmente la vitesse et l'efficacité du traitement des requêtes. Antelope et Corona supportent une exécution multi-requêtes [Li *et al.*, 2008] [Tsiftes and Dunkels, 2011]. Le module de traitement des requêtes peut exécuter l'ensemble des requêtes soumises par les utilisateurs ou les applications.

2.3.7.11 Stockage des données

Cougar utilise le composant *Frontal* comme une base de données pour agréger et stocker les données de capteurs collectées à partir du réseau. TinyDB stocke les données de capteurs dans une table virtuelle nommé "Sensor". Dans cette table, les lignes représentent les nœuds de capteurs et les colonnes représentent les attributs (température, lumière, humidité etc.). Cette table est partagée par l'ensemble des capteurs. Elle est utilisée uniquement en fonction des besoins pour : (i) satisfaire aux requêtes des utilisateurs, ou (ii) pour stocker les données pendant une courte période, ou (iii) pour livrer les données directement à leurs destinations [Madden *et al.*, 2005]. TinyDB permet également d'enregistrer les données de capteurs dans une base de données distante. Les capteurs peuvent stocker seulement les programmes de traitement des données et les informations concernant les nœuds, parce qu'ils ont une capacité de stockage limitée. SINA considère le RCSF comme une collection de fiches techniques et l'ensemble de ces fiches constitue des feuilles de calcul associatives. Le paradigme de la feuille de calcul, utilisée par SINA, consiste à stocker les données dans une cellule référencée par le nom de l'attribut [Chipara *et al.*, 2006]. Antelope utilise un stockage basé sur l'abstraction fourni par le système de fichiers "Coffee" [Tsiftes *et al.*, 2009], qui permet de stocker les données dans la mémoire flash du capteur, afin d'économiser l'énergie dans le réseau. De cette manière, les nœuds de capteur deviennent comme de petites bases de données capables de gérer et de stocker les données. Corona permet de stocker les données du capteur dans la mémoire tampon locale du capteur. Il permet aussi un stockage local des résultats de requêtes dans l'ordinateur de l'utilisateur [Khoury *et al.*, 2010]. Avec MaD-WiSe, les auteurs ont évité le stockage temporaire des données parce que les données sont générées en continu et elles sont traitées à la volée (quand elles arrivent). De plus, ce traitement sert également à répondre aux contraintes de mémoire du système. Les auteurs proposent d'utiliser une mémoire tampon pour stocker les tuples de données entrants, mais ils n'ont pas discuté sa mise en place dans leur système.

2.3.7.12 Méthodes d'indexation

SINA, Cougar, Corona et MaD-WiSe ne proposent pas de méthodes d'indexation. Antelope utilise des méthodes d'indexation nommées MaxHeap Index, Inline Index et Hash Index, afin de réduire la consommation d'énergie et d'améliorer l'utilisation de la mémoire. TinyDB utilise les notions d'indexation pour localiser les nœuds qui ont des données pertinentes pour répondre aux requêtes des utilisateurs. Il utilise la méthode de l'arbre de routage sémantique (SRT), qui permet d'obtenir des informations sur les capacités des nœuds et de déterminer quels sont les nœuds qui peuvent participer à la construction de la réponse pour la requête de l'utilisateur.

2.3.7.13 Coût en énergie

Les bases de données de capteurs ont un objectif principal qui est de réduire le nombre de transmissions entre les nœuds et de préserver l'énergie dans le réseau. Chacun de ces systèmes a mis en place son propre plan d'exécution de requêtes, afin de réduire la consommation d'énergie dans le réseau. Par exemple, Cougar utilise la méthode d'agrégation avec la collecte des données et il utilise aussi un optimiseur de requêtes qui décrit à la fois le flux de données à l'intérieur du réseau et le flux de données calculé au sein de chaque capteur [Yao and Gehrke, 2002].

SINA utilise des méthodes de regroupement, dans lequel les capteurs sont regroupés de manière autonome pour fournir une énergie efficace dans le réseau. TinyDB utilise un arbre de routage sémantique (SRT) et des fonctions d'agrégation pour collecter les données de capteurs. La méthode SRT sélectionne les nœuds qui ont des capacités pour répondre à la requête de l'utilisateur, ce qui permet effectivement de réduire le nombre de nœuds actifs et de réduire la consommation d'énergie dans le réseau. Antelope utilise des méthodes d'indexation et des fonctions d'agrégation pour réduire la consommation d'énergie dans le réseau. Le système Corona propose deux techniques. Tout d'abord, il regroupe les résultats pour fournir plus d'informations sur les données détectées pendant le processus d'agrégation. Par la suite, une deuxième requête collecte tous les résultats des groupes (clusters), ce qui réduit le nombre de résultats transmis sur le réseau. Deuxièmement, Corona adapte un mécanisme de fraîcheur, qui définit le temps nécessaire à l'acquisition de données, basé sur un algorithme heuristique. Pendant ce temps, le capteur est désactivé. Il utilise uniquement les données de la dernière acquisition qui se trouvent dans son cache pour réduire la consommation d'énergie.

MaD-WiSe considère que l'activation de la radio est l'une des principales causes de la consommation d'énergie. Pour cette raison, il synchronise les activités de radio tout au long de la communication multi-saut. Il utilise aussi les méthodes d'agrégation et une méthode d'optimisation des requêtes basées sur des règles de transformation, qui consistent à transformer le plan de requêtes en un plan de requêtes sémantique avec un coût inférieur.

2.3.7.14 Mise en cache des données

La mise en cache des données représente une technique efficace utilisée dans les RCSF pour réduire le nombre de transmission de données. L'accès aux données peut être plus rapide, afin de réduire la consommation d'énergies dans le réseau. Corona utilise la technique de cache de données dans le mécanisme de fraîcheur. Il permet aux nœuds l'utilisation des dernières données acquises, sans activation du processeur de requêtes. TinyDB utilise

	SINA	Cougar	TinyDB	Antelope	MaD-WiSe	Corona
Agrégation des données	Oui	Oui	Oui	Oui	Oui	Oui
Exécution multi-requêtes	Non	Oui	Oui	Oui	Non	Oui
Optimisation de requêtes	Non	Oui	Oui	Oui	Oui	Oui
Requêtes à évènements	Oui	Non	Oui	Non	No	No
Requêtes à durée de vie	Oui	Oui	Oui	Non	Oui	Oui
Synchronisation	Non	Non	Oui	Non	Oui	Oui
Méthodes d'indexation	Non	Non	Oui	Oui	Non	No
Stockage	Non	Non	Oui	Oui	Non	Oui

Tableau 2.3: Fonctionnalités des bases de données de capteurs sans fil

également la mise en cache des données. Il duplique les nœuds parents pour chaque nœud enfant et il met en cache les données de la dernière période d'acquisition sur chaque nœud, pour ne pas perdre les données si les connexions entre les nœuds parents et enfants sont perdues. Cougar, MaD-WiSe, SINA et Antelope n'utilisent pas la technique de mise en cache.

2.4 Conclusion

Nous avons présenté dans ce chapitre un état de l'art sur les BDCSF. Dans un premier temps, nous avons présenté les RCSF, leurs domaines d'applications, et la gestion des données dans les RCSF. Dans un second temps, nous avons présenté les BDCSF, leurs architectures, leurs fonctionnements ainsi que le langage de requêtes utilisé et les différents types de requêtes fournies. Nous avons discuté par la suite les différents systèmes de BDCSF, leurs caractéristiques et leurs faiblesses à travers une comparaison détaillée. Les recherches actuelles menées sur les BDCSF se focalisent sur l'étude des contraintes d'énergie dans ces systèmes. Les BDCSF peuvent rallonger la durée de vie des capteurs et améliorer les communications, ainsi que les traitements des données, qui deviendront au sein du capteur lui-même. Cependant, les contraintes temporelles telles que la validité temporelle des données et les échéances des transactions n'ont pas été assez bien étudiées dans les BDCSF. Les travaux existants s'intéressent aux contraintes temporelles dans le RCSF en général, et ils ne considèrent pas l'architecture des BDCSF. Dans le chapitre 3, nous discuterons en détail les contraintes temporelles dans les BDCSF.

Chapitre 3

Les contraintes temporelles dans les bases de données de capteurs sans fil

Sommaire

3.1 Introduction

Les contraintes temporelles dans les BDCSF constituent un nouveau domaine qui n'a pas été bien exploré, ni dans la littérature des RCSF, ni celles des bases de données temps réel. Comme il a été mentionné dans le chapitre 2, les BDCSF représentent une solution efficace pour la gestion des données de capteurs, et pour minimiser la consommation d'énergie dans le réseau. Les applications de RCSF peuvent intégrer cette nouvelle approche dans leurs mécanismes de fonctionnement. Cependant, elles nécessitent des contraintes temporelles pour refléter de la manière la plus fidèle possible l'état de l'environnement. Les capteurs doivent délivrer leurs mesures avant une échéance donnée, sinon les données délivrées deviennent obsolètes et ne correspondront plus à l'état réel de l'environnement. De même que les opérations de manipulation de la base, qui doivent être contraintes par le temps. Dans ce chapitre, nous discutons les contraintes temporelles dans les BDCSF. Dans la section 1, nous présentons les contraintes temporelles, leurs types d'expression, leurs classifications et enfin, nous discutons les contraintes temporelles dans les systèmes de BDCSF actuelles. Dans la section 2, nous présentons les protocoles temps réel dans les RCSF, ainsi que les différents standards proposés.

3.2 Contraintes temporelles dans les BDCSF

3.2.1 Présentation

La prise en compte des contraintes temporelles dans les BDCSF n'a pas été suffisamment étudiée. Les BDCSF comme TinyDB et Cougar ont utilisé l'approche base de données comme un modèle pour améliorer la collecte de données dans le RCSF. Ils utilisent les techniques de traitement des requêtes, pour minimiser le nombre de transmissions de données, afin de réduire la consommation d'énergie dans le réseau. L'aspect temporel est très important dans la transmission de données, ainsi que pour obtenir une qualité de données supérieure qui reflète l'état courant du système. Les systèmes temps réel tel que les RCSF, sont différents des autres applications, car ils doivent fournir des réponses efficaces dans des délais bien déterminés. On trouve cette définition pour un système temps réel, cité par [Stankovic and Ramamritham, 1989] :

> « A real-time system is defined as a system whose correctness of the system depends not only on the logical results of computations, but also on the time at which the results are produced ».

Dans la littérature des systèmes temps réel, on trouve trois types de contraintes temporelles (critiques, strictes et souples), qui dépendent de la nature du système temps réel. Nous avons choisi cette classification pour les contraintes temporelles dans les RCSF.

- **Contraintes temporelles strictes critiques (hard)** : le non-respect de ces contraintes peut engendrer des dommages au système, des catastrophes humaines ou industrielles. Par exemple, dans une centrale nucléaire de production énergétique, des capteurs sans fil sont déployés dans la salle des générateurs avec un système de climatisation responsable du contrôle de la température dans la salle. Si les capteurs de température ne respectent pas les délais fixés par le système et qu'ils fournissent leurs mesures trop tard à la station de base, le niveau de température va augmenter, ce qui peut engendrer une catastrophe.

- **Contraintes temporelles strictes non critiques (firm)** : le non-respect de quelques échéances peut être tolérée, mais il n'aura pas de conséquences graves. La tâche qui rate son échéance sera abandonnée. Par exemple, un système de surveillance par caméras prend des photos pour surveiller les mouvements des clients dans un magasin. Il les envoie à une station de base où les images sont traitées par un ordinateur. Lorsque l'ordinateur est surchargé, une nouvelle image peut être reçue avant le traitement d'une ancienne image. Dans ce cas, l'image la plus ancienne sera abandonnée et l'image récemment reçue sera traitée à sa place.

- **Contraintes temporelles "souples" ou non strictes (soft)** : le non-respect des échéances est acceptable. La transaction qui rate son échéance continue son exécution, en fournissant une QdS moindre. Par exemple, dans un processus de contrôle des pièces dans une chaîne de production, un retard dans la détection de pièces défectueuses peut être détecté dans un autre niveau de contrôle. Le niveau de production va baisser à cause des retards, mais il ne causera pas de dommage sur le système.

Le tableau 3.1 présente les caractéristiques temporelles des données pour des applications temps réel dans les domaines de l'avionique, de la surveillance du trafic aérien, du contrôle industriel et des systèmes de contrôle/commande. Ces caractéristiques se résument dans la cohérence externe qui décrit le niveau acceptable d'erreur sur l'information par rapport à l'environnement extérieur [Sadeg, 2004]. La cohérence temporelle définit le temps maximal acceptable entre les lectures pour que les données des capteurs représentent fidèlement l'environnement extérieur. La durabilité représente la durée pendant laquelle le système peut fonctionner sans redémarrage [Locke, 2001].

Application	Cohérence externe	Cohérence temporelle	Durabilité
Avionique	0.05 sec.	0.20 sec.	4 heures
Surveillance du trafic aérien	1.5 sec.	3 sec.	12 heures
Contrôle industriel	0.20 sec.	0.50 sec.	5 jours
Contrôle/commande	0.05 sec.	0.10 sec.	1 heure

Tableau 3.1: Caractéristiques des données dans un système temps réel [Locke, 2001]

3.2.2 Types d'expression

Les contraintes temporelles peuvent être exprimées sous la forme d'un temps absolu ou relatif. Elles peuvent être aussi exprimées à l'aide du temps logique, lié à des actions ou des événements [Mammeri, 1998]. Par exemple, l'événement E1 peut être exécuté avant ou après l'événement E2. Allen [Allen, 1983] a défini une logique temporelle à l'aide des opérateurs qui permettent d'exprimer les relations entre les intervalles de temps en utilisant 13 relations (cf. Tableau 3.2).

Types d'expression	Forme	Exemples
Temps absolu	Une date	Jour/Mois/Années hh:mm:ss.
	Un horodatage (timestamp)	26/12/2015 à 13:04:02 en timestamp correspond à 1451131442.
Temps relatif	Un intervalle du temps	Avec une date de début et une date de fin.
	Une durée	Une valeur temporelle qui correspond à la durée d'exécution d'une tâche. qui peut être bornée (minimum et maximum) pour garantir des résultats avant les échéances.
Temps logique	Un algèbre de Allen	Precedes, meets, overlaps, finished by, contains, starts, equals, started by, during, finishes, overlapped by, met by, preceded by.

Tableau 3.2: Types d'expression des contraintes temporelles

3.2.3 Classification

Les contraintes temporelles dans les BDCSF sont liées aux données et aux traitements.

3.2.3.1 Contraintes temporelles liées aux données

Les RCSF produisent une grande quantité de données en temps réel, sous forme de flux, difficiles à gérer par le capteur lui-même, à cause de ses capacités limitées. Les données qui sont collectées doivent être cohérentes, c'est à dire, elles doivent refléter l'état courant de l'environnement. Elles deviennent inutiles une fois qu'elles dépassent leurs intervalles de validité. La validité temporelle et la fraîcheur de données représentent deux paramètres importants pour garantir une meilleure qualité de service dans le RCSF. La validité temporelle des données correspond à l'intervalle de temps qui sépare la date de production de la donnée et la date de son utilisation. On considère que les données sont valides si elles n'ont pas dépassé l'intervalle de validité associé. On considère que les données sont fraîches si elles reflètent l'état courant de l'environnement. Par exemple, dans un système de surveillance environnementale, des capteurs de température envoient périodiquement la moyenne des températures d'une zone surveillée à la station de base. Les données peuvent êtres valides, mais non précises à cause de défaillances de l'un des capteurs. Donc, les données collectées ne reflètent pas fidèlement l'état actuel de la zone surveillée.

Dans les RCSF, les données peuvent avoir un intervalle de validité temporelle absolu (avi) (Absolute Validity Interval) ou un intervalle de validité temporelle relatif (rvi) (Relative Validity Interval). L'intervalle de validité absolu désigne intervalle de temps durant lequel les données doivent refléter fidèlement l'état actuel de l'environnement. L'intervalle de validité relatif désigne l'ensemble des données qui doivent être valides pendant le même intervalle de temps. Ramamritham [Ramamritham, 1993] a défini un modèle pour les donnée temps réel : d = (d_{valeur}, $d_{estampille}$, d_{avi}), où d_{valeur} représente la valeur de la donnée, $d_{estampille}$ représente la date de mesure de donnée et d_{avi} représente l'intervalle de validité absolu de la donnée. Soit R l'ensemble des données utilisées pour dériver une nouvelle donnée, et d, d' ∈ R. On considère que d est temporellement cohérente si :

- cohérence absolue : $instant_{courant} - d_{estampille} \leq d_{avi}$
- cohérence relative : $|d_{estampille} - d'_{estampille}| \leq d_{rvi}$

Les capteurs peuvent avoir des contraintes sur l'intervalle de production des données. Dans une collecte périodique, les capteurs doivent générer des données selon un intervalle de production fixé par l'utilisateur. Cet intervalle doit être respecté par le capteur, entre deux productions de données successives.

3.2.3.2 Contraintes temporelles liées aux actions

Dans les RCSF, il y a deux catégories de traitements. La première concerne l'ensemble des capteurs dans le réseau, pour lesquels, les contraintes temporelles peuvent être ex-

primées soit à l'aide du temps quantifié (temps absolu ou temps relatif) ou en utilisant la logique temporelle d'Allen, qui définit les relations entre les intervalles de temps [Allen, 1983]. La deuxième catégorie de traitements est spécifique au capteur lui-même, dont les contraintes temporelles peuvent être exprimées à l'aide du temps quantifié. Nous avons choisi de discuter les contraintes temporelles dans la première catégorie selon trois classes de traitements (périodique, sporadique et apériodique) :

1. **Contraintes temporelles liées à des tâches périodiques** : une tâche périodique est une tâche qui se répète après un certain intervalle de temps fixe, appelé période de la tâche. La collecte des données dans les RCSF est une tâche périodique. Les capteurs envoient leurs mesures périodiquement à la station de base, selon un intervalle de temps fixé par l'utilisateur ou le système, appelé (EPOCH). Une tâche périodique peut avoir des contraintes temporelles sur la période de la tâche : une date de démarrage au plus tard sur la période de la tâche, une durée maximale d'exécution de la tâche, une durée de validité relative de la tâche.

2. **Contraintes temporelles liées à des tâches sporadiques** : une tâche sporadique est une tâche basée sur un instant aléatoire. L'utilisateur peut interroger le RCSF d'une manière sporadique. Il peut envoyer des requêtes de type SQL à travers la station de base pour récupérer des informations. Rajib Mall [Mall, 2009] a présenté une tâche sporadique avec trois tuples, sous la forme suivante : Ti = (ei, gi, di), avec *ei* qui représente le pire cas de temps d'exécution, *gi* qui désigne le temps minimal qui sépare deux instances consécutives d'une tâche, et *di* qui représente l'échéance relative de la tâche.

3. **Contraintes temporelles liées à des tâches apériodiques** : une tâche apériodique est similaire à une tâche sporadique. Elle est basée sur un instant aléatoire. Par contre, l'intervalle de temps minimal qui sépare deux tâches consécutives peut être nul. Deux ou plusieurs tâches apériodiques peuvent se produire au même instant. En outre, les échéances des tâches apériodiques sont exprimées, soit par des valeurs moyennes, soit statistiquement. Les tâches apériodiques sont généralement des tâches temps réel souple. On peut trouver ce type des tâches dans les RCSF, lorsque plusieurs utilisateurs envoient leurs demandes en même temps à la station de base.

Nous avons choisi de discuter les contraintes temporelles pour les trois tâches suivantes :

- **Acquisition de données** : cette étape consiste à détecter des paramètres de l'environnement telles que la température, la lumière, l'humidité, etc., de les convertir en un signal électrique mesurable à l'aide d'un convertisseur analogique-numérique, puis de les transmette à l'unité de traitement des données. Ce traitement dépend à la fois de la date au plus tôt ou au plus tard de démarrage de l'acquisition des

données, ainsi que de la qualité du détecteur et du temps mis pour la conversion du signal.

- **Traitement de données** : cette étape consiste à gérer les données provenant des autres capteurs, à traiter les requêtes "utilisateur" et à construire la réponse, à exécuter les programmes et à les basculer entre différents modes de fonctionnement (actif, inactif, sommeil) pour économiser l'énergie du capteur. Le temps de traitement dépend de la capacité de calcul du capteur lui-même ainsi que de la date d'arrivée des données des autres capteurs.

- **Transmission de données** : cette étape consiste à envoyer les informations collectées ou les réponses aux requêtes "utilisateur" à la station de base. Le temps de transmission des données dépend du protocole utilisé dans la couche MAC, du protocole de routage, de la distance entre la source et la destination et de la quantité de données transmises.

3.2.4 Discussion

Les BDCSF utilisent des clauses SQL pour définir la durée de vie de la requête, telles que les clauses `<DURATION>` ou`<EPOCH>` pour spécifier la durée entre deux exécutions de requête, ou déterminer la période pendant laquelle les données sont agrégées. Cependant, les BDCSF ne définissent pas un modèle de données temporelles pour gérer les données de capteurs.

Le système Cougar utilise la notion de durée de vie de la requête à travers les clauses SQL `<EVERY>` et `<DURATION>`. La clause `<DURATION>` spécifie la durée de vie de la requête et la clause `<EVERY>` définit la durée du cycle d'exécution de la requête (cf. Requête 9). L'idée de base de ce projet est de réduire la communication entre les nœuds, en effectuent des calculs locaux (au niveau de nœuds) sur les données détectées, cependant, Cougar ne gère pas les contraintes temporelles.
Dans la Requête 9, l'utilisateur cherche la température moyenne qui dépasse 35 degrés dans la région 2. La durée de vie de la requête est de 20 heures à partir du moment où la requête est soumise. La détection se fait par période de 20 minutes.

Requête 8.

```
SELECT AVG(température),
FROM sensors
WHERE R.loc IN region 2
HAVING AVG(température) > 35
DURATION now + 1200
EVERY 20
```

Le système TinyDB utilise également la notion de durée de vie des requêtes pour gérer la consommation d'énergie dans le réseau. L'utilisateur peut spécifier la durée de vie de la requête à travers la clause `LIFETIME <x>`, où `<x>` représente la durée en jours, semaines ou mois (cf. Requête 10).

Requête 9.

```
SELECT nodeid, temp
FROM sensors
LIFETIME 10 days
```

TinyDB effectue une estimation pour satisfaire la clause de durée de vie dans la requête. Cette estimation consiste à calculer la taille de l'échantillon ainsi que le taux de transmission compte tenu de la quantité d'énergie restante au niveau du capteur. Le nœud racine commence par calculer cette estimation, par la suite, il se coordonne avec les autres nœuds dans le réseau pour calculer les taux de livraison de données. Cette estimation est calculée sur la base du coût d'accès aux données du capteur, de la sélectivité de l'opérateur (c.-à-d. le nombre moyen de n-uplets produits par l'opérateur pour chaque n-uplet reçu en entrée de l'opération) [Billet, 2015], du débit prévu et enfin de la tension de la batterie.

TinyDB utilise aussi la notion d'époque (epoch), qui définit la période de temps entre chaque intervalle d'échantillonnage. Elle fournit un mécanisme pratique pour la structuration du calcul afin de minimiser la consommation d'énergie dans le réseau.

Requête 10.

```
SELECT nodeid, light, temp
FROM sensors
SAMPLE INTERVAL 1s FOR 10s
```

Dans la Requête 11, chaque capteur doit envoyer son identifiant, les mesures de lumière et de température une fois par seconde pendant 10 secondes. Bien que TinyDB ait intro-

duit quelques notions temporelles à travers la durée de vie d'une requête ou la période d'échantillonnage, ces fonctions restent concentrer sur la minimisation de la consommation d'énergie dans le réseau. Les contraintes temporelles n'ont pas été discutées avec le système TinyDB.

Corona [Khoury *et al.*, 2010] introduit la notion de *fraîcheur* dans la collecte des données du RCSF. Ce système utilise des requêtes d'agrégation pour réduire les délais de traitement et les communications coûteuses dans le RCSF. La requête 12 est un exemple de requête Corona. La contrainte de fraîcheur, définie par la clause `<FRESHNESS>`, spécifie combien de temps elle est autorisée à passer depuis la dernière acquisition du capteur afin que le moteur de recherche n'active pas à nouveau le capteur, mais il utilise plutôt les lectures précédentes de la mémoire cache du capteur.

Requête 11.

```
SELECT température
FROM sensors
WHERE light > 100
EPOCH 60s RUNCOUNT 60
FRESHNESS 10s
```

Cette décision est prise dynamiquement à chaque activation de la requête. De plus, l'optimiseur de requêtes prend en considération la contrainte de fraîcheur pour déterminer le décalage temporel optimal avant chaque nouvelle requête, afin que les résultats dans le cache soient optimisés. Ce mécanisme réduit le nombre de transmissions de requêtes dans le réseau de capteurs, et offre des données valides à l'utilisateur. Corona permet aux requêtes d'avoir une date de démarrage spécifique représentée par les clauses `START IN <relative time>` ou `START AT <absolue time>` ou `START ON DATE <date>`, et aussi de définir le temps entre deux exécutions de la requête représentée par la clause `<EPOCH>`, et la clause `<RUNCOUNT>`, qui définit le nombre d'exécutions de la requête.

3.3 Protocoles temps réel dans les RCSF

3.3.1 Protocole MAC temps réel

TDMA (Time division multiple access) [Macedo *et al.*, 2009] est un protocole de contrôle d'accès au médium basé sur le principe de la répartition du temps en intervalles (multiplexage temporel). Chaque nœud est affecté à un slot de temps connu par avance, durant lequel il peut utiliser le canal de transmission pour envoyer ses données. Le protocole

TDMA assure aux capteurs un accès au medium sans collision. La collision provoque la retransmission des données, ce qui augmente la latence et la consommation d'énergie. TDMA peut fournir des latences de paquets prévisibles et favoriser des taux de données élevés, car il évite le temps d'écoute des nœuds [Chipara *et al.*, 2007]. Chaque nœud connaît par avance le slot de temps qu'il va occuper pour envoyer ou recevoir des données. Une fois qu'il a terminé, il rentre en mode sommeil, ce qui lui permet d'économiser l'énergie de la batterie.

I-EDF [Caccamo *et al.*, 2002] est un protocole MAC synchrone, utilisant l'algorithme EDF (Earliest Deadline First), pour gérer l'accès au canal de transmission. Les nœuds sont synchronisés et sont regroupés sous forme des cellules hexagonales. Chaque cellule forme un "cluster" qui contient un routeur et un ensemble de nœuds. Les routeurs assurent les transmissions des paquets entre les différents nœuds dans le cluster, pour qu'ils connaissent leurs échéances à tout moment. Pour cette raison, chaque nœud envoie aux autres nœuds du cluster, à travers le routeur, les paramètres du trafic (la période, l'échéance relative, la taille du message). Ces informations lui permettent de construire la séquence d'ordonnancement, basé sur l'algorithme d'ordonnancement EDF. Le nœud ayant le paquet de données avec la plus courte échéance sera prioritaire pour envoyer les données. Le routeur peut envoyer et recevoir des paquets d'une façon cyclique, dans six directions (les 6 côtés de la cellule) pour communiquer avec tous les nœuds du cluster. Il peut utiliser différentes fréquences pour transmettre les données à ses voisins.

Dual-Mode MAC [Watteyne *et al.*, 2006] est un protocole MAC temps réel strict conçu pour l'application avec des réseaux linéaires telles que les applications de surveillance d'accidents sur les autoroutes, les applications de surveillance de voies ferrées et de pipelines. Ce protocole peut basculer entre deux modes de fonctionnement. Un mode protégé et un mode non protégé. Le mode non-protégé est utilisé lorsqu'il n'y a pas de collisions. Les paquets de données sont transmis avec une vitesse quasi-optimale mais les délais de transmission ne sont pas garantis. Chaque nœud envoie son message jusqu'au nœud puits tant qu'il n'y a pas d'autres messages sur le canal de transmission, et lorsque son temps d'attente (Backoff time), qui est proportionnel à sa distance au nœud puits, est écoulé. Le mode protégé est utilisé pour éviter les collisions. Les paquets de données sont transmis moins rapidement mais les délais de transmission ne sont garantis grâce au mécanisme de réservation du canal de transmission. Une fois que le nombre de paquets de données diminue, le protocole revient en mode non protégé, car la transmission des données est plus rapide. Ce protocole ne nécessite pas de synchronisation, cependant il n'est pas efficace en terme de consommation d'énergie et ne prend pas en compte le mécanisme de tolérance aux fautes.

RTXP [Mouradian *et al.*, 2014] est un protocole de communication temps réel cross-layer (MAC et routage), basé sur la localisation. Il garantit les contraintes sur le délai

de transmission de bout en bout, en limitant la durée de transmission d'un saut et de l'ensemble des sauts pour atteindre le nœud puits. Il utilise un système de coordonnées virtuelles qui classe les nœuds selon le nombre de sauts depuis toute source vers le puits. Ce système permet à chaque nœud de connaitre sa distance (nombre de sauts) jusqu'au nœud puits. Il permet de discriminer les nœuds dans un 2-voisinage pour éviter les collisions, ainsi que pour distinguer les nœuds ayant le même nombre de sauts, afin d'améliorer la sélection des nœuds expéditeurs. Il assigne un identificateur unique à des nœuds dans un domaine d'interférence afin d'accéder de manière déterministe au médium. Il implémente aussi un routage de type opportuniste pour organiser les accès au canal (en temps borné) et router de manière déterministe.

3.3.2 Protocoles de routage temps réel

RAP [Lu *et al.*, 2002] est une architecture de communication temps réel pour les RCSF largement déployé. Il fournit une API permettant aux applications de soumettre des requêtes à travers le RCSF. Il permet de spécifier les événements, les contraintes temporelles et la zone de transmission de la requête. Les résultats des requêtes sont envoyés par la suite vers la station de base. RAP utilise une méthode de transmission basée sur l'emplacement géographique des nœuds. Chaque nœud utilise les informations de son voisin immédiat pour transmettre le paquet de données. Cette méthode est plus efficace avec des réseaux à forte densité. RAP utilise un mécanisme d'ordonnancement de paquets, appelé « velocity monotonic », qui attribue des priorités aux paquets sur la base de la vitesse demandée (vélocité). Le paquet qui nécessite une vitesse élevée aura une priorité élevée. La vélocité représente le rapport entre la distance parcourue par le paquet de données et son échéance. Cette technique améliore le nombre de paquets respectant leurs échéances, aussi, elle permet aux paquets situés loin de la station de base d'avoir des priorités plus élevées, pour concurrencer les paquets qui sont proches de la destination.

SPEED [He *et al.*, 2003] est un protocole de routage géographique qui supporte les communications temps réel souples dans les RCSF. Ce protocole prend en compte les contraintes temps réel dans la transmission des données. Il choisit les chemins vers la destination dont les nœuds offrent une vitesse de propagation ou une vélocité suffisante pour respecter l'échéance du paquet de données à envoyer. La vélocité d'un nœud est calculée en divisant la distance parcourue par un paquet par son délai de bout en bout.

Chaque nœud envoie périodiquement un paquet à ses voisins, pour qu'ils échangent leurs informations locales. Il maintient une table contenant les informations sur ses voisins (l'identifiant du voisin, sa position, le délai de transmission, l'échéance). Lorsqu'un nœud veut envoyer un paquet à une destination, il sélectionne l'ensemble de ses voisins les plus proches de cette destination. S'il ne trouve pas des voisins proches de la destination,

le paquet sera rejeté. Sinon, il sélectionne, parmi cet ensemble, les nœuds qui ont une vitesse d'envoi supérieure à la vitesse de référence avec laquelle un paquet doit atteindre sa destination en respectant l'échéance.

$$Vitesse(i,j) = \frac{Distance(i,j)}{Delai(i,j)} \tag{3.1}$$

Dans l'équation 3.1, la vitesse de propagation d'un paquet entre un nœud (i) et son voisin (j) est égale à la fraction de la distance ces deux nœuds sur le délai pour atteindre ce voisin. Le voisin qui a la plus grande vitesse aura une grande probabilité d'être choisi comme candidat pour la transmission du paquet. S'il ne trouve pas le candidat, SPEED emploie un mécanisme d'adaptation du trafic pour réduire la valeur de la vitesse de référence. Il utilise aussi un autre mécanisme qui permet de détecter et de réduire la congestion dans le réseau afin de maintenir la vitesse de référence. SPEED contrôle le trafic dans le réseau : lorsqu'un nœud reçoit un trafic important en provenance de ses voisins, il supprime un pourcentage de paquets émis, sinon, il détourne le trafic vers d'autres nœuds afin d'équilibrer la charge dans le réseau.

MMSPEED (Multi-path Multi-speed Routing Protocol) [Felemban *et al.*, 2006] représente un mécanisme dédié pour la livraison de paquets de données avec une meilleure qualité de service dans les RCSF. Les auteurs se concentrent sur deux aspects de QdS qui sont : le respect des délais de transmission des données et la fiabilité des communications. MMSPEED utilise aussi plusieurs chemins pour envoyer le paquet de données afin d'assurer la fiabilité de la communication dans le réseau. Cette technique permet d'augmenter la probabilité que le paquet atteigne sa destination, et permet d'équilibrer la charge dans le réseau. Le nombre de chemins dépend de la fiabilité du paquet de données. Un paquet de données qui obtiendra un grand degré de fiabilité sera émis a travers plusieurs chemins. De même, un paquet de faible fiabilité sera transmis en utilisant un seul chemin. MMSPEED est basé sur le protocole de routage SPEED [He *et al.*, 2003], qui garantit une seule vitesse d'envoi du paquet de données dans le réseau. Cependant, MMSPEED fournit différentes vitesses d'envoi pour assurer les livraisons de données avec plusieurs échéances. Il utilise aussi la méthode de compensation dynamique de positions. Chaque nœud peut déterminer localement les nœuds intermédiaires, qui peuvent assurer la livraison des données. Les nœuds intermédiaires peuvent répondre avec la probabilité que le paquet atteigne sa destination, sur la base des estimations d'erreurs locales et des distances géographiques (nombre de sauts) aux voisins immédiats. Ils peuvent aussi augmenter la vitesse de transmission d'un paquet lorsque la vitesse de référence est élevée pour qu'il atteigne sa destination. MMSPEED se base sur le mécanisme de gestion des priorités du standard 802.11e au niveau de sa couche MAC. Chaque vitesse se réfère à une classe de priorités de la couche MAC : une vitesse élevée correspond à une priorité élevée. De cette

manière, les paquets à grande vitesse ne sont pas susceptibles d'être bloqués par d'autres paquets à faible vitesse.

RPAR (Real-time Power-Aware Routing) [Chipara *et al.*, 2006] est un protocole de routage temps réel. Il tient compte de l'énergie et du délai de transmission de bout en bout entre la source et la destination. Lorsqu'un nœud veut envoyer un paquet de données à une destination, il utilise une méthode de découverte des voisins, qui lui permet de trouver des nouveaux voisins qui offrent une vélocité supérieure à la vitesse de référence (la vitesse requise par le paquet). Ensuite, il sélectionne parmi cet ensemble le nœuds qui consomment moins d'énergie, qui sera le prochain saut vers la destination. Le cas échéant, RPAR utilise une méthode d'adaptation de puissance de transmission, pour augmenter la puissance de transmission de l'un de ses voisins afin de répondre à la vitesse requise. Lors de l'envoi des données, RPAR privilégie également les paquets en fonction de leurs échéances.

THVR (Two-Hop Velocity based routing protocol) [Li *et al.*, 2009] est un protocole de routage géographique qui supporte les communications temps réel dans les RCSF. Il utilise les informations de ses voisins à deux sauts pour réduire les délais de transmission des données, en considérant l'efficacité énergétique dans le RCSF. Chaque nœud échange régulièrement, avec ses voisins à un saut, leurs informations (ID, position, énergie, etc.). Quand un nœud veut envoyer un paquet de données à une destination, il utilise la méthode d'acheminement basée sur la métrique "vélocité" comme dans le protocole de routage SPEED. Il cherche parmi ses voisins à deux sauts ceux qui ont une vélocité supérieure à la vitesse de référence. Par la suite, le module d'estimation calcule le délai de transmission de la source jusqu'à la destination, en utilisant la méthode Window Mean Exponential Weighted Moving Average (WMEWMA) [Woo *et al.*, 2003]. Le délai de transmission contribue au calcul de la vitesse de livraison des paquets tout en respectant l'échéance. Les auteurs utilisent aussi un mécanisme qui s'appelle Initiative Drop Control, pour assurer l'efficacité énergétique et pour garantir les délais de transmission. Ce mécanisme consiste à abandonner les paquets de données pour toute transmission à deux sauts. Ces transmissions ne vont pas atteindre la vitesse demandée. Ainsi, les paquets qui sont près de la source et qui n'arrivent pas à atteindre la vitesse demandée seront abandonnés les premiers. Ce mécanisme permet de diminuer le nombre de transmissions, réduisant ainsi la consommation d'énergie et améliorer les délais de transmission.

PATH (Real-Time Power Aware Two-Hop) [Rezayat *et al.*, 2009] est un protocole de routage temps réel pour les RCSF, basé sur le même principe que le protocole THVR [Li *et al.*, 2009]. Il utilise les informations de voisinage à deux sauts et il implémente un mécanisme efficace de contrôle d'énergie qui prend en considération la consommation d'énergie, la capacité du réseau et les délais de transmission des données. Lorsqu'un nœud veut envoyer un paquet de données, il fait appel au mécanisme d'acheminement qui lui

permet de sélectionner le chemin efficace en énergie et qui répond aux exigences de vitesse de transmission au niveau de chaque saut, afin de garantir les échéances de livraison. Il utilise aussi un mécanisme d'estimation du délai de transmission pour mettre à jour les informations utilisées par le processus d'acheminement de paquets de données. Lorsqu'il ne trouve pas de voisins à deux sauts qui répondent à la vitesse de transmission demandée, il utilise un mécanisme de découverte de voisinage pour trouver des voisins capables d'assurer ces exigences de vitesse de transmission. PATH utilise le même mécanisme de contrôle de paquets. Il abandonne tous les paquets qui n'arrivent pas répondre à la vitesse de transmission demandée. Il abandonne aussi la transmission des paquets lorsque le mécanisme de contrôle d'énergie ne trouve pas un nœud relais pour livrer les données.

RTLD (Real-Time routing protocol with Load Distribution) [Ahmed and Fisal, 2008] est un protocole de routage temps réel avec distribution des charges. Il offre un meilleur rendement pour la livraison des paquets de données, réduit la redondance des paquets et prolonge la durée de vie du réseau. RTLD propose un nouveau type de communication dans les RCSF basé sur une diffusion géo-directionnelle (geodirection-cast) pour livrer les paquets de données. Cette méthode combine une diffusion géographique (géocast) avec une expédition directionnelle en utilisant des trajets multiples afin d'augmenter le taux de livraison des données. Les auteurs distribuent la charge de livraison des données sur des nœuds qui peuvent assurer une transmission optimale. Ces nœuds sont sélectionnés sur la base du taux de réception des paquets, la consommation d'énergie du nœud et la vitesse des paquets sur un saut. RTLD utilise un choix sélectif basé sur : (1) la vitesse de transmission des nœuds pour garantir une livraison des données en temps réel, (2) la qualité des liens pour améliorer le débit des données, (3) et le niveau d'énergie restant au niveau du nœud pour augmenter la durée de vie du réseau. Les résultats des simulations de RTLD ont montré que ce protocole donne de bonnes performances en termes de taux de livraison de paquets en temps réel, de contrôle des paquets et de consommation d'énergie [Ahmed and Fisal, 2008].

3.3.3 Standards industriels

IEEE 802.15.4 est un standard de communication adapté à des communications sans fil personnelles avec une courte distance de transmission, un faible débit et une consommation énergétique limitée. Il est caractérisé par un taux de transfert maximum de 250 Kbit/s et d'une portée de transmission radio de quelques dizaines de mètres. Il est utilisé souvent par d'autres standards de communication telle que ZigBee, 6LoWPAN et WirelessHART. Il spécifie la couche physique et la couche MAC. La couche physique contient l'émetteur-récepteur radio. Elle gère les transmissions des données, la consommation d'énergie ainsi que la sélection des canaux d'envoi. La couche MAC gère les accès au canal de transmis-

sion. Elle utilise la méthode d'accès CSMA/CA .

ZigBee/ZigBeePRO [Val *et al.*, 2008] est un protocole de communication conçu au sein de la ZigBee Alliance pour contrôler les environnements composés d'appareils sans fil avec des contraintes énergétiques et une faible capacité de calcul tels que les RCSF. Il s'adapte bien à ce type d'environnement car il a une faible consommation énergétique qui permet d'augmenter la durée de vie des batteries. ZigBee est caractérisé par un faible débit (250 Kbit/s) et des transmissions radio qui peuvent atteindre 100 mètres. Il utilise la couche physique et la couche MAC qui sont définies par la norme IEEE 802.15.4, mais il propose sa propre spécification pour la couche réseau et la couche application. La couche réseau définit trois types de nœuds dans le réseau.

1. Le coordinateur ZigBee (ZC) : il est unique dans le réseau et représente le nœud central. Il s'occupe de la création du réseau et du routage des communications.

2. Le routeur ZigBee (ZR) : il représente un nœud intermédiaire qui participe au routage des données dans le réseau. Il peut être associé au coordinateur ou à un autre routeur.

3. Le terminal ZigBee ou ZigBee End Device (ZED) : il représente un nœud simple associé au coordinateur ou aux routeurs. Il ne participe pas au routage des messages, mais il peut détecter des données et envoyer et recevoir des messages.

Le coordinateur utilise la notion de supertrame, qui représente l'espace temporel entre deux trames [Val *et al.*, 2008], pour contrôler l'accès au canal. Une supertrame est composée de deux périodes : une période active et une période inactive. La période active est divisée en 16 slots de temps de même durée. SO (Superframe Order) définit la longueur de la période active et BO (Beacon Order) détermine la longueur de la période de trame (beacon). Un nœud du réseau se réveille puis se met à l'écoute du canal avant l'arrivée de la supertrame (juste avant le slot 0 de la supertrame). S'il n'y a pas de données à envoyer ou à recevoir, le nœud entre dans une période inactive. Cette méthode permet d'économiser l'énergie du capteur et d'augmenter la durée de vie du réseau. Le routage des messages dans le réseau est basé sur le principe de fonctionnement du protocole AODV[1] [Perkins *et al.*, 2003]. Cependant, il ne considère pas les contraintes temps réel dans le réseau.

WirelessHART [HART Communication Foundation, 2014] est un standard industriel pour la communication sans fil, développé par la "HART Communication Foundation". Il représente la version sans fil du protocole de communication (Hart), utilisé en contrôle industriel. WirelessHART est conçu pour satisfaire les exigences industrielles telles que la sécurité, la fiabilité et la facilité d'utilisation. Il s'intéresse aux exigences industrielles en temps réel telles que la synchronisation des messages et la garantie de livraison des données tout en respectant les délais. Par contre, il consomme plus d'énergie par rapport

[1] Ad-Hoc On Demand Distance Vector

aux autres standards de communications sans fil.

WirelessHART est basé sur une topologie dans laquelle les appareils sans fil forment un réseau maillé. Chaque appareil peut être considéré comme une source de données ou nœud relais. La transmission d'un message commence au niveau du nœud source, qui envoie le message à son voisin le plus proche, qui sera transmis ensuite d'un nœud à l'autre jusqu'à ce qu'il atteigne la station de base. WirelessHART est un réseau auto-répare. Il propose des chemins alternatifs pour contourner les obstacles et réparer les liaisons en panne à cause des nœuds défectueux. WirelessHART utilise la norme 802.15.4 et le protocole Hart. Il utilise une bande de fréquence sans licence (2.4 GHz). Il implémente la technique de contrôle d'accès TDMA (Time Division Multiple Access) qui divise la bande passante en slots de temps de 10 ms pour synchroniser les messages et les participants du réseau. Il utilise également la modulation FHSS ("Frequency Hopping Spread Spectrum Modulation") pour éviter de sauter d'un canal à l'autre afin d'éviter les interférences. WirelessHART utilise en parallèle les 15 canaux de la norme IEEE802.15.4. Il désactive tous les canaux qui sont fréquemment utilisés pour éviter toute collision avec d'autres systèmes de communication sans fil. La combinaison de la synchronisation avec un slot de temps de 10 secondes et 15 canaux permet 1500 communications par seconde.

6LoWPAN : (IPv6 over Low power Wireless Personal Area network) [Kushalnagar *et al.*, 2007] est une norme définie par le groupe de travail de l'IETF. Elle permet d'envoyer et de recevoir des paquets IPv6 (Internet Protocol version 6) [Deering and Hinden, 1998] via le protocole de communication IEEE 802.15.4 avec des RCSF à faible puissance. 6LoWPAN permet de réduire la taille d'un paquet IPv6 pour l'adapter à la taille d'une trame 802.15.4 (127 octets). Il utilise (1) un mécanisme de fragmentation qui permet de fragmenter un paquet IPv6 en plusieurs trames 802.15.4, (2) un mécanisme de compression des entêtes IPv6 pour LowPAN sur les adresses de liens locaux. 6LoWPAN utilise deux types de routage « Mesh-under » et « Route-over ». Le routage *Mesh-under* est implémenté au niveau de la couche adaptation où se fait la décision de routage. Par conséquent, le paquet IPv6 n'est reconstitué que sur les dispositifs destinataires. Le routage *Route-over* est implémenté au niveau de la couche réseau et le paquet IPv6 est reconstitué au niveau de chaque dispositif pour prendre une décision de routage. 6LoWPAN utilise deux protocoles de routage : RPL[2] [Winter *et al.*, 2012] basé un routage "Route-over" et LOADng[3] [Vucinic *et al.*, 2013]. Cependant, 6LoWPAN n'est pas adapté aux communications en temps réel. 6LoWPAN utilise aussi le mécanisme de découverte des voisins, qui permet une auto-configuration d'une adresse IPv6 sur les dispositifs sans fil. Il prend en considération l'évolutivité et la mobilité des RCSF.

RoLL (Routing Over Low power and Lossy networks) est le groupe de travail de l'IETF

[2]IPV6 Routing Protocol for Low-Power and Lossy Networks
[3]Lightweight On-demand Ad hoc distance-vector routing protocol – next generation

chargé de la mise en place d'un protocole de routage pour les réseaux de faible puissance et avec perte « LLN » (Low Power and Lossy Network). RoLL propose le protocole de routage RPL [Winter *et al.*, 2012], basé sur la topologie de l'arbre ou graphes acycliques orientés appelé DODAG (Destination Oriented Directed Acyclic Graph) et une fonction objective qui combine différentes métriques (consommation d'énergie, nombre de sauts minimum, nombre de transmissions prévues) et des contraintes, afin de trouver le meilleur chemin de la racine de l'arbre (station de base) vers les capteurs, et vice-versa. Il utilise des messages de contrôle et de construction du DODAG : (1) DIO (DODAG Information Object) est le message d'informations sur le DODAG. Il permet à un nœud de connaître les paramètres de configuration, de sélection de son parent et de calculer son rang (2) DIS (DODAG Information Solicitation) est utilisé pour solliciter des informations sur le DODAG et (3) DAO (Destination Advertisement Object) est le message d'objet d'annonce de destination, qui permet de propager les informations de destination vers la racine de DODAG.

3.4 Conclusion

Nous avons présenté dans ce chapitre les contraintes temporelles dans les BDCSF et les différents protocoles et standards temps réel dans les RCSF. Nous avons conclu que les BDCF ne tiennent pas compte des contraintes temporelles dans les RCSF. Elles implémentent seulement quelques clauses SQL temporelles pour contrôler et réduire la consommation d'énergie dans le RCSF. Le modèle de BDCSF n'a pas été utilisé pour assurer les contraintes temporelles dans les RCSF. D'un autre côté, les systèmes de gestion de bases de données temps réel (SGBDTR) ont montré leurs performances dans la gestion des contraintes temporelles des transactions. De plus, elles permettent d'assurer le maintien de la cohérence des données et de minimiser le temps de réponse des transactions. L'architecture actuelle des systèmes de BDCSF n'est pas adaptée pour tenir compte de l'aspect temporel dans les RCSF. Aussi, les protocoles et les techniques de gestion des données, implémentés dans les BDCSF, s'intéressent plutôt à la minimisation de la consommation d'énergie dans le réseau. Dans le chapitre suivant, nous nous focalisons sur l'étude des aspects temporels dans les BDCSF et dans les bases de données classiques utilisées pour stocker les données de capteurs, afin de trouver quel type de bases de données est capable d'assurer le respect des contraintes temporelles dans les RCSF.

Partie II

Contributions

Chapitre 4

Étude comparative de deux approches de traitement de données dans les RCSF

Sommaire

4.1 Introduction

L'aspect temps-réel est un facteur important pour les applications de RCSF qui exigent une livraison de données dans les temps, pour refléter l'état courant de l'environnement. Cependant, la plupart des travaux sur les RCSF s'intéressent uniquement aux traitements des données et à la consommation d'énergie, afin d'améliorer la durée de vie des RCSF. Plusieurs travaux de recherches ont été faits sur le traitement des données dans RCSF. Certains travaux se focalisent sur l'amélioration des méthodes de collecte de données, et d'autres ont mis l'accent sur diverses techniques de traitement des requêtes. Au cours de ces dernières années, l'approche BDCSF, basée sur le traitement des requêtes, a montré des résultats efficaces dans de nombreux travaux de recherches. Les applications utilisant les RCSF adoptent des méthodes de collecte des données qui peuvent être périodiques, événementielles ou à la demande. Elles utilisent aussi des méthodes de stockage des données qui peuvent être centralisées ou distribuées. Ces applications exigent des délais stricts pour envoyer les données des nœuds de capteurs vers la station de base. Cependant, les RCSF sont souvent déployés sans donner beaucoup d'importance aux contraintes temporelles, y compris la contrainte de fraîcheur des données et les contraintes de délais d'exécution des transactions. Les contraintes temporelles diffèrent d'une application à une autre. La fraîcheur des données impose des limites sur la période de validité des données, à partir du moment où les mesures sont acquises par le capteur. En outre, les tâches sont soumises à des échéances, afin d'assurer une meilleure QdS à l'utilisateur. La fraîcheur et la validité temporelle des données, ainsi que le respect des échéances des transactions, sont des facteurs importants pour les applications de RCSF temps réel. Il devient nécessaire de bien étudier l'efficacité temporelle des mécanismes de gestion des données dans les RCSF. Cependant, on ne trouve pas dans la littérature sur les RCSF, des travaux qui s'intéressent à l'étude et à la comparaison des contraintes temporelles dans les approches de traitement des données. Aujourd'hui, les chercheurs s'intéressent aux techniques de traitement des données dans le RCSF telles que l'agrégation, le traitement des requêtes, l'indexation des données, pour simplifier l'extraction des données du réseau et augmenter sa durée de vie. Le reste de ce chapitre est structuré comme suit : dans la section 2, nous présentons les travaux connexes liés à ce travail. Dans la section 3, nous décrivons deux modèles de gestion des données de capteurs. Dans la section 4, nous présentons l'environnement de simulation et les différents tests réalisés. Par la suite, nous présentons et nous discutons les résultats obtenus. Enfin, dans la section 5, nous concluons ce chapitre, en livrant notre point de vue sur la meilleure technique qui contribue à l'amélioration des contraintes temporelles dans les RCSF.

4.2 Motivation et travaux connexes

Notre première contribution consiste à effectuer une étude comparative sur deux approches de gestion des données dans les RCSF. La première approche représente une collecte périodique des données des capteurs, dont le stockage est centralisé, dans une base de données distante. La deuxième approche représente une collecte des données à la demande. Elle est basée sur un traitement des requêtes et une BDCSF (TinyDB). Nous avons réalisé quelques expériences, afin de déterminer l'approche qui donne les meilleures performances temporelles, et qui assure la validité temporelle des données dans le RCSF. Nous avons considéré les paramètres de performance suivants : (i) le temps de convergence du réseau, qui représente le temps nécessaire aux nœuds pour se connecter à la station de base et pour construire et mettre à jour les tables de routage dans le réseau, (ii) le temps de collecte des données, et enfin (iii) le temps de réponse de la base de données aux requêtes "utilisateur". Nous avons étudié aussi les influences suivantes : le nombre de nœuds, le protocole MAC utilisé, les positions des nœuds et le nombre des sauts, les topologies du réseau et enfin l'influence du choix de la base de données sur le temps de réponse de requêtes. Nous considérons que l'amélioration des paramètres discutés dans cette étude contribuera au respect des contraintes temporelles dans les RCSF.

Parmi les travaux basés sur une approche de collecte des données avec un stockage centralisé, nous trouvons le système de traitement de flux des données proposé par Kitagawa *et al.* [Kitagawa *et al.*, 2010]. Les auteurs utilisent un système événementiel à requêtes continues, dont les données sont collectées à partir de plusieurs sites de RCSF. Ils ont résolu le problème de stockage des données en continu, en utilisant le SGBDR[1] PostgreSQL [Group, 2013], qui assure une lecture des données en mémoire pour les traitements des requêtes. Cependant, cette méthode nécessite un accès au disque pour chaque arrivée des données. Ceci augmente considérablement le temps de stockage des données.

Kanzaki *et al.* ont discuté un banc d'essai de RCSF basé sur une approche distribuée, nommée X-Sensor [Kanzaki *et al.*, 2010]. Il gère plusieurs sites de RCSF et s'adapte bien aux réseaux de capteurs à grande échelle. Ce banc d'essai est composé de trois parties : (i) un serveur qui fournit une interface Web, qui permet aux utilisateurs d'accéder aux sites de capteurs. Chaque site contient (ii) une passerelle et (iii) plusieurs capteurs de type MICAz [Ali *et al.*, 2011]. La passerelle collecte les données provenant des capteurs. Ensuite, elle les stocke dans une base de données PostgreSQL. La base de données est accessible par la passerelle et le serveur de bancs d'essais. Ce dernier gère les informations de toutes les passerelles. X-Sensor offre une interface web à l'utilisateur qui lui permet de trouver un site de capteurs, d'archiver les données de capteurs et aussi un banc d'essai expérimental.

[1] Système de Gestion de Base de Données Relationnelle

Becker *et al.* ont proposé une nouvelle application logistique utilisant les RCSF [Becker *et al.*, 2010]. Ils ont un système de stockage des données décentralisé. Chaque capteur utilise une mémoire flash pour écrire régulièrement les données détectées. Ensuite, la passerelle utilise les fichiers de la base des données intégrée (fichiers SQLite) pour écrire les données acquises. Les données sont envoyées par la suite à travers Internet à un serveur qui les stocke dans une base de données. Les auteurs ont montré que les RCSF sont applicables pour une surveillance du transport des aliments car les données peuvent être collectées en ligne, quelque soit l'emplacement des capteurs, même au milieu de l'océan Atlantique. Cependant, les problèmes d'auto-configuration et la mobilité des capteurs réduit la QdS des applications de RCSF dans le domaine de logistique.

Elias *et al.* ont proposé un modèle de surveillance utilisant les RCSF [Elias *et al.*, 2012]. Il utilise le service web REST[2] et les messages XML. Les données sont collectées et stockées dans une base de données. Ensuite, l'utilisateur se connecte à l'application à travers son mobile, pour obtenir des informations du RCSF. Un module d'analyse syntaxique convertit les données des capteurs sous format XML, pour être facilement réutilisées et déployées dans un autre environnement ou une autre plate-forme (application mobile Android). Les auteurs ont choisi le système de gestion de bases de données MySQL (SGBD) [Corporation, 2015] pour le stockage des données, car il peut soutenir une grande quantité de données recueillies et il peut aussi réduire les données redondantes et nulles. Le service Web REST a été choisi pour permettre l'interaction entre la base de données et les appareils mobiles. L'architecture REST s'adapte bien au modèle client-serveur, basé sur le protocole HTTP et qui est supporté par la plupart des dispositifs mobiles.

4.3 Modèles de réseau et scénarios

4.3.1 Collecte périodique des données avec un stockage centralisé

Dans ce scénario, nous avons utilisé un modèle de réseau composé de quatre éléments : (i) des nœuds de capteurs pour communiquer les informations sur l'environnement à l'utilisateur, (ii) une station de base pour recueillir les données des nœuds de capteurs, (iii) une base de données distante pour stocker les données reçues du RCSF, (iv) un ordinateur qui permet à l'utilisateur de se connecter à la station de base pour lui envoyer des commandes et pour afficher les données collectées. L'ordinateur permet à l'utilisateur de se connecter à la base de données pour envoyer des requêtes ou pour obtenir un historique des données stockées. Les nœuds de capteurs sont déployés de manière aléatoire à travers une zone et la station de base est placée en dehors du réseau.

[2] REpresentational State Transfer

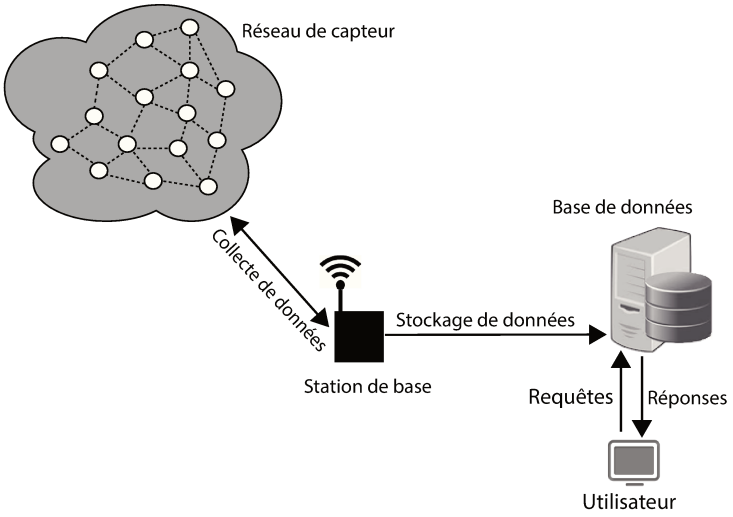


Figure 4.1: Collecte périodique des données avec une base de données distante

Les capteurs communiquent grâce à des communications sans fil. Ils captent des grandeurs physiques telles que la température, l'humidité ou la lumière et les envoient périodiquement vers la station de base. La station de base collecte les données des capteurs, et les transmet à l'utilisateur. La collecte des données est périodique, selon la période définie au départ par l'utilisateur ou le système. Les données sont affichées dans une interface graphique ou stockées dans une base de données distante. L'utilisateur peut interroger la base de données en utilisant des requêtes de type SQL pour obtenir des données historiques sur le RCSF (cf. Figure 4.1 et Figure 4.2). Dans la section suivante, nous testons les deux scénarios décrits ci-dessus et nous discutons les paramètres de performance pour les deux approches. Nous identifions et nous analysons également différents critères qui peuvent influer sur ces paramètres.

4.3.2 Traitement des requêtes avec une base de données abstraite

Dans ce scénario, nous utilisons la technique de traitement des requêtes dans les RCSF, dans laquelle nous intégrons une BDCSF, nommée TinyDB. Le modèle de réseau est composé de quatre éléments : (i) des capteurs sans fils, qui contiennent l'application TinyDB et qui assurent le traitement des requêtes, la gestion de la mémoire du capteur, ainsi que la gestion de la topologie du réseau, (ii) une station de base, (iii) l'ordinateur de l'utilisateur qui contient l'application "client" de TinyDB et qui représente l'interface graphique, (iv) une base de données distante pour stocker les données de capteurs. Plus de détails sur l'architecture du système TinyDB, se trouvent dans le chapitre 2.
L'utilisateur saisit une requête SQL, à travers une interface graphique fournie par TinyDB.

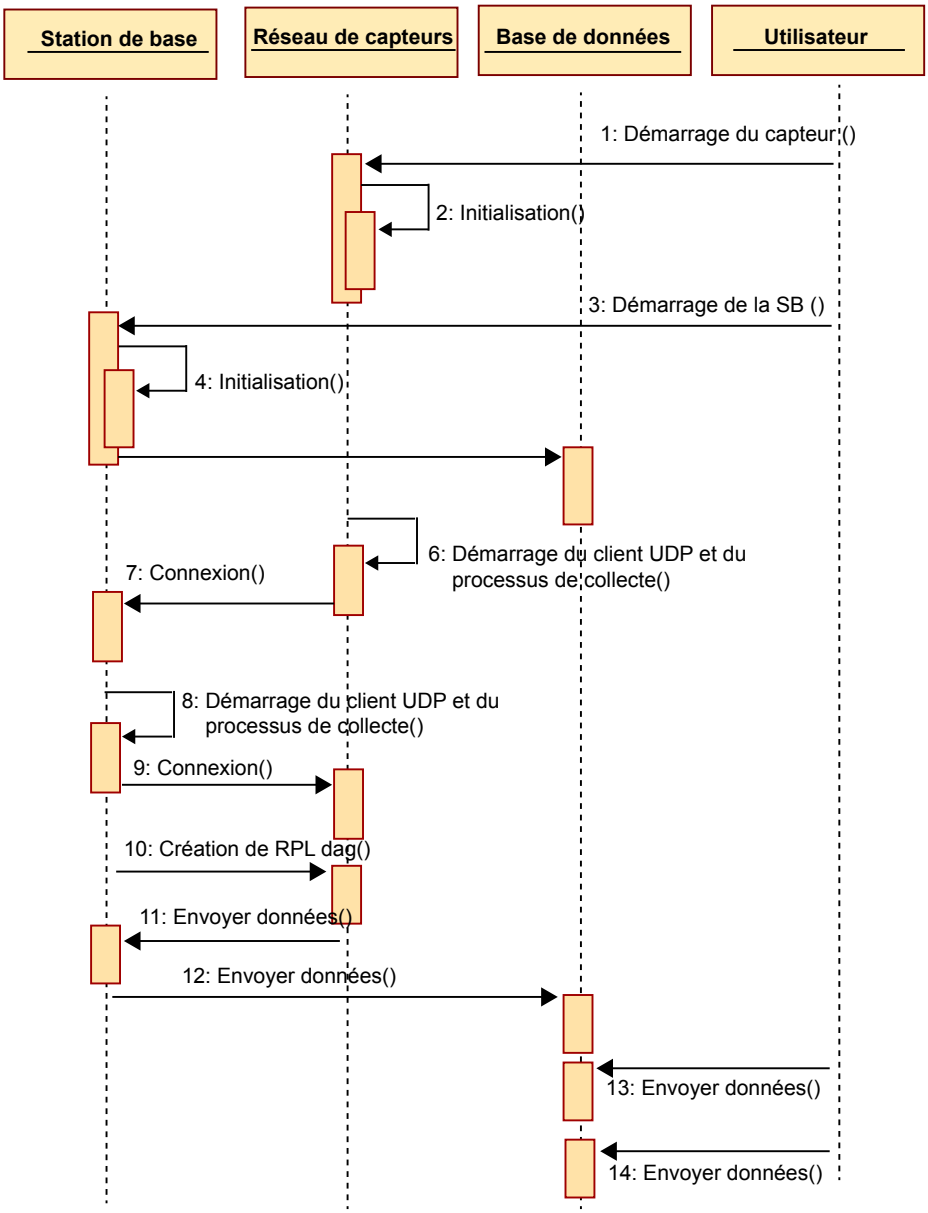


Figure 4.2: Diagramme de séquences pour une collecte périodique des données

Il définit, dans la requête, les données qui seront extraites du RCSF, et les envoie à la station de base. La station de base injecte la requête dans le réseau. Chaque capteur reçoit la requête et répond avec les données demandées, pour une durée définie dans la requête. La station de base reçoit les réponses, et les envoie à l'utilisateur via la BDCSF. Les réponses seront affichées par la suite dans l'interface graphique de l'utilisateur. Elles peuvent aussi être stockées dans une base de données distante pour être historiées (cf. Figure 4.3).

Dans la section suivante, nous testons les deux scénarios décrits ci-dessus et nous discutons

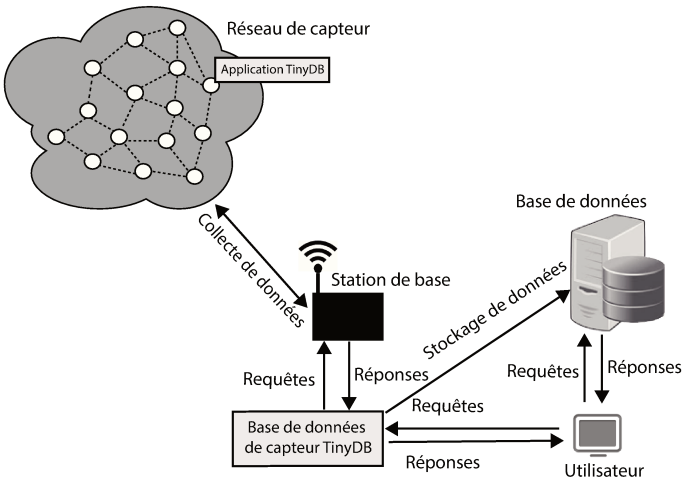


Figure 4.3: Traitement de requêtes avec une BDCSF (TinyDB)

les paramètres de performance pour les deux approches. Nous identifions et nous analysons également différents critères qui peuvent influer sur ces paramètres.

4.4 Simulations et analyse des performances

Nous avons considéré certains paramètres de performance pour tester les deux approches. Ces paramètres sont : (i) le temps de convergence du réseau, qui représente un facteur important pour garantir les contraintes temporelles dans le RCSF. Ce processus précède le processus de collecte des données. Il peut affecter le temps d'arrivée des données lorsqu'il est long. Nous avons considéré aussi (ii) le temps de collecte des données, qui représente un facteur important pour garantir la fraîcheur des données, ainsi que le délai de livraison des données. Lorsque la collecte des données est rapide, la fraîcheur des données sera améliorée, (iii) le temps de réponse de la base de données est considéré comme l'un des facteurs les plus importants pour analyser les performances des deux approches. Nous discuterons quelques contraintes telles que le nombre des capteurs, la position de la station de base, ainsi que les autres nœuds dans le réseau, le nombre de sauts vers la station de base, le protocole MAC utilisé, la topologie du réseau et enfin le choix de la base de données. Ces contraintes peuvent affecter les aspects temporels dans un RCSF.

4.4.1 Environnement de simulation

Dans le premier scénario, nous avons utilisé COOJA [Österlind *et al.*, 2006], un simulateur de réseau pour le système d'exploitation pour RCSF ContikiOS [Dunkels *et al.*, 2004]. Cooja fournit différents types de capteurs (Tmote Sky, MicaZ, Z1 Mote, wismote). Il

prend également en charge un ensemble de modèles de propagation de radio (UDGM[3] Constant Loss, UDGM Distance Loss). ContikiOS intègre le protocole de routage RPL et différents standards tels que 6LoWPAN et CoAP[4] [Shelby *et al.*, 2014]. Il implémente aussi le protocole IEEE 802.15.4, ainsi que deux piles de communication uIP et Rime, et les protocoles Internet IPV6 et IPV4.

Nous avons créé un nœud "puits" (sink) considéré comme une station de base et un ensemble de nœuds expéditeurs de Type Tmote Sky [Corporation, 2006]. Ils possèdent une mémoire "flash" de 48 Ko. Les capteurs sont répartis aléatoirement dans une zone de 100m sur 100m. La station de base est placée en dehors de la zone de déploiement des capteurs. Nous avons fait varier le nombre de capteurs de 10 à 100 avec un pas de 10 pour chaque simulation. Chaque nœud envoie ses mesures de température et d'humidité vers la station de base toutes les 60 secondes. Nous avons choisi trois systèmes de gestion de bases de données (PostgreSQL, MySQL et SQLite) pour le stockage des données de capteurs. Le Tableau 4.1 résume les paramètres de simulation utilisés dans le premier scénario.

Paramètre	**Valeur**
Système d'exploitation	ContikiOS 2.6
Simulateur	Cooja
Modèle de capteur	Tmote sky
Zone de simulation	100m*100m
Nombre de nœuds	10 à 100
Portée de transmission d'un nœud	80m
Taille maximale d'un paquet	128 bytes
Vitesse de transmission	250 kbps
Modèle de propagation radio	Unit Disk Graph Medium (UDGM) Distance Loss

Tableau 4.1: Paramètres de simulation

Nous avons utilisé le protocole de routage pour les réseaux à faible puissance et avec pertes, (RPL[5]) [Winter *et al.*, 2012], pour assurer la communication entre les capteurs et la station de base. RPL est plus efficace en terme du temps de livraison des données que les autres protocoles existants tels que DSR[6], CTP[7] [Radoi *et al.*, 2012] ou LOAD[8]. Ce dernier représente une version dérivée de protocole de routage AODV[9] [Vucinic *et al.*, 2013]. RPL fournit des chemins de routage efficaces garantissant l'échéance de la livraison des données. Nous avons aussi utilisé le protocole ContikiMac [Dunkels, 2011] qui a montré une gestion efficace du temps de transmission des paquets de données. ContikiMac utilise

[3]UDGM : Unit Disk Graph Medium
[4]CoAP : Constrained Application Protocol
[5]RPL : Routing Protocol for Low power and Lossy Networks
[6]DSR: Dynamic Source Routing
[7]CTP: Collection Tree Protocol
[8]LOAD: Lightweight On-demand Ad hoc Distance-vector
[9]AODV: Ad hoc On-Demand Distance Vector Routing

un mécanisme de réveil pour définir un calendrier précis entre les transmissions de paquets des données. Le Tableau 4.2 résume les différentes couches et protocoles utilisés dans la simulation.

Couche transport	UDP
Couche réseau	IPV6/RPL
Couche Mac	ContikiMAC
Méthode d'accès	CSMA-CA
Couche physique	IEEE 802.15.4

Tableau 4.2: Pile de protocoles

Dans le deuxième scénario, nous avons utilisé le système TinyDB comme BDCSF. TinyDB fonctionne sous le système d'exploitation TinyOS. Il ne peut être utilisé qu'avec des capteurs Mica2 possède une mémoire "flash" de 128 Ko. L'exécution du programme TinyDB utilise plus de 60 Ko de mémoire (plus que la mémoire utilisée par des capteurs de type Tmote Sky, utilisant une mémoire de 48 Ko). Nous avons choisi le système TinyDB dans cette étude parce qu'il représente le système le plus utilisé dans les travaux de recherche. De plus, il a montré son efficacité dans la minimisation de l'énergie et dans la gestion des données dans les RCSF. Les capteurs sont dispersés aléatoirement. Ils sont interrogés à la demande de l'utilisateur avec des requêtes de type SQL, à travers la station de base. L'utilisateur possède une interface graphique qui lui permet de saisir ses requêtes. Le Tableau 4.3 résume les paramètres de simulation utilisés dans le deuxième scénario.

Paramètre	**Valeur**
Système d'exploitation	TinyOS
Simulateur	Tossim
Modèle de capteur	Mica2
Zone de simulation	100m*100m
Nombre de nœuds	10 à 100
Portée de transmission d'un nœud	80m
Taille maximale d'un paquet	128 bytes
Vitesse de transmission	250 kbps

Tableau 4.3: Paramètres de simulation

TinyDB utilise un routage sémantique à base d'arbres [Gehrke and Madden, 2004]. Il utilise les protocoles Drip et Drain. Drip est un composant de la couche transport dédié à la diffusion des messages (requêtes) à travers le réseau. Drain est un protocole de routage pour la collecte des résultats. Chaque nœud transmet la requête à ses voisins dans le réseau, afin de former l'arbre de calcul d'itinéraires. Ce processus se termine lorsque tous les nœuds ont reçu la requête. TinyDB utilise la méthode d'accès CSMA utilisée par défaut par le simulateur TOSSIM. Il utilise également le protocole LPL (Low Power Listening) pour les communications à faible puissance. Le Tableau 4.4 résume les différentes couches

du protocole utilisé dans la simulation.

Couche transport	Drip/Drain
Couche réseau	Semantic Routing Tree protocol
Couche liaison	Low-Power Listening
Couche Mac	CSMA
Couche physique	IEEE 802.15.4

Tableau 4.4: Pile de protocoles

4.4.2 Description des simulations

Dans le premier scénario, chaque nœud envoie ses données à la station de base, en fonction d'un intervalle d'envoi défini par l'utilisateur. Nous avons fixé cet intervalle à 60 secondes avec un temps d'attente aléatoire, avant chaque envoi de paquets, pour éviter que les capteurs envoient leurs données en même temps. Nous avons par la suite activé le composant numérique de capteur (SHT11), relatif à la détection de l'humidité et de la température. Chaque nœud envoie ses données à la station de base, puis, il reste à l'écoute de ses voisins. Lorsque la station de base reçoit les données de capteurs, elle les stocke dans une base de données distante en utilisant des requêtes d'insertion SQL. L'Algorithme 1 présente le processus de collecte des données inclus au niveau de chaque capteur.

Algorithm 1: Processus d'envoi UDP

```
initialization;
PROCESS BEGIN ();
Define PERIOD 60;
Define SEND_TIME random_Value;
SENSORS ACTIVATE(sht11_sensor);
while 1 do
    if ev = tcpip_event then
        tcpip_handler();
    end
    if etimer_expired(PERIOD) then
        etimer_reset(PERIOD);
        ctimer_set(BACOFF_TIME, SEND_TIME, send_packet, NULL);
    end
end
PROCESS END ();
```

Dans le deuxième scénario, lorsque TinyDB reçoit une requête "utilisateur", il l'analyse, puis transforme en un plan d'exécution. Avant de l'injecter dans le réseau, TinyDB utilise une méthode de routage sémantique à base d'arbre, nommée (SRT). Cette méthode permet de déterminer quels sont les nœuds dans l'arbre de routage qui possèdent les capacités pour

répondre à la requête ”utilisateur” et pour participer à la production des résultats. La méthode SRT comporte deux phases : la première consiste à construire un arbre initial permettant aux nœuds parents de connaître les capacités de leurs enfants. SRT diffuse une requête y compris le type des données demandées. Chaque nœud sélectionne son parent et il répond avec les informations demandées. Ensuite, chaque parent reçoit l'identifiant et les réponses de ses enfants. De la même manière, chaque parent choisit son propre parent jusqu'au nœud racine. Dans la deuxième phase, la vraie requête est diffusée pour les nœuds ayant des réponses pertinentes. Une fois que la requête est diffusées dans le réseau, chaque nœud qui reçoit la requête commence son exécution. L'exécution de la requête inclut les processus de filtrage et d'agrégation des résultats, suivant le plan d'exécution défini par TinyDB. Ensuite, les données sont délivrées aux nœuds parents et enfin vers la station de base. TinyDB contrôle les taux de transmission des résultats. Il les adapte afin de réduire la consommation d'énergie. Les résultats de la requête seront affichés sous forme d'un tableau à l'aide d'une interface graphique basée sur Java.

4.4.3 Influence du nombre de nœuds sur le temps de convergence

Le temps de convergence du réseau peut être défini comme étant le temps nécessaire pour que les capteurs se connectent à la station de base, ainsi que pour la construction des tables de routage dans le réseau. Cette étape peut affecter le temps d'arrivée des données, car elle vient juste avant le processus de collecte des données. Sur la figure 4.4, le temps de convergence du réseau, dans le scénario 1, augmente lorsque le nombre de nœuds augmente. Le temps de convergence du réseau dépend du processus de construction de l'arbre de routage du réseau. Dans le scénario 1, le protocole de routage RPL construit l'arbre de routage DODAG[10] basé sur les étapes suivantes : d'abord, la racine de l'arbre du réseau diffuse un message, nommé DIO (Informations DODAG Object) à tous les nœuds voisins. Chaque nœud qui reçoit le message, décidera par la suite s'il rejoint ou pas la structure de l'arbre. Chaque nœud qui rejoint la structure de l'arbre présentera un itinéraire jusqu'à la racine. La construction de la table de routage finale dépend du nombre de nœuds ainsi que du coût des trajets entre des nœuds.

On note aussi dans la figure 4.5 que le temps de convergence pour TinyDB est inférieur à celui du scénario de la collecte des données.

TinyDB construit un arbre de routage ou un graphe acyclique orienté (DAG) composé de tous les capteurs y compris la station de base. TinyDB utilise une méthode de sélection multisauts, basée sur un routage efficace qui tient compte de la consommation d'énergie de chaque nœud et de sa capacité à fournir la réponse. Les capteurs doivent prendre des décisions intelligentes de routage , où chaque nœud sélectionne un parent basé sur

[10]Destination Oriented Directed Acyclic Graph

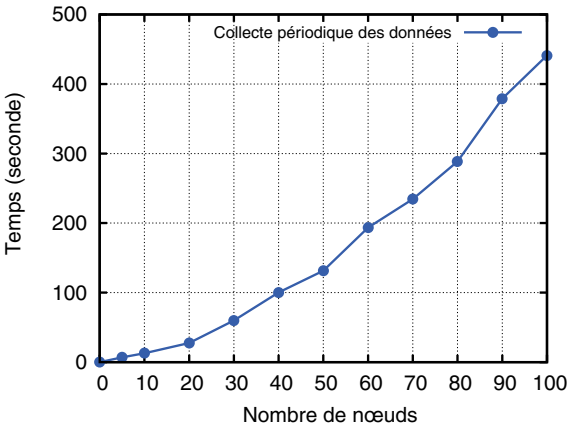


Figure 4.4: Temps de convergence avec une collecte périodique des données

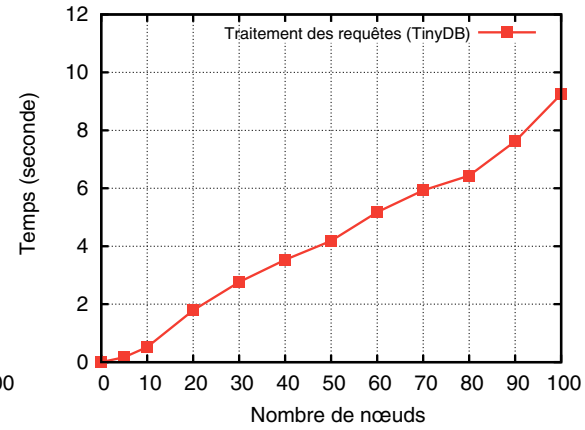


Figure 4.5: Temps de convergence avec une BDCSF (TinyDB)

la qualité des liens vers la station de base. Chaque nœud doit garder une courte liste des voisins qui ont réussi leurs récentes transmissions ainsi que quelques informations sur le routage, la connectivité et la qualité de la liaison avec leurs voisins. Dans le premier scénario, tous les capteurs participent dans la procédure de convergence, ce qui augmente considérablement le temps de convergence. En revanche, dans le deuxième scénario, une sélection des nœuds (les plus performants) qui peuvent participer à cette procédure.

4.4.4 Influence du nombre de nœuds sur le temps de collecte des données

Le temps de collecte des données représente le temps nécessaire pour obtenir les réponses de tous les capteurs connectés à la station de base. Il dépend toutefois de la capacité des capteurs, de leurs positions et de la qualité du trajet suivi pour la livraison des données, en fonction du nombre de sauts jusqu'à la station de base. Généralement, les capteurs utilisent des chemins multi-sauts pour envoyer leurs données à la station de base. Parfois, la transmission des données conduit à des retards, ce qui affecte la validité temporelle des données. Comme conséquence, les données ne reflètent plus l'état actuel de l'environnement.

La figure 4.6 montre le temps passé par les nœuds pour terminer l'envoi de leurs données, dans les deux scénarios. On note que sur les deux courbes, le temps augmente lorsque le nombre de capteurs augmente pour les deux techniques. Le temps nécessaire pour envoyer les données dépend du nombre de sauts et de la disponibilité des nœuds parents pour envoyer les données reçues de leurs enfants. La disponibilité des nœuds parents n'est pas garantie tout le temps parce que le choix des nœuds parents dépend de la décision des nœuds fils, qui dépend de l'adhésion à la structure de l'arbre de routage.

Nous remarquons aussi dans la figure 4.6 que le temps nécessaire pour obtenir toutes

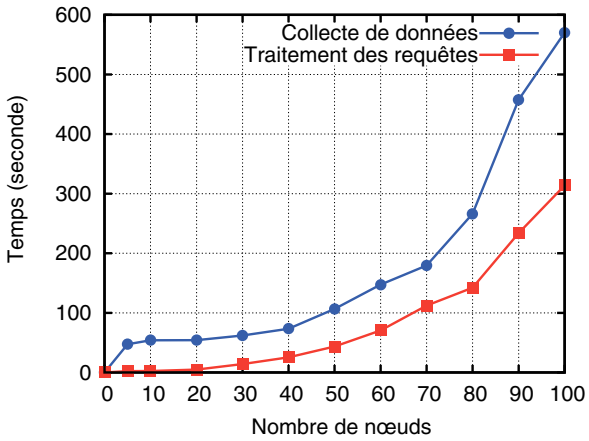


Figure 4.6: Temps de collecte des données

les réponses de tous les capteurs en utilisant la technique de traitement des requêtes est inférieur à celui de la collecte des données. Avec TinyDB, les données sont régulièrement déclarées, agrégées et filtrées à l'intérieur du réseau de capteurs pour maintenir l'ensemble des informations de routage. Les nœuds parents qui sont près de la racine se mettent d'accord avec leurs enfants sur un intervalle de temps pour la réception des données, ce qui améliore fortement le temps de collecte des données. L'ensemble du processus est répété pour chaque période et pour chaque requête.

4.4.5 Influence des protocoles MAC sur le nombre de données envoyées

L'objectif de cette expérience est d'observer la durée d'un cycle complet de collecte des données, ainsi que les doubles envois de données avec une collecte périodique des données, et avec une méthode basée sur un système de traitement des requêtes. Nous avons fait varier le nombre de capteurs à chaque fois. Dans le Tableau 4.5, nous remarquons que le nombre de données transmises avec la méthode de traitement des requêtes est supérieur à celui de la collecte périodique des données. Le protocole de la couche MAC peut être considéré comme l'un des facteurs qui explique ces résultats.

Dans le scénario de traitement de requêtes, le simulateur TOSSIM implémente par défaut le protocole MAC (CSMA). Le principe de base de ce protocole est d'écouter avant la transmission. Chaque nœud vérifie l'état du canal avant chaque envoi. Si le canal n'est pas occupé, le capteur envoie ses données, sinon, il se rendort durant une période de temps aléatoire, appelée « backoff », jusqu'à ce que le canal devienne libre. Lorsque le nombre de nœuds augmente, il existera une concurrence pour accéder au canal de transmission. Cette concurrence provoque un retard de transmission et une accumulation de périodes backoff, ce qui augmente bien évidemment le temps nécessaire pour la collecte des données de tous les capteurs. D'autre part, chaque capteur envoie à plusieurs reprises son paquet dans la

	Collecte périodique		Traitement de requêtes	
Nb. Nœuds	Cycle complet (ms)	Envoi dupliqué	Cycle complet (ms)	Envoi dupliqué
5	47.430	0	2.203	0
10	53.955	0	2.434	0
20	54.157	0	4.766	21
30	62.082	0	14.060	102
40	73.302	0	25.441	121
50	106.609	0	43.543	174
60	147.357	2	70.502	294
70	179.478	1	112.261	308
80	266.357	15	142.114	405
90	457.570	78	233.737	420
100	570.100	72	313.832	633

Tableau 4.5: Envoi dupliqué des données

période entre deux réveils successifs (wake-up period), jusqu'à ce qu'il reçoive un accusé de réception du récepteur, ce qui explique l'augmentation de nombre d'envois dupliqué dans ce scénario.

Dans le scénario de collecte périodique, les capteurs implémentent le protocole Mac à réveil cyclique (Duty-cycling), nommé ContikiMac, qui utilise un mécanisme de réveil (wake-up) pour écouter les transmissions de ses voisins et pour garder l'émetteur-récepteur hors tension, afin de réduire la consommation d'énergie dans le RCSF. La méthode de synchronisation inclut : (1) l'intervalle entre chaque transmission de paquets, (2) l'intervalle entre chaque détection physique de porteuse ou CCA (Clear Channel Assessment), (3) le temps entre envoi/réception d'accusé de réception, (4) le temps nécessaire pour détecter avec succès un accusé de réception en provenance du récepteur, (5) le temps nécessaire pour avoir un indicateur stable du niveau de la puissance du signal reçu (RSSI[11]). Le RSSI est nécessaire aussi pour avoir une indication stable de CCA. L'utilisation du protocole ContikiMac dans ce scénario, a pour objectif d'assurer une synchronisation précise entre les transmissions des données dans le réseau, ce qui explique la diminution des paquets dupliqués lors des transmissions (cf. Tableau 4.5).

4.4.6 Influence du choix de la base de données sur le temps de réponse

Cette expérience consiste à collecter les données à partir du RCSF, et ensuite à les insérer dans une base de données distante. Nous avons utilisé trois systèmes de gestion de bases des données (SGBD) : PostgreSQL [Group, 2013], MySQL [Corporation, 2015] et SQLite [Kreibich, 2010]. Nous avons évalué leurs temps de réponse aux requêtes des utilisateurs.

[11] Received Signal Strength Indicator

Le temps de réponse représente le temps total passé par la requête pour compléter son exécution (cf. tableau 4.6). D'abord, nous allons présenter les trois SGBD utilisés :

PostgreSQL est un système de gestion de bases de données relationnelles (SGBDR), basé sur un projet de recherche appelé Postgres, de l'Université de Californie. PostgreSQL supporte une grande partie du standard SQL et il offre de nombreuses fonctionnalités avancées telles que l'intégrité transactionnelle des données, l'utilisation de requêtes complexes et il implante aussi l'héritage des tables. L'utilisateur peut ajouter de nouveaux types de données, des opérateurs, des fonctions d'agrégation, et des méthodes d'indexation.

MySQL est un système de gestion de bases de données développé et distribué par la société "MySQL AB". Il supporte plusieurs fonctionnalités telles que la contrainte de clé étrangère, les fonctions multithreads, les procédures stockées partielles et les moteurs de stockage transactionnel et non transactionnel. Les moteurs de stockage agissent comme un gestionnaire pour différents types de tables, par exemple MyISAM, le moteur de stockage par défaut de MySQL et le moteur de stockage InnoDB ont été conçus pour des performances maximales. MySQL prend en charge les principales contraintes d'intégrité référentielle.

SQLite est un système de gestion de bases de données relationnelles. Il représente une version simplifiée de systèmes de bases de données. Il fournit un moteur de base de données SQL qui lit et écrit directement dans des fichiers sur le support de stockage (disque dur). Il ne nécessite pas une configuration de serveur. Il supporte les tables, les déclencheurs, les vues et plusieurs fonctionnalités du langage de requêtes SQL. Les transactions dans SQLite ont les propriétés ACID [12]. Elles assurent un accès sécurisé à partir de multiples processus ou threads.

Type de requêtes	PostgreSQL	MySQL	SQLite
Requêtes d'insertion	9 397 ms	48 626 ms	72 788 ms
Requêtes de sélection	992 ms	690 ms	225 ms

Tableau 4.6: Temps de réponse moyen aux requêtes

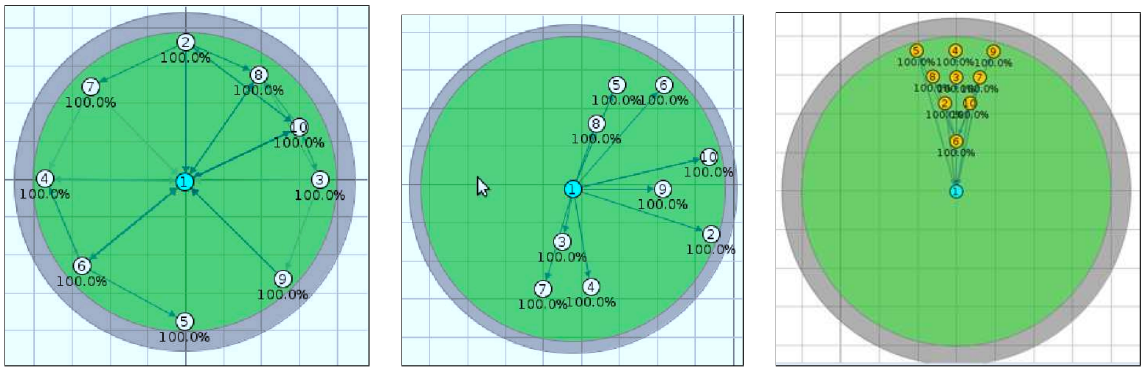
Le tableau 4.6 montre que le système SQLite donne le meilleur temps de réponse pour les requêtes de sélection. On trouve par la suite MySQL et enfin PostgreSQL. Pour les requêtes d'insertion, le système PostgreSQL possède le meilleur temps de réponse, qui est inférieur à celui de MySQL et SQLite. On peut expliquer ces résultats comme suit : SQLite passe beaucoup de temps pour insérer les données, car il ne dispose pas d'un serveur central pour coordonner les accès. Il doit fermer et ouvrir le fichier dans lequel il écrit ses données. Ensuite, il doit vider sa mémoire cache à la fin de chaque transaction, ce qui augmente le temps de réponse à la requête. SQLite applique aussi un mécanisme

[12] ACID : Atomicité, Cohérence, Isolation et Durabilité

de verrouillage sur la base de données durant les opérations d'écriture. Il ne permet pas de multiples écritures simultanées dans la base de données. Au contraire, PostgreSQL permet une exécution multithread aux opérations d'insertion ou d'écriture. MySQL est bien meilleur que PostgreSQL pour les requêtes de sélection car il utilise le mécanisme de "cache de requêtes" pour accélérer les requêtes et la lecture des données. Il utilise aussi le moteur de stockage InnoDB qui optimise la recherche des données pour l'utilisateur. SQLite est plus rapide que MySQL dans la lecture des données. Il n'a pas besoin de se connecter au serveur. Il utilise un accès au disque pour lire et écrire directement à partir des fichiers dans la base de données.

4.4.7 Influence des positions des nœuds et du nombre de sauts sur le temps de collecte des données

Dans cette section, nous discutons les performances du réseau de capteurs en modifiant la position des nœuds. Dans toutes les expériences, la station de base est placée à 40 mètres de chaque nœud (cf. Figure 4.7). La première expérience consiste à placer la station de base au centre du réseau. Chaque nœud envoie ses données à la station de base en utilisant un seul saut. Dans la deuxième expérience, la station de base est placée sur les bords du réseau. Chaque nœud envoie ses données à la station de base en utilisant deux sauts. Dans la dernière expérience, nous avons placé les nœuds sous forme d'un arbre et nous avons observé si cette forme permettra d'améliorer le temps d'arrivée des données. Chaque nœud envoie ses données à la station de base en utilisant des sauts multiples.



(a) Transmissions avec un saut (b) Transmissions avec deux sauts (c) Transmissions à sauts multiples

Figure 4.7: Position des nœuds et nombre de sauts

Nous remarquons dans le Tableau 4.7 que le temps d'un cycle de collecte des données change d'une expérience à une autre. Généralement, la position de la station de base, ainsi que la position des nœuds peut affecter le temps de collecte des données. Ceci est dû au nombre de sauts utilisés par chaque nœud pour envoyer ses données à la station de

Nombre de sauts	Durée d'un cycle complet	Moyenne du temps de réponse
Transmissions avec un saut	32 931 ms	3 659 ms
Transmissions avec deux sauts	34 740 ms	3 860 ms
Transmissions à sauts multiples	53 528 ms	5 947 ms

Tableau 4.7: Influence du nombre de sauts

base. Le temps de collecte des données augmente avec le nombre de sauts utilisés.

4.4.8 Influence des topologies du réseau sur le temps de collecte des données

Les RCSF peuvent avoir différentes topologies qui ont une influence forte sur le temps de collecte des données. Dans cette expérience, nous avons testé 15 nœuds et une station de base avec quatre topologies différentes de réseau (arbre, étoile, Mesh et Grid). Dans une topologie à base d'arbre, les nœuds sont organisés par niveaux hiérarchiques. L'arbre est composé par des nœuds enfants, des nœuds parents et des nœuds relais. Ils communiquent tous ensemble et transmettent leurs données au nœud racine (station de base) par des transmissions multi-sauts. Dans une topologie en étoile, la station de base est placé au centre du réseau et les autres nœuds sont dans son voisinage. Ils communiquent avec la station de base en utilisant des transmissions à un seul saut. Dans une topologie maillée, un nœud peut communiquer avec plusieurs voisins et un message peut prendre différents chemins pour atteindre la destination. De plus, lorsqu' un nœud rate sa transmission, le réseau se réorganise pour maintenir la communication avec la station de base. Dans une topologie en grille, le réseau est divisé en grille. Un nœud est élu comme coordinateur de grille et sera responsable de toutes les transmissions d'informations dans la grille. Les nœuds envoient périodiquement leurs données au coordonnateur, qui les transmet par la suite à la station de base. Les résultats du Tableau 4.8 représentent le temps passé pour la collecte des données de tous les capteurs dans le RCSF, pour les différentes topologies et en utilisant les deux scénarios.

Topologies	Collecte des données périodique	Traitement de requêtes
Etoile	62 312 ms	1 832 ms
Mesh	56 002 ms	1 362 ms
Grille	54 121 ms	2 163 ms
Arbre	53 515 ms	2 143 ms

Tableau 4.8: Influence des topologies de réseau

On note que le temps de collecte des données en utilisant la topologie à base d'arbre est

inférieur aux autres temps pour les autres topologies. On peut expliquer ces résultats par le fait qu'on utilise le protocole de routage RPL qui ne s'adapte pas aux topologies étoile, maillée et grille, car il prend beaucoup de temps pour construire l'arbre de routage et pour définir les chemins vers la racine. Dans le deuxième scénario de traitement des requêtes, la topologie maillée offre une meilleure durée de collecte des données par rapport aux autres topologies. Dans la topologie maillée, tous les nœuds coopèrent à la transmission des données dans le réseau. La même technique est utilisée par TinyDB, les nœuds prennent des décisions de routage intelligent, où chaque nœud sélectionne un parent en se basant sur la qualité des liens vers la station de base.

4.5 Conclusion

Dans ce chapitre, nous avons étudié et comparé les aspects temporels dans une approche de collecte des données à partir d'un RCSF, en utilisant une base de données classique, par rapport à une approche de traitement des requêtes qui inclut une BDCSF, nommée TinyDB. Nous avons évalué quelques performances temporelles telles que le temps de collecte des données, les temps de réponse des différentes bases des données, et le temps de convergence du réseau dans les deux approches. Nous avons trouvé, selon les tests effectués, que de nombreux facteurs peuvent influer sur les contraintes temporelles dans un RCSF. Nous avons déterminé que la topologie du réseau et le protocole de routage ensemble peuvent jouer un rôle important sur le temps de collecte des données. Le temps de convergence a également une influence sur le processus de collecte des données. Nous avons montré clairement que les meilleurs résultats en termes de temps de réponse sont obtenus en utilisant une BDCSF par rapport à une base de données classique. Donc, nous pouvons conclure que le choix de la topologie du réseau et du protocole de routage peut améliorer le respect des contraintes temporelles dans le RCSF. Dans le chapitre 5, nous proposons un nouveau modèle de traitement des requêtes pour améliorer la validité temporelle des données dans le RCSF.

Chapitre 5

Un modèle de traitement de requêtes temps réel

Sommaire

5.1 Introduction

Les RCSF sont conçus pour surveiller et contrôler l'environnement ainsi que les activités industrielles. Aujourd'hui, il existe de nombreuses applications qui utilisent les RCSF telles que les applications dédiées à la surveillance de processus industriels et à la prévention des catastrophes naturelles. Ces applications ont besoin de données cohérentes qui reflètent l'état courant de l'environnement, pour prendre des mesures préventives si nécessaire. Cependant, le maintien de la validité temporelle des données peut échouer pour plusieurs raisons : (i) une augmentation du temps de traitement des données, due à la grande quantité des données traitées et à la capacité de calcul limitée des capteurs, (ii) des communications asynchrones entre les capteurs qui provoquent des retards de transmission des données (iii) et la topologie dynamique du réseau lorsqu'il y a des défaillances au niveau des capteurs.

La validité temporelle des données varie d'un système à l'autre. Elle dépend de la nature du système, c'est-à-dire, s'il fonctionne avec des contraintes temps réel strictes, souples ou molles. Avant chaque collecte de données, l'utilisateur ou le système définit à l'avance la durée de validité temporelle des données, car les capteurs ne sont pas capables de fixer seuls le taux de changement des données ou de définir leur validité temporelle. Le concept de cohérence temporelle a été proposé afin de mesurer la façon dont les données reflètent l'état actuel de l'environnement. Il comporte deux aspects : une cohérence absolue qui reflète l'état courant de l'environnement et une cohérence relative qui concerne la cohérence entre les données utilisées pour dériver d'autres données [Madden, 2009]. En général, une donnée en temps réel est temporellement valide à l'instant (t) si la somme de sa date d'échantillonnage et de sa validité temporelle est inférieure à l'instant (t). Ramamritham [Ramamritham, 1993] a défini un modèle pour les données temps réel, dans lequel il a associé un intervalle de validité absolu à une donnée temps réel.
Le modèle est le suivant : $\mathrm{d} = (d_{valeur}, \mathrm{d}_{estampille}, \mathrm{d}_{avi})$, où d_{valeur} représente la valeur de la donnée, $\mathrm{d}_{estampille}$ représente la date de mesure de la donnée et d_{avi} représente l'intervalle de validité absolue de la donnée. Soit R l'ensemble des données utilisées pour dériver une nouvelle donnée, avec d et d' $\in$ R. On considère que la donnée (d) est temporellement cohérente si elle est cohérente d'une manière absolue, i.e. $instant_{courant} - d_{estampille} \leq d_{avi}$ et cohérente d'une manière relative par rapport à d', i.e. $|d_{estampille} - d'_{estampille}| \leq d_{rvi}$.

Trois approches ont été proposées pour maintenir la validité temporelle des données dans une BDTR[1] : l'approche One-One (OO), l'approche Half-Half (HH) et l'approche More-Less (ML) [Ramamritham *et al.*, 2004].

- **L'approche One-One** : dans cette approche, la période et l'échéance relative de la

[1] Bases des données temps réel

transaction sont égaux à la durée de validité temporelle de la donnée. Selon Xiong *et al.* [Xiong and Ramamritham, 2004], cette approche n'est pas avantageuse car les données peuvent perdre leur validité temporelle due à l'intervalle de temps séparant deux transactions consécutives, qui peut être parfois plus grand que l'intervalle de validité temporelle des données.

- **L'approche Half-Half** : selon cette approche, la période et l'échéance relative de la transaction doivent être inférieures ou égales à la moitié de l'intervalle de validité de la donnée. La validité temporelle de la donnée est garantie tant que la période et l'échéance de la transaction sont égales à la moitié de l'intervalle de validité temporelle de la donnée. Selon Xiong *et al.* [Xiong and Ramamritham, 2004], même si la fraîcheur des données est garantie, cette approche engendre une charge de travail du système au moins deux fois plus grande que celle de l'approche One-One.

- **L'approche More-Less** : le principe de cette approche est de réduire au minimum la charge de travail des transactions des capteurs tout en garantissant la validité temporelle des données. En effet, dans cette approche, la période d'une transaction est fixée au plus à la moitié de l'intervalle de validité de la donnée, alors que l'échéance relative doit être inférieure à la moitié de l'intervalle de validité temporelle de la donnée. Toutefois, la somme de la période et de l'échéance relative est toujours égale à la longueur de l'intervalle de validité temporelle de la donnée.

Nous avons structuré ce chapitre comme suit : dans la section 2, nous présentons la problématique et les motivations pour améliorer la validité temporelle des données. Nous citons, dans la section 3, les travaux liés au problème de cohérence temporelle des données dans les RCSF. Dans la section 4, nous décrivons un nouveau modèle de réseau dynamique basé sur l'auto-sélection des nœuds "puits". Il s'appuie sur la latence de communication entre les capteurs. Nous décrivons également un nouveau modèle de données temporelles, de nouvelles clauses SQL et un mécanisme de traitement des requêtes. Dans la section 5, nous discutons les résultats des simulations en les comparant avec les résultats d'autres systèmes de traitement des requêtes dans les RCSF. Enfin, dans la section 6, nous concluons le chapitre en montrant l'efficacité de notre mécanisme de traitement des requêtes.

5.2 Problématique et motivations

La problématique de la cohérence temporelle des données dans les RCSF a attiré l'attention de plusieurs chercheurs. Ils se sont focalisés principalement sur des méthodes basées sur la planification, l'affectation de priorités aux données ou sur des méthodes d'agrégation. Cependant, il existe peu de travaux qui utilisent les méthodes de traitement des données basées sur les requêtes pour résoudre ce problème, ce qui nous a conduit à proposer un

nouveau modèle de requêtes SQL. Ce modèle s'appuie principalement sur un modèle de réseau dynamique basé sur une auto-sélection des nœuds "puits". Il tient compte de la qualité du chemin en matière de latence, afin d'améliorer la diffusion des requêtes et la collecte des données. Il permet à l'utilisateur de communiquer facilement avec les capteurs, afin qu'il puisse mettre à jour l'intervalle de validité temporelle associée aux données, tout au long du processus de collecte des données. Nous avons choisi l'approche de traitement des requêtes, car elle a montré son efficacité dans la gestion des données dans les RCSF. En effet, cette approche considère le RCSF comme une grande base des données distribuée qui peut être interrogée à travers un langage de requêtes similaire à SQL, ce qui permet à l'utilisateur d'exprimer facilement la requête pour collecter les données du RCSF. Nous avons défini un nouveau modèle de requêtes SQL avec de nouvelles clauses SQL. La première clause est nommée *data.AVI* (data.Absolute Validity Interval). Elle permet à l'utilisateur de définir la validité temporelle des données. La deuxième clause est nommée *échéance* (deadline). Elle permet à l'utilisateur de définir une échéance pour une requête. Ces valeurs sont transmises à tous les nœuds de capteurs dans le réseau, à l'aide de requêtes SQL. Nous proposons également un algorithme de traitement des requêtes au niveau de chaque capteur permettant de contrôler la validité temporelle des données et le délai d'exécution des requêtes. En conséquence, nous réduisons la quantité de données circulant sur le réseau, ce qui contribue à maintenir la cohérence temporelle des données dans le RCSF.

5.3 Travaux connexes

Dans la littérature, de nombreux chercheurs se sont intéressés au problème de la cohérence temporelle des données dans les RCSF. Certains se sont concentrés sur le délai de transmission des données. *Saifullah et al.* [Saifullah *et al.*, 2011] ont proposé une nouvelle méthode pour l'analyse du retard dans un réseau *WirelessHART* [Chen *et al.*, 2010]. Leur méthode est basée sur la planification de la transmission des données en temps réel entre les capteurs et les actionneurs. Elle est capable de calculer et de définir de manière efficace les limites supérieures de la communication de bout en bout dans un réseau *WirelessHART*, ce qui a permis de réduire les délais de transmission des données.

Surachai *et al.* [Suriyachai *et al.*, 2010] ont discuté de la livraison des données dans un RCSF industriel en temps réel critique. Ils ont proposé un protocole MAC nommé "GinMAC", basé sur la technique TDMA. Ce protocole intègre des mécanismes de routage et de contrôle de la topologie et de la fiabilité. Il possède les caractéristiques suivantes : (i) Un dimensionnement hors ligne, dont le modèle de trafic et les caractéristiques du canal de transmission sont déterminés avant le déploiement du réseau. De même, les opérations complexes telles que le calcul des dates de transmission sont réalisées hors ligne et avant

le déploiement du réseau. (ii) Le protocole GinMAC utilise la technique d'accès TDMA, dans laquelle le réseau ne peut pas contenir plus de 25 nœuds de capteurs, car un trop grand nombre du nœuds peut créer des interférences. (iii) Le protocole GinMAC doit respecter les délais de transmission et les échéances. Il doit aussi garantir la transmission du plus grand nombre de données fiables, avec une consommation minimale d'énergie. Les auteurs ont montré que GinMAC est en mesure de bien gérer les retards de transmission, la fiabilité des données et la consommation d'énergie pour les systèmes avec des délais de transmission des données à temps critique.

Sharaf *et al.* [Sharaf *et al.*, 2003] ont présenté une nouvelle approche nommée "Tina", qui exploite la cohérence temporelle des données de capteur, afin de réduire le nombre de messages transmis par chaque nœud à la station de base. Les auteurs proposent une nouvelle clause SQL pour exprimer le pourcentage de cohérence temporelle à accepter, appelée "TCT" en anglais (Tolerance Coherency Temporal). Chaque capteur reçoit une requête. Il calcule la cohérence temporelle des données, en se basant sur les nouvelles lectures V_{new} et les anciennes lectures V_{old}. Ensuite, il compare la cohérence temporelle à la valeur de TCT dans la requête, fixée par l'utilisateur. La nouvelle valeur de la donnée est transmise si $\frac{|V_{new}-V_{old}|}{V_{new}} > tct$, sinon elle est supprimée. La requête 12 consiste à obtenir la somme des lumières pour les chambres, regroupées par étage, avec une valeur de TCT = 10%, pour une période de 30 secondes. Une nouvelle lecture ne sera pas transmise si la différence entre celle-ci et l'ancienne valeur n'est pas inférieure à la valeur associée de TCT.

Requête 12.

```
SELECT FLOOR, SUM(LIGHT)
FROM SENSORS
GROUP BY FLOOR
EPOCH DURATION 30s
VALUES WITHIN 10%
```

Lum et al. [Tan *et al.*, 2006] ont proposé un ensemble de mécanismes pour assurer une livraison des données dans un temps critique, afin d'améliorer la fraîcheur des données dans les RCSF. Le premier mécanisme est basé sur la transmission des données par ordre de priorité. Les données à haute priorité sont sélectionnées pour être transmises en premier. Le second mécanisme priorise les nœuds qui possèdent des données de haute priorité pour accéder au support de transmission sans fil. Le troisième mécanisme réserve le support de transmission sans fil pour toute la durée de transmission des données à haute priorité. Les résultats des simulations ont montré que le second mécanisme est meilleur que le premier, en matière de taux de livraison des données et de délais de transmission de bout en bout

pour les données à haute priorité. Le deuxième mécanisme est capable de livrer 100% des données de haute priorité avec un taux de retard acceptable et avec une faible latence, même avec une forte communication dans le réseau.

Choi et al. [Choi *et al.*, 2006] ont proposé un nouvel algorithme de contrôle du temps d'agrégation, nommé "ATC" (Aggregation Time Control), pour respecter les contraintes temporelles de transmission des données dans le RCSF. Leur algorithme permet de contrôler le temps d'agrégation des données au niveau des nœuds, afin d'assurer une livraison des données sans dépasser leurs échéances. Les auteurs ont utilisé deux paramètres pour évaluer leur algorithme : (i) le gain d'agrégation pour mesurer la réduction du trafic, (ii) et le taux d'échec pour mesurer le pourcentage de paquets qui n'ont pas respecté les échéances. Les résultats des simulations ont montré que l'algorithme ATC garantit le respect des contraintes temporelles de transmission des données. Par contre, il n'a pas réussi à réduire le nombre de transmissions.

Pinto et al. [Pinto *et al.*, 2013] proposent un nouveau mécanisme temps réel pour l'estimation du retard de bout en bout dans un RCSF. Leur mécanisme combine les retards des trajets et les délais de traitement des nœuds, ainsi que deux nouvelles métriques de routage RPL : PathDelayMetric qui représente les retards cumulatifs des chemins et Processing Delay Metric qui représente les retards cumulés de traitement. Les auteurs ont comparé leur solution avec la solution ETT (Expected Transmission Time) [Draves *et al.*, 2004], qui estime le nombre de transmissions et de retransmissions nécessaires pour acheminer un paquet sur un lien tout en considérant la charge du réseau. Les résultats des simulations ont montré que la proposition des auteurs est plus précise par rapport à la solution ETT. En outre, elle ne dégrade pas de manière significative les performances du réseau.

Dans la section suivante, nous présentons le modèle de réseau, le nouveau modèle pour les données temporelles et le langage de requête basé sur les nouvelles clauses SQL que nous proposons. Ensuite, nous donnons un aperçu sur les conditions nécessaires avant le déploiement du réseau (connectivité et synchronisation) et nous expliquons le fonctionnement de notre mécanisme de traitement des requêtes.

5.4 Modèles et nouvelles clauses SQL

5.4.1 Modèle de réseau dynamique basé sur l'auto-organisation

Notre modèle de réseau représente un arbre de routage basé sur la latence, dans le quelle la station de base constitue le nœud racine. Le reste du réseau est composé d'une structure hiérarchique basée sur des relations parent-enfant entre les nœuds. Les nœuds "puits" représentent les parents et les nœuds émetteurs représentent les enfants. La sélection des

nœuds ”puits” est basée sur la latence de communication. Cette méthode est conçue pour permettre à chaque nœud de déterminer efficacement le chemin vers la station base avec un temps minimal de transmission. Chaque nœud dans le réseau sélectionne un nœud parent (nœud ”puits”), qui assure la diffusion des requêtes ”utilisateur”, ainsi que la collecte des réponses des capteurs (cf. Figure 5.1). Nous utilisons le protocole de routage RPL [Winter *et al.*, 2012] pour la construction de l'arbre de routage avec une fonction objective basée sur la latence. RPL s'adapte aux caractéristiques des réseaux à faible puissance et avec perte (LLNs[2]). De plus, il forme une topologie arborescente, qui correspond à la structure de notre modèle de réseau.

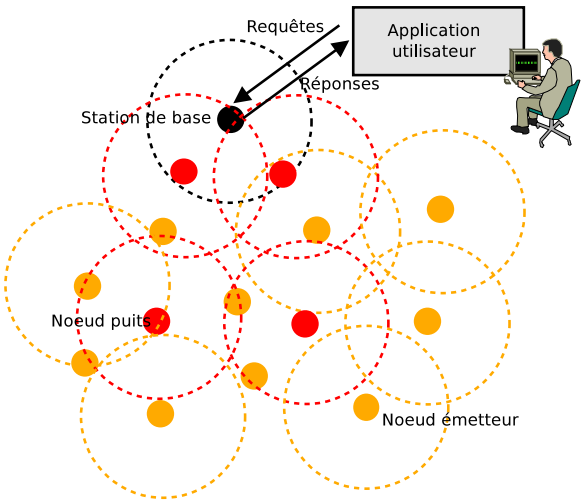


Figure 5.1: Modèle de réseau dynamique basé sur l'auto-organisation

5.4.1.1 Présentation du protocole RPL

Le groupe de travail ROLL[3] de l'IETF[4] a défini le protocole de routage RPL[5] en 2008 [Olivier Hersent, 2014] pour s'adapter aux caractéristiques des réseaux maillés tels que les réseaux de capteurs sans fil. Chaque nœud dans le réseau sélectionne un parent préféré pour construire l'arbre et pour assurer le routage dans le graphe DODAG (*Destination Oriented Directed Acyclic Graph*). Il utilise une fonction objective (OF) qui combine des métriques et des contraintes pour calculer la valeur de son rang et pour sélectionner le meilleur chemin vers la destination. Par exemple, ”trouver le meilleur chemin en fonction de la latence (métrique), tout en évitant le routage à travers les nœuds alimentés par batterie (contrainte)”. Vasseur *et al.* [Vasseur *et al.*, 2012] ont défini des métriques qui concernent les liens de routage telles que le débit, la latence, le niveau de qualité du

[2]LLNs: Low power and Lossy Networks
[3]Routing Over Low power and Lossy networks
[4]Internet Engineering Task Force
[5]Routing Protocol for Low power and lossy networks

lien, le taux de retransmission (ETX) ou des métriques qui concernent le capteur telles que le nombre de sauts, l'énergie du nœud ou l'état du nœud. Les nœuds utilisent aussi des messages de contrôle ICMPv6 [Winter *et al.*, 2012] et le principe de "rang" pour sélectionner les parents préférés et pour maintenir la connectivité et la stabilité de la topologie du réseau. RPL définit quatre types de messages de contrôle :

- **Message d'information (DIO[6])** : il permet de transporter des informations aux nœuds qui veulent rejoindre le réseau. Il leur permet de choisir leurs parents ou de maintenir leur affiliation existante à un DODAG.
- **Message de sollicitation d'information DODAG (DIS[7])** : pour rejoindre le réseau, un nœud a besoin d'un message DIO, qui contient les informations sur le DODAG. Il diffuse un message DIS pour solliciter la réception du message DIO de la part de ses voisins. Sinon, il peut attendre la réception du message DIO. Il permet aussi de découvrir rapidement la structure de l'arbre de routage, les nouveaux itinéraires et d'explorer son voisinage.
- **Objet d'annonce de destination (DAO[8])** : ces messages sont utilisés pour propager les informations de destination le long de l'arbre (DODAG), suivant la direction vers le haut pour remplir les tables de routage des nœuds. Chaque nœud, qui rejoint le réseau, envoie un message DAO contenant ses adresses et ses préfixes aux parents pour annoncer sa présence.
- **Objet d'acquittement de DAO (DAO_ACK[9])** : c'est un message d'acquittement envoyé par un nœud parent à son fils, en réponse à un message DAO reçu, pour confirmer la bonne réception des DAO et pour maintenir la construction des routes descendantes.

5.4.1.2 Construction de l'arbre de routage

Le processus de construction de l'arbre de routage (DODAG) commence au niveau du nœud racine (cf. Figure 5.2) [Olivier Hersent, 2014]. Le nœud racine diffuse un message d'information DIO à ses voisins. Ce message inclut la fonction objective (FO) et la valeur de son rang. Le nœud racine possède le rang le plus petit puisqu'il représente le premier nœud dans l'arbre de routage. En effet, le rang d'un nœud correspond à son emplacement dans l'arbre de routage par rapport à la racine. Sa valeur augmente en descendant dans le graphe. Lorsqu'un nœud veut rejoindre l'arbre de routage, il diffuse un message DIS pour solliciter le réception d'un message DIO qui contient les informations qui lui permettent

[6]DODAG Information Object
[7]DODAG information solicitation message
[8]Destination Advertisement Object
[9]Destination Advertisement Object Acknowledgement

de découvrir le graphe. Généralement, les nœuds diffusent des messages DIO selon une période définie par un temporisateur basé sur l'algorithme Trickle [Levis *et al.*, 2011].

Chaque nœud dans le graphe calcule le coût des chemins avec ses voisins sur la base de la métrique de routage sélectionnée, ce qui lui permet de quantifier la propriété d'un chemin de bout en bout. Le calcul du coût du chemin dépend de la métrique de routage définie dans la fonction objective. Par exemple, pour calculer le coût du chemin en se basant sur le taux de retransmissions (ETX), il suffit de faire la somme de la fiabilité attendue des liens traversés. Pour calculer le coût du chemin, en se basant sur l'énergie, il faut faire la somme de l'énergie de tous les nœuds le long du chemin de la source vers la destination. Ensuite, la sélection du chemin est basée sur la fonction objective qui maximise ou qui minimise le coût du chemin vers le nœud "puits".

Lorsqu'un nœud reçoit les messages DIO, il calcule son rang en utilisant la fonction objective spécifiée dans le message. Le rang d'un nœud ne correspond pas au coût du chemin, bien que sa valeur puisse être dérivée et influencée par les métriques de routage. Le nœud construit par la suite une liste de successeurs possibles parmi les nœuds qui ont envoyé les messages DIO. Il choisit le nœud qui a le plus petit rang dans sa liste de successeurs comme nœud parent préféré. S'il s'agit d'un nœud routeur, il met à jour la valeur de la métrique de routage, en calculant le coût du chemin du nœud parent jusqu'à la racine. Ensuite, il calcule son propre rang "Rank(N)" en se basant sur le rang de son père "Rank (P)" et la valeur de "RankIncrease". Ensuite, il diffuse son propre DIO. La valeur de "RankIncrease" est calculée sur la base de la somme des coûts du chemin (Step) et la valeur de "MinHopRankIncrease" qui représente l'augmentation minimale du rang entre le nœud et l'un de ses parents. Le "MinHopRankIncrease" crée un compromis entre les coûts et le nombre maximal de sauts. Chaque nœud répète les mêmes opérations jusqu'à ce que tous les nœuds reçoivent le message DIO et choisissent leurs nœuds parents.

$$Rank(N) = Rank(P) + RankIncrease \tag{5.1}$$

$$RankIncrease = Step + MinHopRankIncrease \tag{5.2}$$

5.4.2 Métrique de routage RPL basée sur la latence

Les métriques de routage implémentées actuellement sous RPL sont basées soit sur l'énergie du nœud, soit sur le taux de retransmissions (ETX). Par contre, il n'existe pas d'implémentation qui prenne en considération la latence. Notre contribution consiste à implémenter la métrique de routage "latence" sous contikiRPL, afin de construire un arbre de routage basé sur le minimum de temps de transmission entre les nœuds. Nous avons utilisé le

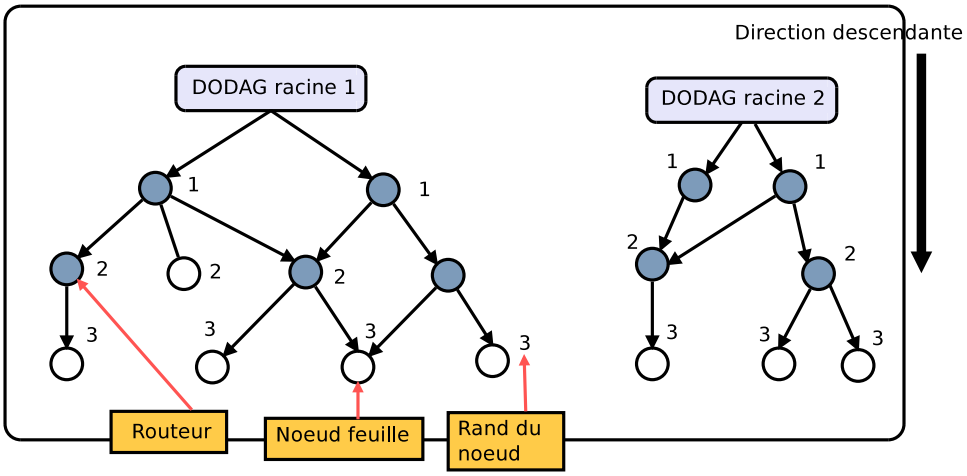


Figure 5.2: Construction de l'arbre de routage (DODAG)

champ "Reserved" du message DIO original (cf. Figure 5.3) pour que chaque nœud puisse ajouter la date de diffusion du message DIO à ses voisins. Ce champ correspond au champ "Sending time" dans le message DIO modifié (cf. Figure 5.4). Le message DIO est composé des champs suivants (cf. Tableau 5.1) :

Champs	Description
RPLInstanceID	L'identifiant de l'instance RPL. Tous les DODAG dans un réseau RPL partagent le même identifiant que l'instance RPL.
Version Number	Le numéro de version de DODAG, qui aide à maintenir les nœuds à la même version DODAG, à chaque mise à jour d'informations sur le réseau.
Rank	Indique le rang du nœud émetteur dans le message DIO.
Grounded (G)	Indique si le DODAG peut satisfaire l'objectif défini par l'application.
MOP	Le mode de fonctionnement de l'instance RPL (sans routage descendant, sans stockage, avec stockage sans multicast, avec stockage avec multicast).
Prf	Le degré de préférence de la racine du DODAG courant par rapport aux autres racines DODAG dans le réseau.
DTSN	L'annonce de destination utilisée pour maintenir les routes vers le bas de l'arbre.
DODAGID	L'identifiant de DODAG.
Flags	Il est réservé pour les flags et initialisé à zéro par l'expéditeur.
Reserved	Il est non utilisé et initialisé à zéro par l'expéditeur.
Sending time	La date de diffusion dans le message DIO modifié.

Tableau 5.1: Les champs d'un message DIO

Avant la diffusion du message DIO, le nœud émetteur inclut la date d'envoi dans le champ "sending time". Ensuite, le nœud récepteur la récupère, pour calculer la latence avec le nœud émetteur.

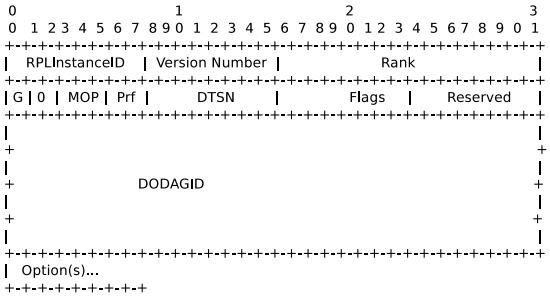


Figure 5.3: Structure de base d'un message DIO

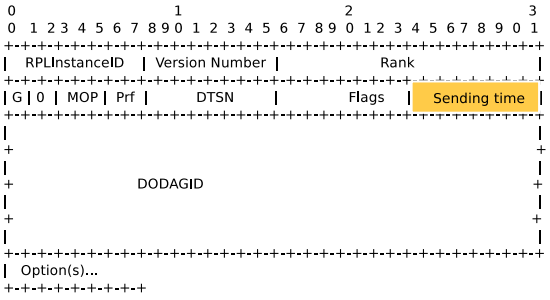


Figure 5.4: Structure modifiée d'un message DIO

Le message DIO peut inclure plusieurs options [Winter *et al.*, 2012] [Vasseur *et al.*, 2012]. Nous avons ajouté la métrique latence dans l'option "DAG Metric Container" dans le message DIO (cf. Figure 5.6). L'option "DAG Metric Container" est utilisée pour signaler la métrique de routage le long du DODAG.

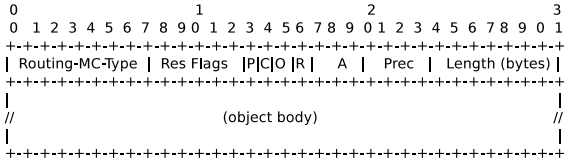


Figure 5.5: Structure de l'option "DAG Metric Container"

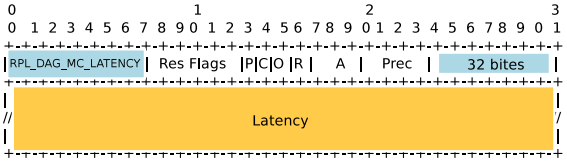


Figure 5.6: Structure de l'option "DAG Metric Container" modifiée

Chaque nœud reçoit le message DIO, il extrait la date d'envoi du message, puis calcule la valeur de la latence, qui représente la différence entre la date d'arrivée du message DIO et sa date d'envoi. Ensuite, le nœud met à jour la valeur de latence dans l'option "DAG Metric Container", et calcule son rang, basé sur le rang de son père et la valeur de la latence. Ensuite, il diffuse le message DIO avec la nouvelle valeur de la latence (cf. Figure 5.7).

$$Rank(current_Node) = Rank(Parent_node) + RankIncrease \tag{5.3}$$

$$RankIncrease = Latency + MinHopRankIncrease \tag{5.4}$$

$$Latency = current_{time} - DIO_{sending-time} \tag{5.5}$$

Une fois que le réseau a convergé et que les routes entre les différents nœuds et la station de base sont bien définies, nous appelons des méthodes permettant d'effectuer quelques vérifications de base avant de recevoir la requête de l'utilisateur.

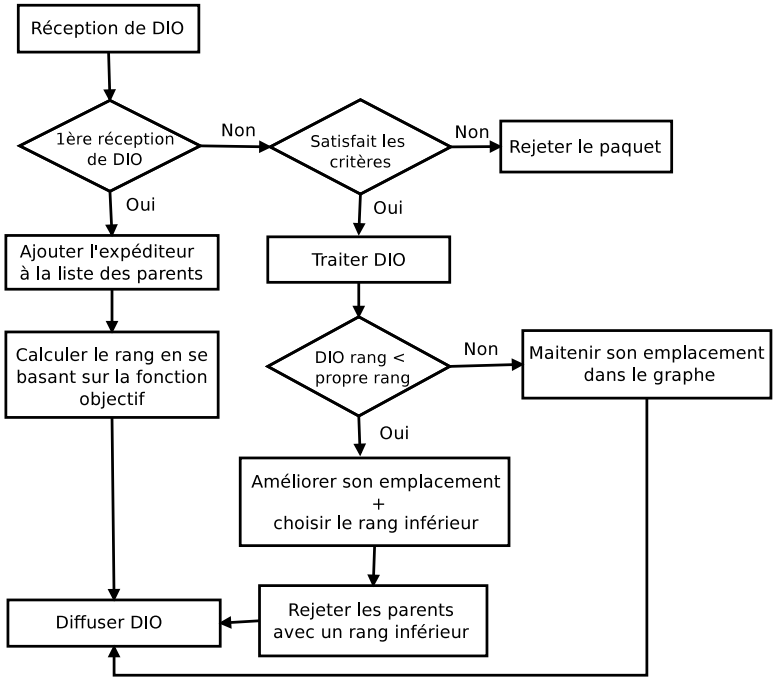


Figure 5.7: Diagramme d'activité d'un routeur dans un DODAG avec RPL

5.4.2.1 Vérification de la connectivité du réseau

Le maintien de la connectivité du réseau assure une communication fiable entre la station de base et les différents nœuds du réseau. Cela permet au protocole de routage de définir et de construire facilement des itinéraires vers la station de base. Il peut également garantir que les données sont en mesure d'atteindre la destination sans dépasser leur validité temporelle. Cependant, lorsque le rayon de transmission de la station de base ne couvre pas tous les nœuds, il devient difficile de maintenir la connectivité dans le réseau. Nous définissons donc des nœuds relais (nœuds puits) (cf. sous section 5.4.1), pour assurer la communication dans les deux directions entre la station de base et les autres nœuds (cf. Figure 5.8). Chaque nœud "puits" forme, avec ses nœuds fils, des groupes (clusters). Dans ce cas, le nœud parent est un chef de groupe. De cette manière, les nœuds "puits" peuvent diffuser les requêtes reçues de la station de base vers les membres du groupe. Ils peuvent également envoyer les réponses des membres du groupe vers la station de base.

5.4.2.2 Vérification de la synchronisation de réseau

Chaque capteur dans le réseau possède sa propre horloge. Toutefois, il peut perdre sa synchronisation avec les autres capteurs pour les raisons suivantes : (1) le démarrage des capteurs ne se fait pas au même instant. Il existe quelques nœuds qui peuvent être démarrés plus tard que les autres, et certains qui peuvent être ajoutés au réseau, (2) les capacités

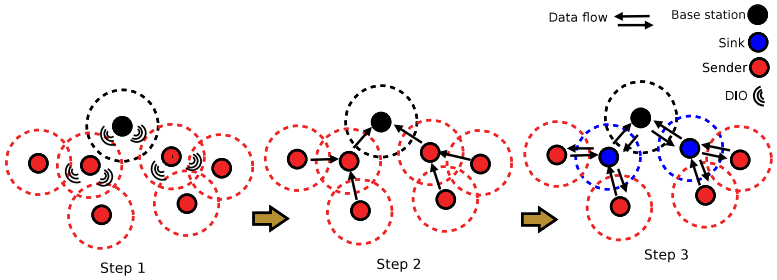


Figure 5.8: Maintien de la connectivité du réseau

matérielles limitées d'un capteur qui ne lui permet pas de maintenir une synchronisation contenue, et (3) la grande densité du réseau qui peut provoquer des interférences entre les capteurs et des pertes de communication. Tous ces défis peuvent affecter, bien évidemment, la synchronisation des nœuds. Dans les systèmes distribués tels que les RCSF, les requêtes finales exigent des réponses dans des délais de communication stricts. Par exemple, le calcul de la température moyenne d'une zone spécifique dépend des réponses de tous les capteurs de cette zone. La réponse finale ne sera pas précise si des réponses tardives sont données. Les protocoles actuels de synchronisation sont basés sur trois modes de synchronisation [Noh *et al.*, 2008] : (1) une synchronisation Émetteur-Récepteur, (2) une synchronisation Récepteur-Récepteur, et (3) une synchronisation basée seulement sur le récepteur. Dans notre modèle, nous avons appliqué une approche de synchronisation Émetteur-Récepteur (cf. Figure 5.9).

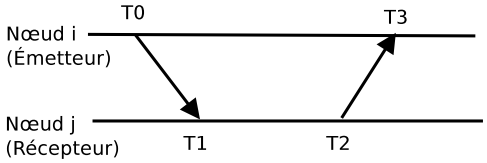


Figure 5.9: Modèle de synchronisation émetteur-récepteur

Dans la figure 5.9, le nœud (i) envoie un message de synchronisation au nœud (j). Ce message contient l'horodatage d'envoi T0. Le nœud (j) reçoit le message à l'instant T1. Ensuite, il calcule la différence entre les deux horodatages T1 et T0, qui représente le premier indicateur de décalage d'horloge entre les nœuds (i) et (j). Le nœud (j) intègre la valeur de décalage d'horloge dans un message, qu'il envoie par la suite au nœud (i), en intégrant aussi T2, l'horodatage d'envoi du message.
Le nœud (i) calcule le deuxième indicateur de décalage d'horloge, entre T3, l'horodatage de réception de messages et la valeur d'horodatage T2. Ensuite, il calcule le décalage d'horloge globale comme suit : $offset = [(T1 - T0) + (T2 - T3)]/2$. Le nœud (i) envoie le décalage d'horloge au nœud (j). Chaque nœud effectue cette procédure avec son nœud parent, jusqu' à ce que tous les nœuds soient synchronisés avec la station de base.

5.4.3 Modèle de données temporelles

Nous avons conçu un nouveau modèle pour les données temporelles, basé sur un modèle de données temps réel introduit par *Ramamritham* [Ramamritham, 1993]. Son modèle est le suivant : $d = (d_{value}, d_{avi}, d_{estampille})$, où d_{value} est la valeur de la donnée extraite, d_{avi} est son intervalle de validité absolue et avec $d_{estampille}$ l'instant de la dernière extraction à partir de l'environnement. Nous avons ajouté de nouveaux attributs pour le modèle de base, afin de répondre aux exigences de notre mécanisme de traitement des requêtes. Ce sont les attributs suivants : d_{source} qui permet de définir la source des données, d_{epoch} pour définir la période pendant laquelle le capteur peut envoyer ses données et d_{age} qui définit l'âge des données. Nous obtenons le modèle suivant : $d = (d_{estampille}, d_{source}, d_{value}, d_{epoch}, d_{avi}, d_{age})$.

5.4.4 Langage de requêtes et nouvelles clauses SQL

Nous utilisons un langage de requête proche du langage SQL, qui permet aux utilisateurs d'exprimer des requêtes, de demander des données à partir du réseau, de filtrer et d'organiser les données sous forme de tuples. Il fournit également de nouvelles clauses SQL. La forme générale de la requête "utilisateur" est présentée dans la requête 13.

Requête 13 (Forme générale de la requête utilisateur)**.**

```
SELECT attributes
FROM Sensor
WHERE conditions
EPOCH duration
DATA.AVI value
DEADLINE value
```

La clause <SELECT> définit les données qui seront récupérées à partir du réseau. La clause <FROM> indique la source des données. La clause <WHERE> spécifie les conditions sur les données demandées. La clause <EPOCH> définit le cycle d'interrogation des données. La clause <DATA.AVI> définit l'intervalle de validité absolue des données à collecter. La clause <DEADLINE> définit la date d'échéance d'une requête.

5.4.5 Méthode de propagation des requêtes

Nous utilisons une technique simple pour la transmission des requêtes "utilisateur" vers les nœuds, basée sur la méthode de diffusion. Lorsque la station de base reçoit la requête

de l'utilisateur, elle ajoute sa date d'arrivée, puis, elle la diffuse vers les nœuds "puits" (sink). Chaque nœud "puits" reçoit la requête, et la diffuse aux membres du groupe (ses fils), qui répondent par les valeurs demandées. Chaque nœud "puits" continue à envoyer la requête pour les membres de son groupe, tant qu'il n'obtient pas une réponse de l'un d'entre eux. L'avantage de notre modèle est que même si la portée de transmission de la station de base ne couvre pas tout le réseau, les nœuds "puits" assureront la transmission des requêtes pour tous les nœuds dans le réseau (cf. Figure 5.1).

5.4.6 Modèle de traitement des requêtes

Une fois le réseau déployé, la connectivité et la synchronisation seront établies entre les différents nœuds du réseau. La station de base diffuse la requête reçue de l'utilisateur vers le RCSF. Nous avons installé un programme dans chaque capteur pour vérifier et contrôler la validité temporelle des données, ainsi que les échéances des requêtes (cf. Figure 5.10).

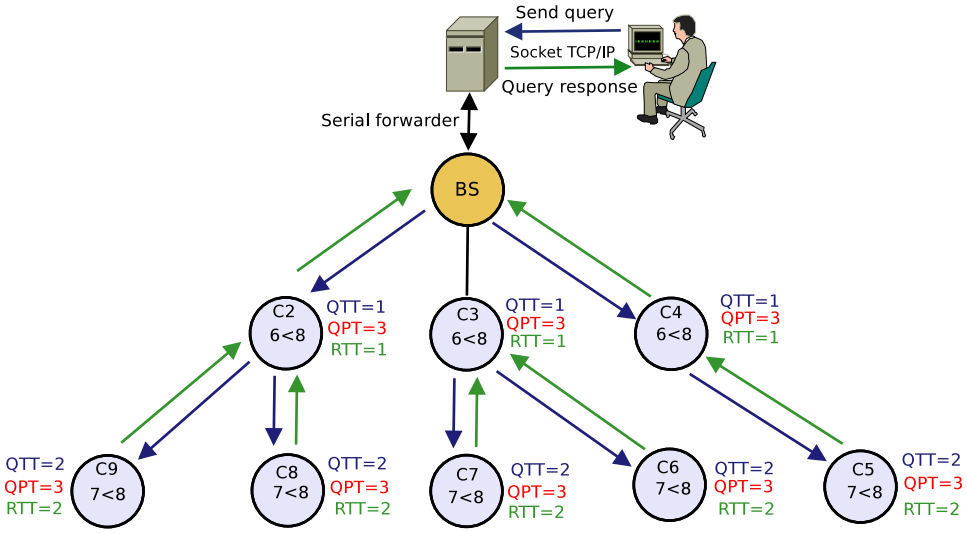


Figure 5.10: Modèle de traitement des requêtes

Lorsqu'un nœud reçoit une requête, il construit un message de réponse, qui contient les données demandées par l'utilisateur, tout en respectant les conditions de la clause <WHERE>. Le message de réponse est composé des attributs suivants : l'identifiant de la source des données, la valeur des données, l'estampille, le cycle d'interrogation des données (epoch), l'intervalle de validité absolue des données et l'âge des données. La figure 5.11 représente les différents éléments d'un message de réponse. Par exemple, la requête dans la figure 5.11 consiste à obtenir l'identifiant du nœud et la température de tous les capteurs dont la durée de validité absolue pour la température est inférieure ou égal à 60 secondes.

Parfois, le processus de traitement d'une requête prend beaucoup de temps, ce qui influe

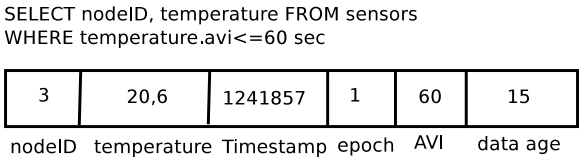


Figure 5.11: Structure d'un message de réponse

sur la validité temporelle des données. Si la date courante est inférieure à l'intervalle de validité absolue, le nœud envoie la réponse à son parent (le nœud "puits"). Dans le cas contraire, le nœud supprime les données extraites et attend la prochaine période. En même temps, il vérifie le temps de transmission de la requête (QTT) et le temps de traitement (QPT). Avant d'envoyer la réponse, chaque nœud considère que le temps estimé de transmission de la réponse (RTT) est égal au temps mis pour recevoir la requête. Si la somme de ces trois temps (QTT, QPT et RTT) ne dépasse pas l'intervalle de validité absolue défini dans la requête, le nœud envoie la réponse à son parent. Lorsqu'un nœud parent reçoit la réponse de son fils, il vérifie d'abord qu'il n'a pas dépassé la durée de validité des données, puis il envoie les résultats à la station de base (cf. Algorithme 2).

Dans l'algorithme 2, la fonction *tcpip_handler*() assure le traitement de la requête. Elle crée et renvoie aussi le message de réponse à la station de base. La valeur "`period`", dans l'algorithme, définit l'intervalle d'envoi de la réponse pour le nœud.

5.5 Simulation et analyse des performances

5.5.1 Environnement de simulation

Nous avons utilisé le simulateur de réseau Cooja de ContikiOS [Österlind *et al.*, 2006], pour évaluer notre modèle de traitement des requêtes. Le simulateur Cooja fournit différents modèles de capteurs tels que Tmote Sky [Corporation, 2006] et MicaZ [Ali *et al.*, 2011]. Nous avons utilisé deux modèles de réseaux. Le premier modèle est statique. Il est composé de 13 nœuds (une station de base, 3 nœuds "puits" et 9 nœuds émetteurs). Les nœuds "puits" et les nœuds émetteurs sont fixés manuellement. Chaque nœud "puits" possède trois fils. La portée de transmission de la station de base couvre les nœuds "puits", et la portée de transmission des nœuds "puits" couvre les nœuds émetteurs (cf. Figure 5.23). Le deuxième modèle est dynamique. Il est basé sur un programme de sélection, qui fixe les nœuds "puits" et leurs fils. Le tableau 5.2 présente les paramètres de simulation utilisés dans nos expériences.

Nous avons utilisé dans nos expériences trois modèles de requêtes Q1, Q2 et Q3, ayant respectivement les tailles 39, 67 et 85 octets. La première requête est basique :

Algorithm 2: Vérification de la durée de validité des données

```
Input: Packetbuf (buffer des données qui contient la requête "utilisateur")
Input: QTT (temps de transmission de la requête)
Input: QPT (temps de traitement de la requête)
Input: QRT (temps de transmission de la réponse)
Input: Query (requête utilisateur)
Input: Period (intervalle d'envoi d'un nœud)
Input: AVI (intervalle de validité absolue d'une donnée)
Initialization: QTT=QPT=QRT=0;
while true do
    if new data = Query then
        tcpip_handler();
    end
    if etimer_expired(Period) then
        T1←clock_time();
        T2← get_timestamp(Packetbuf);
        QTT ← T1-T2;
        AVI ← get_AVI(Query);
        if QTT<AVI then
            params ← get_param(Query);
            reponse ← get_values(Params);
            QPT ← get_QPT(Query);
            RTT ← QTT;
            if QTT+QPT+RTT<AVI then
                send(response);
            else
                delete(response);
            end
        end
    end
end
```

Paramètres	Modèle statique	Modèle dynamique
Modèle de capteur	Tmote sky	Tmote sky
Zone de simulation	100m*100m	100m*100m
Nombre de nœuds	13 nœuds	10 à 40 nœuds
Portée de transmission (Tx)	30m	20m à 40m
Taille maximale d'un paquet	128 bytes	128 bytes
Vitesse de transmission	250 kbps	250 kbps
Couche MAC	CSMA/CA	CSMA/CA
Couche radio (duty-cycle)	ContikiMac	ContikiMac
Modèle de propagation radio	UDGM[10]	UDGM
Function Objective RPL	Énergie	Énergie et Latence

Tableau 5.2: Paramètres de simulation

Q1 : "SELECT idnode, temperature FROM sensors".

Le deuxième requête contient une clause de validité temporelle des données :

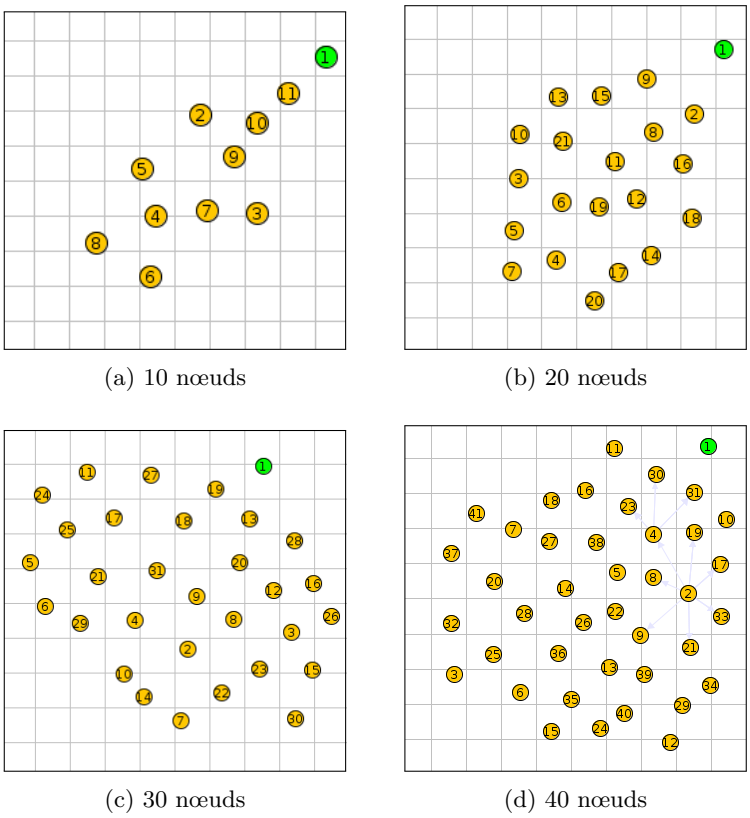
Q2 : "SELECT idnode, temperature FROM sensors WHERE temperature.AVI<= Value (ms)".

La troisième requête contient une clause de validité temporelle des données et une clause qui définit l'échéance de la requête :

Q3 : "SELECT idnode, temperature FROM sensors WHERE temperature.AVI <= Value (ms) AND DEADLINE <= Value (ms)".

5.5.2 Analyse de performances du modèle de réseau dynamique

Dans cette sous-section, nous analysons les performances du modèle de réseau dynamique sur notre modèle de traitement des requêtes et sa capacité à maintenir la cohérence temporelle des données. Nous avons fait varier le nombre de nœuds entre 10 et 40 (cf. Figure 5.12), et nous avons utilisé des portées de transmission qui varient entre 20 et 50 mètres pour chacune des expériences réalisées.
Nous analysons l'influence de la portée de transmission sur le nombre de nœuds "puits" et sur le nombre de sauts. Ensuite, nous comparons les performances basées sur la métrique de routage "latence" par rapport au temps de réponse des requêtes avec d'autre métriques de routage (énergie et taux de retransmission (ETX)). Nous discutons par la suite les performances du modèle dynamique avec la métrique de routage "latence" par rapport au nombre de données valides et aux taux de transactions réussies.



(a) 10 nœuds (b) 20 nœuds

(c) 30 nœuds (d) 40 nœuds

Figure 5.12: Topologies utilisées avec un modèle de réseau dynamique

5.5.2.1 Influence de la portée de transmission sur le nombre de nœuds "puits" et sur le nombre de sauts

Le tableau 5.3 montre que le nombre de nœuds "puits" diminue lorsque la portée de transmission du nœud augmente. Lorsque la portée de transmission d'un nœud est grande, certains nœuds peuvent transmettre leurs données directement à la station de base avec un minimum de sauts. Dans le cas contraire, où la portée de la transmission d'un nœud est faible, notre modèle de propagation définit un nombre de nœuds "puits", permettant aux données d'atteindre la station de base. L'augmentation du nombre de nœuds "puits" dans un réseau multi-saut permet de minimiser la charge de travail au niveau des nœuds "puits", ce qui améliore le temps de réponse dans le réseau.

La figure 5.13 illustre le rapport entre le nombre de nœuds "puits" et le nombre total de nœuds. Nous remarquons que ce rapport diminue avec l'augmentation de la portée de transmission des nœuds. Nous remarquons aussi que ce rapport est lié au nombre de nœuds. Par exemple : avec une portée de transmission de 20 mètres, la moyenne des nœuds "puits" est plus grande que celle avec une portée de transmission de 50 mètres. L'augmentation du nombre de nœuds améliore le temps de transmission des données et

	20 m	30 m	40 m	50 m	60 m
10	5	5	2	3	1
20	10	5	4	2	1
30	18	9	8	4	3
40	21	12	6	4	3

Tableau 5.3: Nombre de nœuds ”puits”

	20 m	30 m	40 m	50 m	60 m
10	6	4	3	3	2
20	7	4	4	3	2
30	6	5	3	3	3
40	7	5	4	4	3

Tableau 5.4: Nombre maximale de sauts

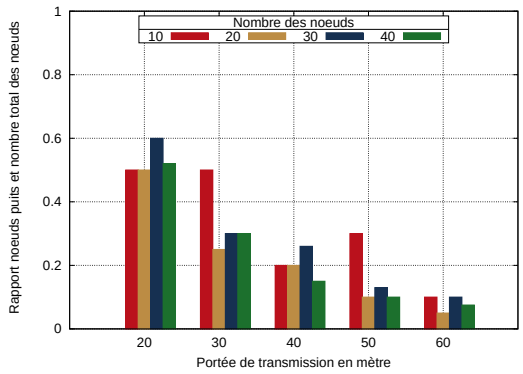


Figure 5.13: Nombre de nœuds puits

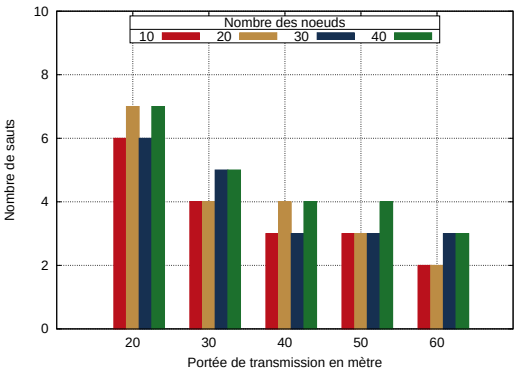


Figure 5.14: Nombre de sauts

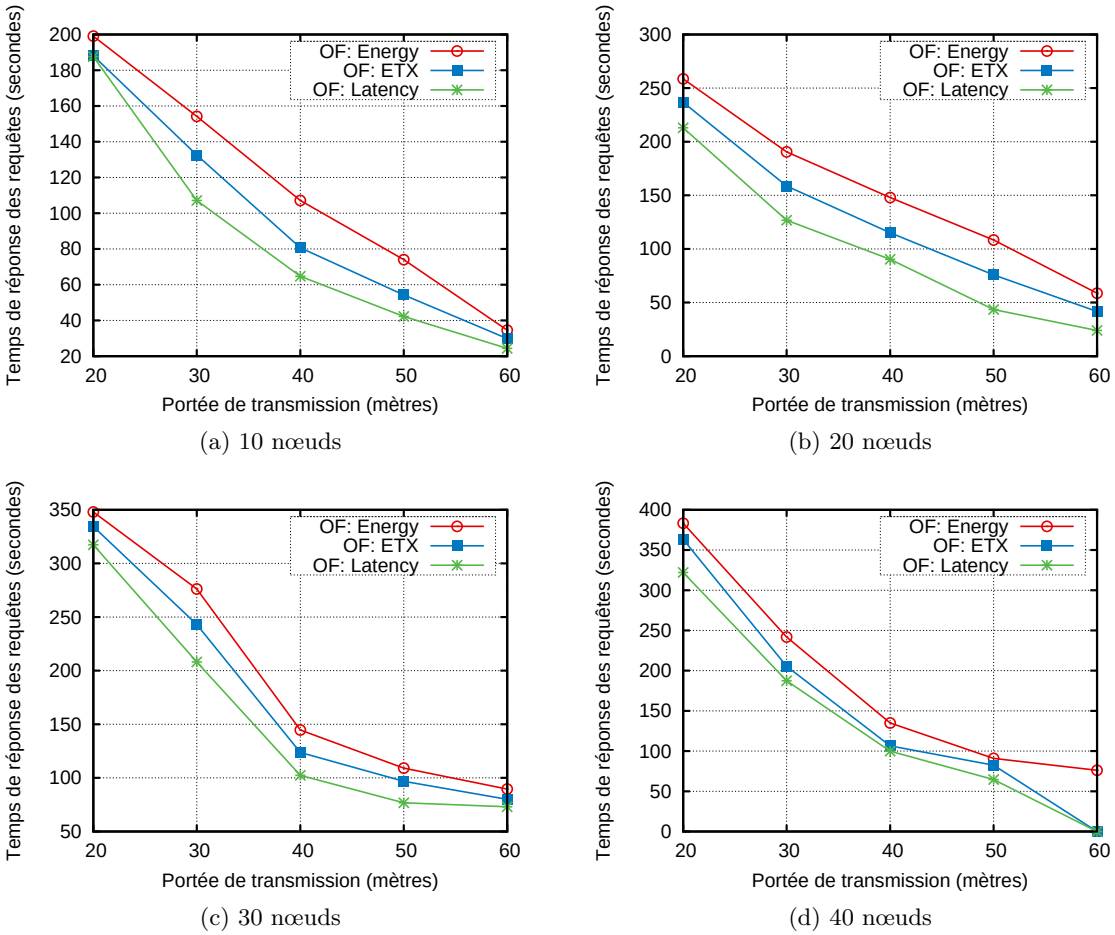
réduit la charge de travail dans le réseau.

La figure 5.14 montre le nombre de sauts effectués en faisant varier le nombre de nœuds ainsi que la portée de transmission. On remarque que le nombre de sauts diminue avec l'augmentation de la portée de transmission. On note aussi que ce nombre augmente avec l'augmentation du nombre de nœuds (40 nœuds). Le nombre de sauts dépend de la position des nœuds (cf. Figure 5.12) et du protocole de routage qui définit le prochain saut pour chaque nœud.

5.5.2.2 Temps de réponse des requêtes

Dans cette expérience, nous avons testé les performances de notre modèle de réseau dynamique avec trois métriques de routage (énergie, latence, taux de retransmission (ETX)) sous RPL, afin de déterminer quelle est la métrique de routage qui donne le meilleur temps de réponse.

Nous remarquons dans les figures 5.15a, 5.15b, 5.15c, 5.15d que l'utilisation de la métrique ”latence” donne le meilleur temps de réponse par rapport aux autres métriques de routage. On remarque aussi que le temps de réponse avec la métrique ”énergie” est meilleur que le temps de réponse avec la métrique ”taux de retransmission” (ETX). Le temps de réponse augmente avec la taille du réseau et diminue avec l'augmentation de la portée de transmission. Nous expliquons ces résultats par le choix effectué par le protocole de routage



(a) 10 nœuds

(b) 20 nœuds

(c) 30 nœuds

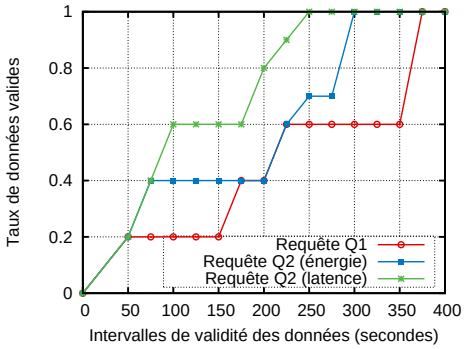
(d) 40 nœuds

Figure 5.15: Temps de réponse des requêtes avec les métriques de routage RPL

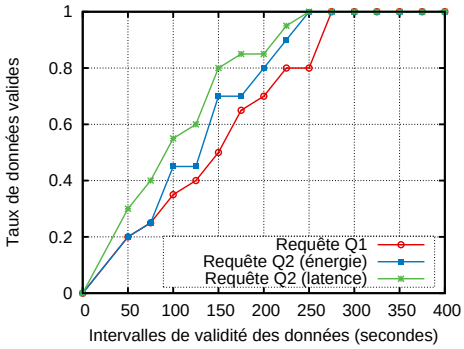
dans la sélection des chemins vers la station de base avec un coût minimal pour la latence.

5.5.2.3 Portée de transmission et nombre de données valides

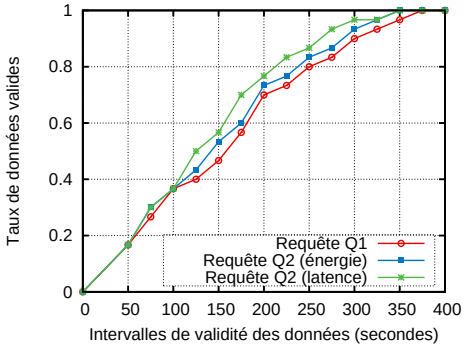
Nous étudions dans cette expérience l'influence de la portée de transmission des nœuds sur le nombre de données valides. Nous avons fixé quatre simulations avec un nombre de nœuds qui varie entre 10 et 40 et une portée de transmission qui varie entre 20 mètres et 40 mètres. Nous avons testé notre modèle de requêtes avec deux métriques de routage (énergie et latence) pour le protocole RPL.



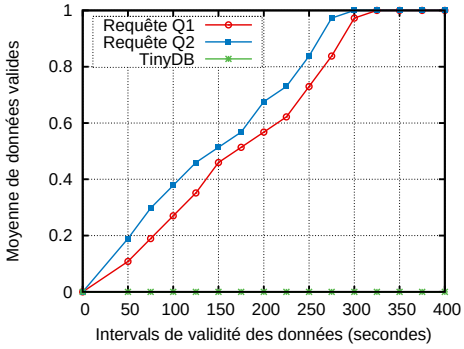
(a) 10 nœuds



(b) 20 nœuds

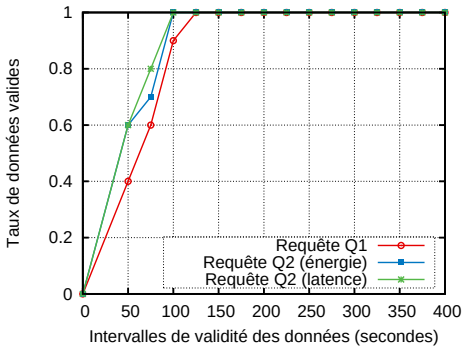


(c) 30 nœuds

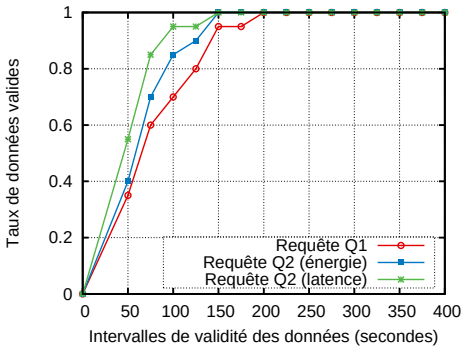


(d) 40 nœuds

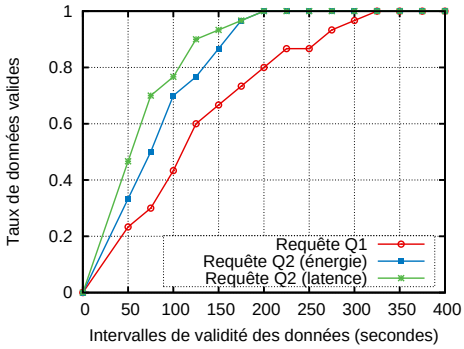
Figure 5.16: Nombre de données valides avec une portée de transmission de 20m



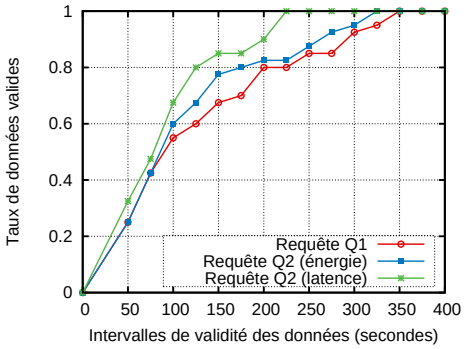
(a) 10 nœuds



(b) 20 nœuds

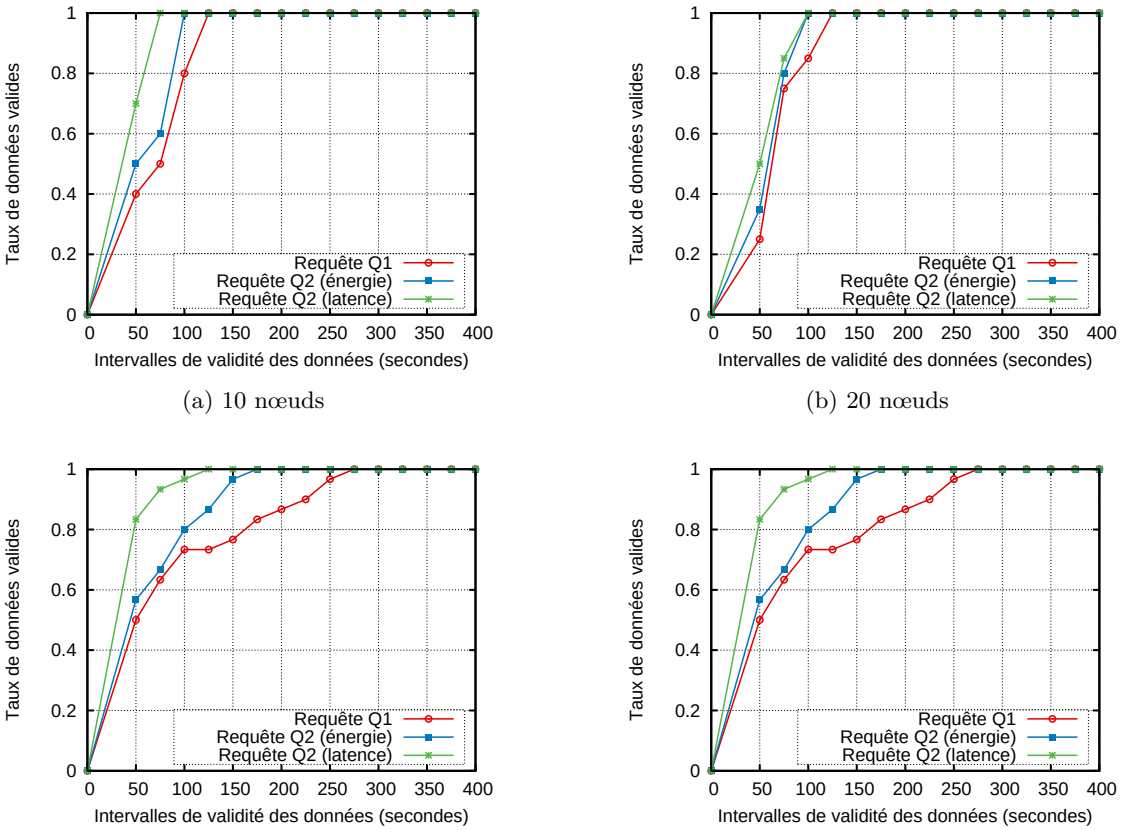


(c) 30 nœuds



(d) 40 nœuds

Figure 5.17: Nombre de données valides avec une portée de transmission de 30m



(a) 10 nœuds

(b) 20 nœuds

(c) 30 nœuds

(d) 40 nœuds

Figure 5.18: Nombre de données valides avec une portée de transmission de 40m

Dans chaque simulation, nous avons augmenté les échéances relatives des données en commençant par 50 secondes avec une augmentation de 25 secondes à chaque fois jusqu'à ce qu'on aboutisse à 100% de données valides. À travers cette expérience, nous pouvons déterminer l'échéance relative d'une transaction, qui s'adapte au nombre de nœuds et à la portée de transmission. À chaque fois, nous utilisons une seule diffusion. Par la suite, la station de base récupère les réponses des nœuds. Les résultats des simulations sont présentés dans les figures 5.16, 5.17 et 5.18 qui montrent que la requête Q2 donne plus de données temporellement valides que la requête Q1, surtout en utilisant la métrique de routage "latence", que par une métrique de routage basée sur l'énergie. On peut expliquer les performances de la requête Q2 par l'ajout d'un mécanisme de contrôle des données qui permet de filtrer les données transmises, en évitant les données temporellement non valides. Ceci a permis aussi de réduire les congestions entre les nœuds et de réduire la charge de travail au niveau des nœuds "puits". Il a aussi amélioré le taux de données valides reçues par la station de base. L'utilisation de la latence comme métrique de routage a amélioré le temps de transmission de bout en bout. De plus, elle a amélioré le taux de données valides.

Nous remarquons dans les figures 5.16, 5.17 et 5.18 qu' à chaque fois qu'on augmente le nombre de nœuds, le temps de réponse des nœuds augmente aussi. En conséquence, l'intervalle de validité des données s'étend. Le taux de données qui ont respecté l'échéance, en utilisant une métrique de routage "latence" est plus élevé que celui avec une métrique de routage "énergie". Nous observons aussi que l'augmentation de la portée de transmission des nœuds a amélioré le taux de données qui ont respecté l'échéance, car les nœuds peuvent envoyer les réponses avec un nombre minimal de sauts vers la destination. Ceci réduit le nombre de nœuds "puits" et augmente le nombre de nœuds fils pour chaque nœud "puits". En conséquence, la charge de travail au niveau des nœuds va augmenter à cause du nombre élevé de réponses reçues. Dans cette situation, il est important d'utiliser un mécanisme d'agrégation au niveau des nœuds "puits".

5.5.2.4 Analyse des performances du modèle dynamique avec des requêtes périodiques

Dans cette expérience, nous comparons les trois approches "One-One", "Half-Half" et "More-Less", conçues pour maintenir la validité temporelle des données avec notre modèle de traitement de requêtes. Nous avons fait varier le nombre de nœuds entre 10 et 40 par pas de 10. Chaque nœud utilise une portée de transmission égale à 40 mètres. Le temps de réponse des nœuds varie selon le nombre et la position des nœuds dans le réseau. Les nœuds situés près de la station de base, possèdent un temps de réponse inférieur à celui des nœuds situés loin de la station de base. La durée de validité des données varie d'un système à un autre, selon la nature du système temps réel (avec des contraintes temporelles strictes, souples ou molles). Nous avons testé plusieurs collectes de données pour fixer le temps de réponse des nœuds. Pour chaque expérience, nous avons défini une durée de validité temporelle des données égale à deux fois le temps total de collecte des données dans chaque expérience. Le tableau 5.5 résume les temps de réponse trouvés et les durées de validité définies pour le reste des expériences.

Nb. nœuds	Temps de réponse	Moy. des TR. d'un nœud	Durées de validité des données
10	200 sec.	20 sec.	400 sec.
20	300 sec.	30 sec.	600 sec.
30	400 sec.	40 sec.	800 sec.
40	400 sec.	40 sec.	800 sec.

Tableau 5.5: Valeurs des intervalles de validité temporelles

Nous avons proposé une règle (cf. Équation 5.6), qui définit le rapport entre le temps de réponse des requêtes et la durée de validité des données. Nous avons utilisé les symboles de tableau 5.6 dans cette équation.

$$\sum_{i=1}^{n} \frac{T1_i - T0_i}{n} \leq \alpha_i \tag{5.6}$$

Dans l'Équation 5.6, nous considérons que le temps de réponse d'une requête est égal à la différence entre la date d'arrivée des données à la station de base et la date de leur extraction au niveau des capteurs. La somme des temps de réponses des nœuds doit être inférieure à la somme des durées de validité de toutes les données envoyées.

Symboles	Descriptions
Tr_i	La transaction.
Pi	La période de la transaction
Di	L'échéance relative de la transaction
di	L'échéance absolue de la transaction
Ri	Le temps de réponse de la transaction
$\boldsymbol{\alpha_i}$	La durée de validité des données x_i
n	Le nombre de données.
$T0_i$	La date d'extraction de la donnée x_i.
$T1_i$	La date d'arrivée de la donnée x_i à la station de base.

Tableau 5.6: Définitions de symboles

Nous utilisons des transactions de capteurs périodiques avec des échéances strictes. Chaque transaction doit être exécutée une fois par période. Cependant, il n'y a aucune garantie sur la date d'exécution d'une transaction, ni sur la date d'arrivée des données à la station de base. Nous supposons également que les transactions sont préemptables et que la période et l'échéance relative (Di) d'une transaction de capteur sont fixées au départ par l'utilisateur à travers dans la requête à envoyer.
`Di = Échéance absolue (di) - Temps d'échantillonnage`.

Nous avons testé les trois approches "One-One", "Half-Half" et "More-Less" avec notre modèle de traitement de requêtes, et nous avons fixé les valeurs de la période et de l'échéance relative suivant le principe de chaque approche (cf. Tableaux 5.7, 5.8, 5.9, 5.10, 5.11).

Approche	Di	Pi
One-One	α_i	α_i
Half-Half	$\alpha_i/2$	$\alpha_i/2$
More-less	$\alpha_i/3$	$2\alpha_i/3$

Tableau 5.7: Valeurs de périodes et d'échéances

- **Approche "One-One" :** Di=Pi= α_i, avec T1 - T0 $\leq \alpha_i$.
  Dans l'approche "One-One", on considère que la période et l'échéance relative de la transaction sont égales à la durée de validité des données.

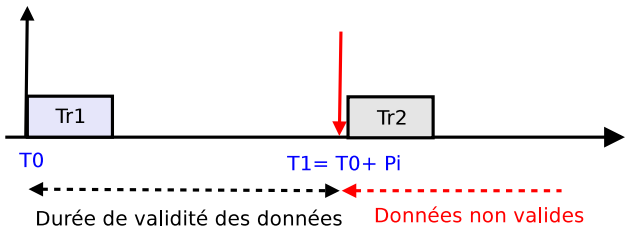


Figure 5.19: Approche One-One

- **Approche "Half-Half" :** Di$\leq \frac{\alpha_i}{2}$, Pi $\leq \frac{\alpha_i}{2}$, avec T1 - T0 $\leq \alpha_i$.
  Dans l'approche "Half-Half", on considère que la période et l'échéance relative de la transaction sont égales à la moitié de la durée de validité des données.

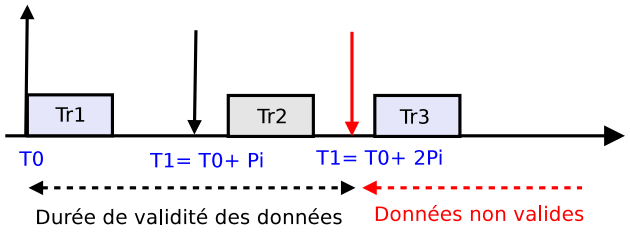


Figure 5.20: Approche Half-Half

- **Approche "More-Less" :** Di$\leq \frac{\alpha_i}{3}$, Pi $\leq \frac{2\alpha_i}{3}$, avec T1 - T0 $\leq \alpha_i$.
  Dans l'approche "More-Less", on considère que la période est inférieure ou égale au tiers de la durée de validité des données, et l'échéance relative de la transaction est inférieure ou égale aux deux tiers de la durée de validité des données, avec la contrainte de validité : $P_i + D_i \leq \alpha_i$.

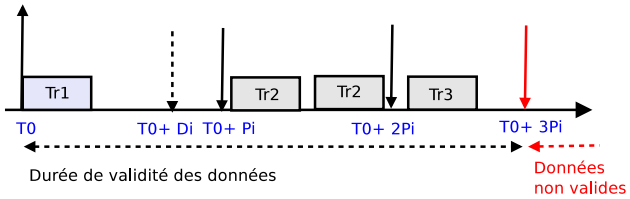


Figure 5.21: Approche More-Less

Approches	Di	Pi	α_i
One-One	400 sec.	400 sec.	400 sec.
Half-Half	200 sec.	200 sec.	400 sec.
More-Less	133 sec.	267 sec.	400 sec.

Tableau 5.8: Intervalles temporels avec 10 nœuds

Approches	Di	Pi	α_i
One-One	600 sec.	600 sec.	600 sec.
Half-Half	300 sec.	300 sec.	600 sec.
More-Less	200 sec.	400 sec.	600 sec.

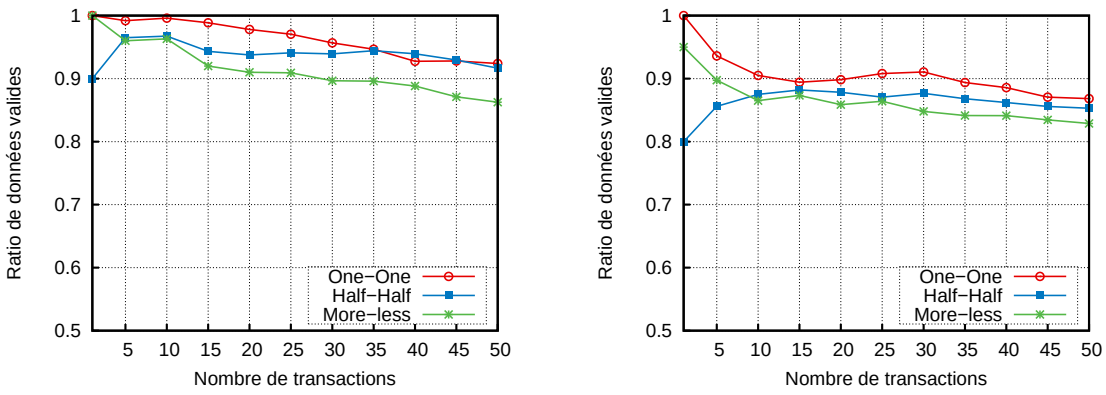
Tableau 5.9: Intervalles temporels avec 20 nœuds

Approches	Di	Pi	α_i
One-One	800 sec.	800 sec.	800 sec.
Half-Half	400 sec.	400 sec.	800 sec.
More-Less	264 sec.	533 sec.	800 sec.

Tableau 5.10: Intervalles temporels avec 30 nœuds

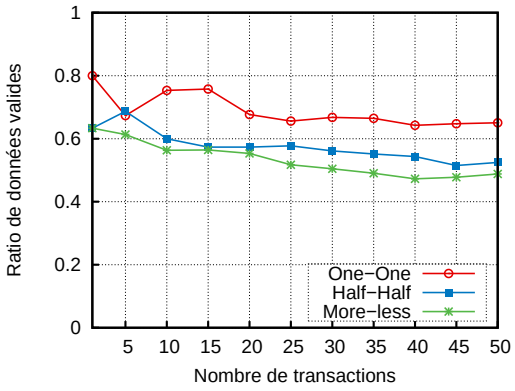
Approches	Di	Pi	α_i
One-One	800 sec.	800 sec.	800 sec.
Half-Half	400 sec.	400 sec.	800 sec.
More-Less	264 sec.	533 sec.	800 sec.

Tableau 5.11: Intervalles temporels avec 40 nœuds

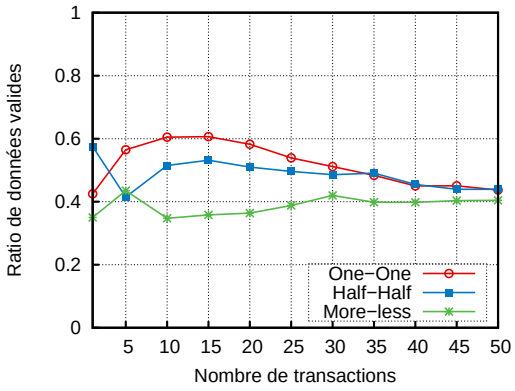


(a) 10 nœuds

(b) 20 nœuds



(c) 30 nœuds



(d) 40 nœuds

Figure 5.22: Comparaison des approches de maintien de la validité des données

Dans les figures 6.11a, 6.11b, 6.11c, 5.22d, nous remarquons que l'approche "One-One" donne le meilleur taux de données qui respectent leurs échéances. Ensuite nous trouvons

l'approche "Half-Half" et enfin l'approche "More-Less". Le taux de données valides diminue avec l'augmentation du nombre de nœuds pour les trois approches, mais il reste supérieur à la moitié des données envoyées à la station de base. Avec l'approche "One-One", l'échéance relative est égale à la validité temporelle des données, ce qui offre plus de temps pour le capteur pour traiter les données et pour les envoyer à la station de base. Cependant, avec l'approche "Half-Half", l'échéance relative est égale à la moitié de la validité temporelle des données, ce qui peut retarder la date d'arrivée des données à la station de base. Pour l'approche "More-Less", l'échéance relative est égale au tiers de la validité temporelle des données, ce qui peut restreindre la capacité du capteur à envoyer sa réponse avant l'échéance.
Nous remarquons également que lorsque nous augmentons le nombre de transactions, le nombre de données valides diminue. En effet, l'augmentation du nombre de transactions dans le réseau augmente la charge de travail au niveau des nœuds "puits". Certaines données restent bloquées dans la file d'attente du nœud "puits". La mise en place d'un système d'ordonnancement au niveau des nœuds "puits", permet de gérer les priorités d'envoi des données. Dans notre système, le blocage des données dans les nœuds "puits" réduit la quantité de données valides reçues par la station de base.

5.5.3 Analyse de performances du modèle de réseau statique

Nous avons testé deux scénarios pour déterminer la capacité du modèle statique de réseau à maintenir la cohérence temporelle des données (cf. Figure 5.23) :

1. Le premier scénario représente une collecte périodique des données. L'utilisateur diffuse une seule requête à tous les capteurs. Ensuite, chaque capteur envoie sa réponse périodiquement, durant l'intervalle de temps défini par l'utilisateur.

2. Le deuxième scénario représente une collecte des données à la demande. L'utilisateur diffuse périodiquement des requêtes à tous les capteurs. Ensuite, chaque capteur envoie la réponse à la station de base, pendant l'intervalle d'envoi défini par l'utilisateur, pour chaque requête reçue.

ContikiOS utilise des bibliothèques de gestion du temps telles que :

- La bibliothèque *timer* : elle fournit plusieurs fonctions de démarrage et de redémarrage des temporisateurs. Elle permet aussi de régler le temporisateur et de vérifier si le temps a expiré.

- La bibliothèque *stimer* : elle fournit un mécanisme de temporisation similaire à la bibliothèque *timer*, mais elle utilise la seconde comme unité de temps.

- La bibliothèque *ctimer* : elle fournit un mécanisme de temporisation qui fait appel

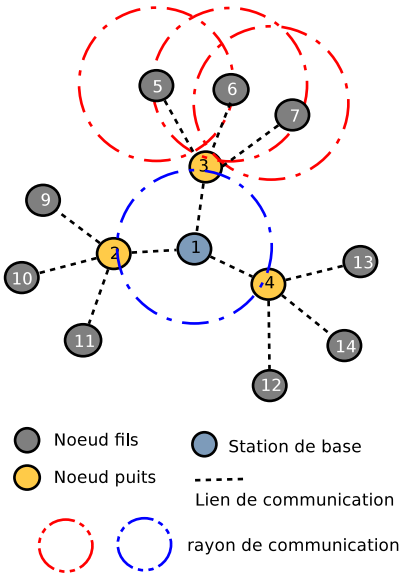


Figure 5.23: Modèle de réseau statique

à une fonction spécifique lorsque le temporisateur arrive à expiration.

- La bibliothèque *etimer* : elle fournit un mécanisme de temporisation qui permet de générer des événements chronométrés.
- La bibliothèque *rtimer* : elle fournit un mécanisme de temporisation qui gère l'ordonnancement et l'exécution des tâches en temps réel.

Nous avons mené trois expériences basées sur les bibliothèques de gestion du temps. La première expérience consiste à mesurer l'influence des intervalles d'envoi de la station de base et des nœuds sur le nombre de réponses aux requêtes. La deuxième expérience consiste à mesurer l'influence des intervalles d'envoi sur le nombre de données valides. La troisième expérience consiste à mesurer l'influence des intervalles d'envoi sur le temps de réponse moyen des requêtes.

5.5.3.1 Influence des intervalles d'envoi sur le nombre de réponses aux requêtes

Nous avons fait varier les intervalles d'envoi des requêtes au niveau de la station de base, ainsi que pour chaque nœud émetteur, afin de trouver le meilleur rapport entre les intervalles d'envoi. Nous avons effectué 100 envois périodiques entre la station de base et les capteurs, avec des requêtes du type Q1, avec des intervalles d'envoi qui varient entre 10 secondes et 70 secondes. Les résultats des simulations sont représentés dans la figure 5.24.

La figure 5.24 montre une augmentation du nombre de réponses aux requêtes lorsque nous

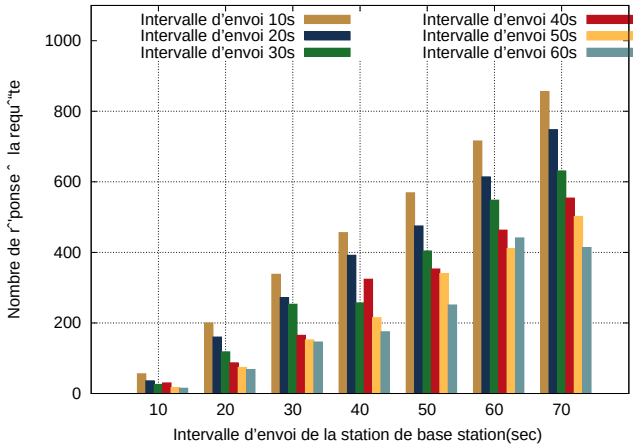


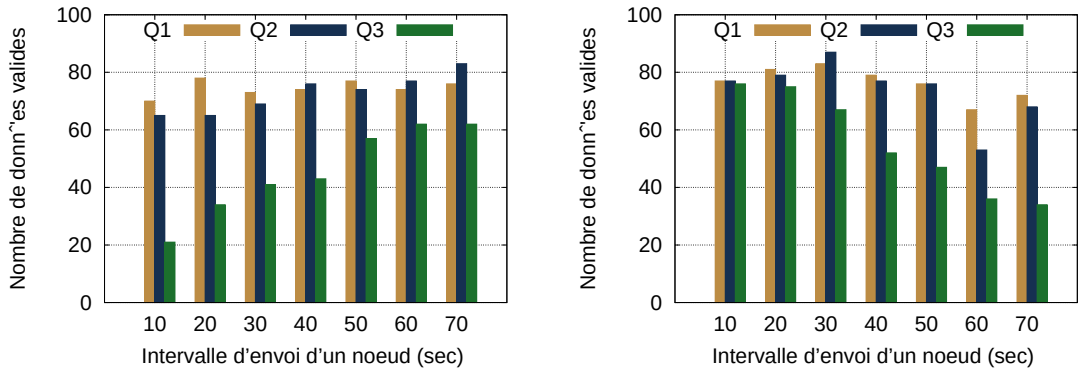
Figure 5.24: Nombre de réponses aux requêtes

réduisons l'intervalle d'envoi des nœuds émetteurs et que nous augmentons l'intervalle d'envoi de la station de base. Les meilleurs résultats sont obtenus avec un intervalle d'envoi égal à 10 secondes pour un nœud émetteur et un intervalle d'envoi égal à 70 secondes pour la station de base. Nous allons considérer ces résultats dans les expériences suivantes, afin d'améliorer le nombre de données valides et le nombre de transactions réussies.

5.5.3.2 Influence des intervalles d'envoi sur le nombre de données valides

L'objectif de cette expérience est d'analyser les effets de la variation de la période d'envoi des nœuds émetteurs et de la station de base sur le nombre de données valides. Nous avons testé les deux scénarios 1 et 2 avec les requêtes Q1, Q2 et Q3, en faisant varier la valeur de l'intervalle d'envoi pour les nœuds émetteurs de 10 secondes à 70 secondes. Pour la requête Q3, nous avons fixé une échéance inférieure ou égale à l'intervalle d'envoi d'un nœud émetteur pour chaque période, afin d'exploiter tout l'intervalle d'envoi du capteur. Dans le scénario 1, l'échéance absolue (di) est égale à dix fois l'échéance relative (Di), tel que Di est égale à l'intervalle d'envoi du capteur. Dans le scénario 2, l'utilisateur envoie dix requêtes successives. Chaque requête possède une échéance relative égale à l'intervalle d'envoi du capteur. Les résultats des simulations sont représentés dans les figures 5.25a et 5.25b.

La figure 5.25b montre que le scénario 2 donne le meilleur nombre de données valides avec un petit intervalle d'envoi pour un nœud émetteur, par rapport au scénario 1, qui donne de meilleurs résultats avec un grand intervalle d'envoi pour un nœud émetteur. Dans les deux cas de figure, le temps de traitement de notre algorithme de contrôle de validité temporelle des données retarde certaines données pour atteindre la station de base avant leur échéance. Cependant, nous considérons, en général, que les résultats sont



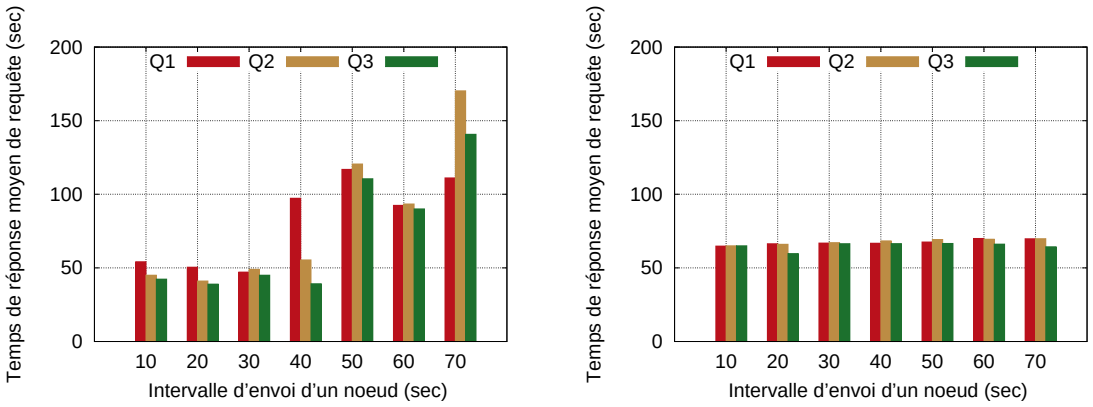
(a) scenario 1 (b) scenario 2

Figure 5.25: Nombre de données valides avec un modèle statique

satisfaisants par rapport au nombre total de données envoyées. Nous pourrons en déduire que les performances de notre mécanisme de traitement des requêtes sont meilleures avec le scénario 2, en nous basant sur la quantité de données valides reçues dans les deux scénarios.

5.5.3.3 Influence des intervalles d'envoi sur le temps de réponse moyen des requêtes

Dans cette expérience, nous étudions le temps de réponse moyen pour les requêtes Q1, Q2 et Q3. Nous avons utilisé les mêmes paramètres pour les intervalles d'envoi pour les nœuds et pour la station de base et le même nombre de requêtes que dans l'expérience précédente. Les résultats de simulation sont présentés dans les figures 5.26a et 5.26b.



(a) scénario 1 (b) scénario 2

Figure 5.26: Temps de réponse moyen aux requêtes

Au travers des résultats trouvés, nous remarquons que la moyenne des temps de réponse

aux requêtes pour le scénario 2 avec les trois requêtes est quasi stable pour les différents intervalles d'envoi pour les nœuds émetteurs. Nous expliquons ces résultats par la quantité de données qui a été réduite par notre algorithme, et par le choix de l'intervalle d'envoi pour la station de base qui est égal à l'intervalle d'envoi à un nœud émetteur. Ceci donne plus de temps à un nœud émetteur pour envoyer ses réponses. Concernant le scénario 1, nous notons que la moyenne du temps de réponse des requêtes augmente avec l'intervalle d'envoi des nœuds émetteurs. Lorsqu'un nœud émetteur dispose d'un grand intervalle d'envoi, ceci lui donne plus de liberté pour choisir la date d'envoi de la réponse. Cependant, si un nœud "puits" reçoit les réponses dans un temps plus court, ceci augmente sa charge de travail et peut affecter le temps de réponse moyen. Nous considérons que notre approche de traitement des requêtes avec un modèle statique de réseau donne la meilleure moyenne des temps de réponses aux requêtes avec le scénario 2.

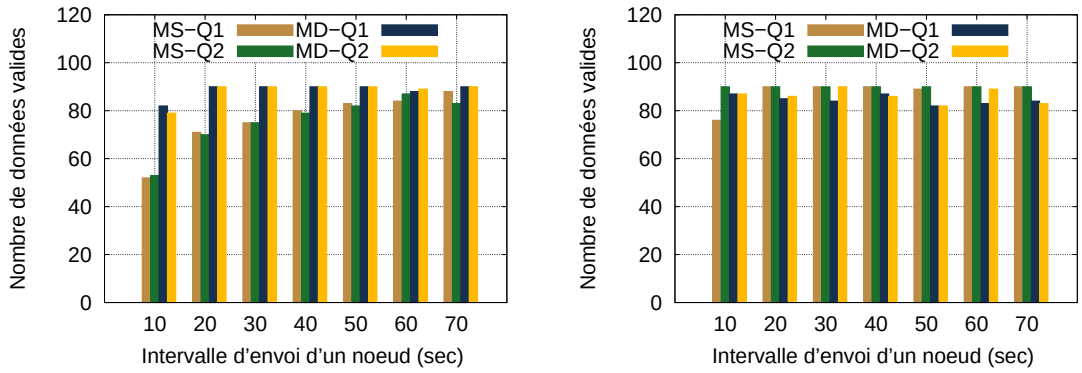
5.5.4 Comparaison du modèle statique et du modèle dynamique

Dans cette section, nous comparons les performances de notre modèle de traitement des requêtes avec les deux modèles réseaux : statique (MS) et dynamique (MD). D'abord, nous discutons de l'influence du choix de l'intervalle d'envoi du capteur sur le taux de données qui respectent les échéances et sur le temps de réponses des requêtes pour les deux modèles. La première expérience consiste à tester les deux modèles en utilisant le premier scénario, qui consiste en une collecte périodique. L'utilisateur envoie une seule requête à travers le réseau. Ensuite chaque capteur envoie sa réponse à la station de base. Nous faisons varier les intervalles d'envoi pour les capteurs et nous définissons une échéance relative pour la collecte des données et un nombre d'envois (n), défini au départ par l'utilisateur :

```
Di = n * intervalle d'envoie du capteur                    (5.7)
```

Cela permet de déterminer le taux de données qui respectent l'échéance, dans les deux modèles. La deuxième expérience consiste à envoyer des requêtes périodiques vers le réseau. Nous utilisons des requêtes de type Q1 et Q2. Les résultats des simulations sont présentés dans les figures 5.27a et 5.27b.

La figure 5.27a montre que le taux de données valides obtenu avec un modèle dynamique en utilisant le scénario 1 est plus élevé que celui obtenu avec un modèle statique. Cependant, les performances du modèle dynamique diminuent par rapport au modèle statique, qui donne le meilleur taux de données valides. Avec un modèle dynamique, la sélection des nœuds "puits" est basée sur la métrique "latence". Ceci améliore le temps de transmission des données entre les nœuds et permet aux données de respecter leur échéance. Dans le



(a) scénario 1 (b) scénario 2

Figure 5.27: Taux de données valides avec un modèle dynamique

scénario 1, la transmission des données de capteurs est périodique. La charge de travail au niveau des nœuds "puits" peut augmenter le traitement des données et retarder leur arrivée à la station de base.

La figure 5.28 montre que le modèle dynamique donne un meilleur temps de réponse par rapport au modèle statique dans les deux scénarios. L'utilisation de la métrique "latence" avec le protocole RPL a permis d'améliorer le temps de réponse. Nous remarquons aussi que le scénario 1 est meilleur que le scénario 2. Lorsque l'utilisateur interroge périodiquement le RCSF, ceci créé deux flux de données au niveau des nœuds "puits". Le premier flux de données concerne l'arrivée des requêtes et le deuxième flux concerne l'arrivée massive des réponses des capteurs. Le temps de traitement des données va augmenter le temps de réponse global des requêtes. Un algorithme d'ordonnancement au niveau des nœuds présente une solution pour gérer les transactions "utilisateur", ainsi qu'un contrôleur de similarité, qui permet de réduire la quantité de données à transmettre vers la station de base.

5.6 Conclusion

Dans ce travail, nous avons proposé un nouveau modèle de traitement des requêtes pour améliorer la cohérence temporelle des données dans le RCSF. Nous avons proposé un nouveau modèle de réseau dynamique basé sur l'auto sélection des nœuds "puits" et qui s'appuie sur la latence de communication entre les capteurs. L'implémentation de cette nouvelle métrique de routage a amélioré le temps de réponse des requêtes, ce qui participe à l'amélioration de la validité temporelle des données. Nous avons proposé également deux nouvelles clauses SQL pour contrôler la validité temporelle des données et respecter les délais de réponse aux requêtes au cours du processus de collecte des données, avec un

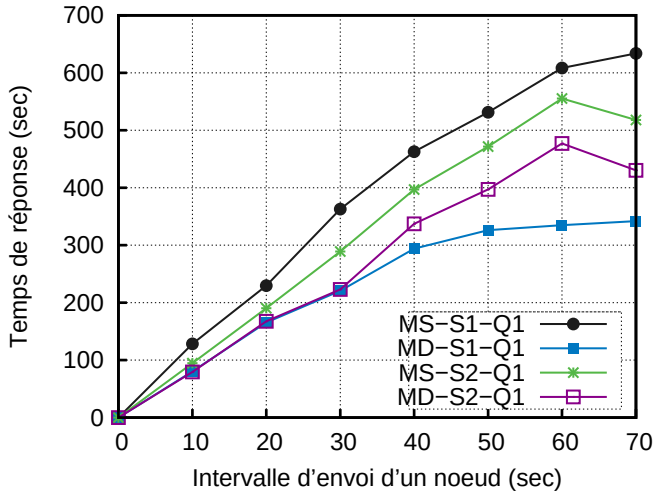


Figure 5.28: Temps de réponses pour les modèles statique et dynamique

algorithme de contrôle de la validité temporelle des données, installé au niveau de chaque nœud. Nous avons testé le modèle de traitement des requêtes avec deux modèles de réseau : un modèle dynamique basé sur l'auto-sélection des nœuds "puits" et un modèle statique dont les nœuds sont fixés manuellement. Les résultats des simulations ont montré que le modèle dynamique est meilleur que le modèle statique pour le nombre de données valides reçues par la station de base, ainsi que pour le temps de réponse aux requêtes. Nous pouvons déduire que l'algorithme "One-One" est le plus adapté à notre modèle de traitement des requêtes car il donne le meilleur taux de données valides par rapport aux autres approches ("MoreLess" et "Half-Half"). L'algorithme de traitement des requêtes a réussi à réduire la quantité de données échangées dans le réseau et à améliorer la quantité de données valides reçues par la station de base. Nous pouvons en conclure aussi que l'intervalle d'envoi des données pour les nœuds ainsi que pour la station de base peut améliorer le temps de réponse des requêtes s'il est bien défini. Nous prévoyons d'améliorer les performances de l'algorithme de contrôle des données en ajoutant des ordonnanceurs d'événements au niveau des nœuds "puits" pour coordonner les envois des requêtes et la réception des réponses.

Chapitre 6

Une base de données temps réel pour les RCSF

Sommaire

6.1 Introduction

Ce chapitre présente un nouveau système de Bases de Données de Capteurs Temps Réel (BDCTR). Il est basé sur une architecture de bases de données de capteurs, qui contient des composants provenant des bases de données temps réel, telles que les ordonnanceurs temps réel et les contrôleurs d'échéances. Nous proposons ce système de BDCTR parce que les architectures de bases de données de capteurs existantes ne sont pas appropriées pour supporter les contraintes temporelles. Elles ne fournissent pas de composants qui peuvent vérifier la validité temporelle des données ou garantir le respect des délais assignés aux transactions. Elles sont plutôt conçues pour réduire le nombre de communications et la consommation d'énergie dans le réseau, afin d'augmenter sa durée de vie. La consommation d'énergie est un facteur important dans les RCSF, mais la cohérence temporelle des données ainsi que les échéances des transactions présentent aussi des facteurs primordiaux pour de nombreuses applications temps réel.
Dans la littérature, les bases de données temps réel ont montré qu'elles peuvent assurer des contraintes temporelles associées aux données et aux transactions. Elles utilisent des mécanismes basés sur des ordonnanceurs temps réel pour attribuer les priorités aux transactions. Elles peuvent également retourner des données temporellement valides grâce aux contrôleurs d'échéances, qui éliminent toutes les données qui ne respectent pas leur validité temporelle. Une base de données temps réel peut être une solution pour garantir le respect des contraintes temporelles dans les RCSF, à condition qu'elle s'adapte aux caractéristiques spécifiques des RCSF.

Nous proposons un système de BDCTR, qui fournit les fonctionnalités suivantes : (i) il assure une diffusion des requêtes et une collecte des données, en utilisant une méthode de sélection automatique des nœuds "puits" (cf. section 5.4.1), basée sur les chemins de routage avec une latence minimale entre les nœuds, (ii) il garantit le respect des contraintes temporelles dans le RCSF à l'aide de contrôleurs d'échéances, qui sont mis en place sur les différents nœuds (nœuds émetteurs, nœuds "puits" et station de base), (iii) il prend en considération une exécution multi-requête, en utilisant des ordonnanceurs temps réel pour attribuer les règles de priorité aux transactions. À notre connaissance, il n'existe pas de bases de données de capteurs sans fil qui fournissent ces fonctionnalités.

Ce chapitre est organisé comme suit : dans la section 2, nous présentons la problématique et les motivations pour le choix d'une base de données temps réel comme solution. Dans la section 3, nous présentons les modèles et l'architecture du système. Dans la section 4, nous donnons les résultats des simulations. Dans la section 5, nous commentons les résultats et nous comparons notre système avec les bases de données de capteurs existantes. Enfin, dans la section 6, nous donnons une conclusion pour ce chapitre.

6.2 Problématique et motivations

Les RCSF sont conçus pour retourner des informations qui reflètent l'état réel de l'environnement où ils sont déployés. Cependant, les capacités limitées des capteurs en termes de stockage des données, de capacités de calcul et de mémoire, etc. et la grande quantité de données échangées entre la station de base et les capteurs peuvent entraîner de longs délais de livraison des données. De plus, la station de base peut recevoir un grand nombre de requêtes provenant de différents utilisateurs en même temps. Par conséquent, la charge de travail de la station de base augmente et les transactions peuvent manquer leurs échéances. Alors, le système devient inefficace, car il ne peut pas accomplir ses fonctions prévues. Gürgen *et al.* ont défini trois types de transactions de capteurs [Gürgen *et al.*, 2006] : (1) les transactions "one-time", qui consistent à interroger le RCSF à un moment précis, pour donner à l'utilisateur une vue instantanée sur une zone surveillée du réseau, (2) des transactions continues, qui consistent à interroger le RCSF périodiquement, selon des intervalles de temps définis par l'utilisateur, (3) les transactions de mise à jour, qui consistent à mettre à jour les attributs des capteurs. Les transactions doivent maintenir les propriété d'atomicité, de cohérence, d'isolation et de durabilité, connues sous l'acronyme de propriétés ACID, pour assurer la fiabilité de la base de données. Cependant, les bases de données de capteurs ne peuvent pas garantir facilement les propriétés ACID, car l'environnement du RCSF est dynamique.

Des travaux antérieurs ont été réalisés afin de garantir le respect des contraintes temporelles des données et des transactions dans les bases de données temps réel [Duvallet *et al.*, 1999]. Dans la littérature, une base de données temps réel est définie comme une base de données dans laquelle les transactions possèdent des échéances [Ramamritham, 1993]. Elles sont utilisées dans des applications temps réel qui nécessitent un accès rapide aux données [Stankovic *et al.*, 1999]. Une base de données temps réel peut supporter l'exécution de transactions concurrentes et assurer la cohérence temporelle des données [Ramamritham, 1993]. En effet, les bases de données temps réel utilisent des algorithmes d'ordonnancement efficaces pour maintenir la cohérence des données et pour garantir les délais d'exécution des transactions. C'est pourquoi, nous considérons qu'un système de bases de données de capteurs temps réel peut garantir les contraintes temporelles des RCSF.

Dans le chapitre 5, nous avons proposé un nouveau modèle de traitement des requêtes pour maintenir la cohérence temporelle des données dans les RCSF. Ce modèle fournit un algorithme efficace, qui s'exécute sur chaque capteur, pour vérifier la validité temporelle des données afin de réduire le nombre de données non valides. En outre, nous avons conçu un nouveau modèle de données temporelles pour améliorer les performances de l'algorithme de contrôle des données. Il permet à l'utilisateur d'ajuster l'intervalle temporel de validité

des données et l'échéance d'exécution d'une requête, en utilisant deux nouvelles clauses SQL. Nous avons réussi à améliorer la quantité de données valides reçues par la station de base [Belfkih *et al.*, 2015a]. Cependant, le modèle de traitement des requêtes ne peut pas assurer la coordination entre l'envoi des requêtes et la réception des résultats lorsque le nombre de requêtes augmente.

Dans ce chapitre, nous proposons une architecture de bases de données de capteurs temps réel (BDCTR) pour garantir les contraintes temporelles dans un RCSF. Elle est basée sur l'architecture de bases de données de capteurs et l'architecture de SGBDTR[1]. Nous utilisons des *ordonnanceurs temps réel* et des *contrôleurs d'échéances* pour gérer l'augmentation du nombre de transactions et pour leur attribuer des priorités. Nous avons mis en place un nouveau plan d'exécution des requêtes pour améliorer la diffusion des requêtes "utilisateur" et la collecte des données à partir du RCSF. Nous utilisons aussi le modèle de données temporelles et les clauses SQL proposées dans [Belfkih *et al.*, 2015b]. Dans la section suivante, nous présentons l'architecture de la BDCTR.

6.3 Architecture du système

L'architecture de la BDCTR se compose de quatre éléments : (1) une interface graphique, qui permet à l'utilisateur de saisir ses requêtes et d'afficher les réponses, (2) une station de base pour diffuser les requêtes des utilisateurs dans le réseau. Elle les ordonnance et elle contrôle leurs échéances et la validité temporelle des réponses reçues, (3) les nœuds "puits" pour assurer la transmission des requêtes reçues par les nœuds fils ainsi que la validité temporelle des données. Les nœuds "puits" utilisent des ordonnanceurs temps réel pour prioriser les requêtes en cas de surcharge. Ils agrègent les réponses des capteurs en vérifiant également la similitude des données, pour réduire le nombre de réponses similaires et pour conserver l'énergie du réseau, et enfin, (4) le nœud émetteur qui répond aux requêtes des utilisateurs avec les données demandées. Dans la suite, nous discutons en détails de l'architecture de la BDCTR.

6.3.1 Station de base

Elle est composée des éléments suivants (cf. Figure 6.1) :

1. *Un contrôleur de similarité des requêtes* : il compare la nouvelle requête avec les anciennes requêtes utilisées. Si la requête a été envoyée, la station de base envoie les réponses à partir de l'historique des réponses des requêtes stockées dans la mémoire cache, qui correspondent à la requête de l'utilisateur. Sinon, la nouvelle requête est

[1]System de Gestion de Base de Données Temps Réel

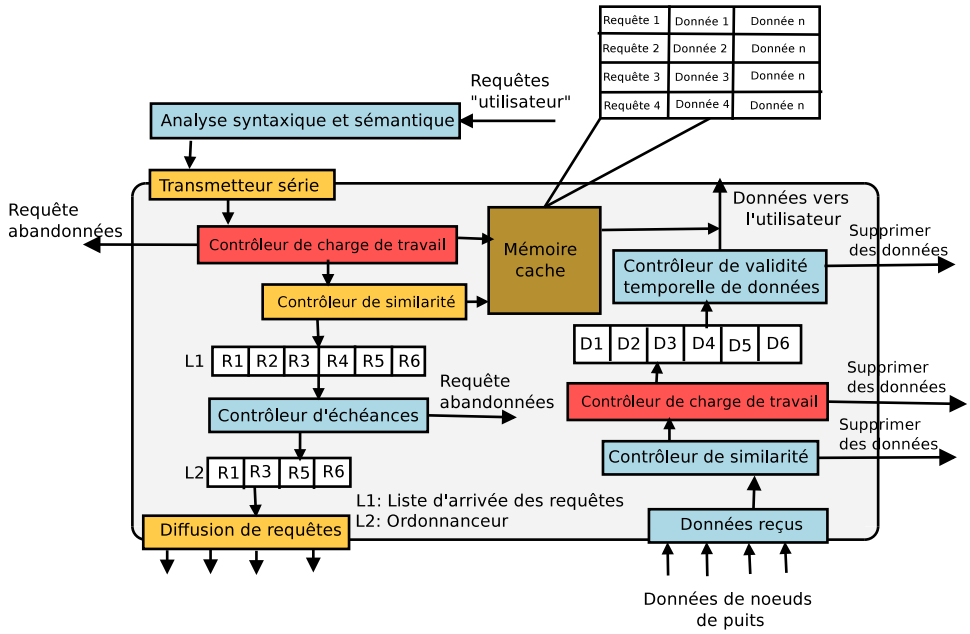


Figure 6.1: Architecture générale de la station de base

envoyée à tous les capteurs. Les réponses des capteurs ont une validité temporelle dans la mémoire cache. Elles sont supprimées périodiquement lorsque leur validité temporelle est terminée.

2. *Une mémoire cache* : elle est utilisée pour conserver temporairement l'historique des réponses des requêtes.

3. *Une liste d'arrivée des requêtes* : elle est utilisée pour recevoir les requêtes des utilisateurs. Elle peut contenir de une à n requêtes.

4. *Un contrôleur d'échéances des transactions* : il est utilisé pour vérifier les échéances des requêtes en se basant sur l'échéance absolue définie dans la requête et sur la date courante. Si l'échéance absolue de la requête est supérieure à la date actuelle alors la requête sera envoyée. Sinon, la requête sera abandonnée.

5. *Un ordonnanceur de transactions* : il est utilisé pour définir la façon dont les transactions doivent être exécutées au cours du temps, afin de garantir leurs échéances. Il existe plusieurs algorithmes d'ordonnancement, qui dépendent de la stratégie mise en œuvre par le système temps réel. La requête qui manque son échéance sera abandonnée. La requête qui ne satisfait pas les conditions de faisabilité sera placée dans la file d'attente. La requête dont l'échéance est la plus proche sera la première à être injectée dans le réseau.

6. *Un contrôleur de validité des données* : il est utilisé pour contrôler la validité temporelle des données. Il se base sur la date de génération des données et sur la date

courante pour supprimer les données qui ont dépassé leur validité temporelle.

7. *Un contrôleur de similarité des données* : il est utilisé pour attribuer des valeurs de similarité pour les données, afin d'éviter les mêmes mises à jour et pour supprimer des données similaires.

8. *Une liste d'arrivées des données* : elle reçoit les données non similaires du contrôleur de similarité, qui seront classées par ordre de priorité dans la liste. Cet ordre dépend de la validité temporelle de données.

6.3.2 Nœud "puits"

Il contient les éléments suivants : (1) une liste d'arrivées des requêtes, (2) un contrôleur d'échéances des transactions, (3) un ordonnanceur des transactions, (4) un contrôleur de validité des données et (5) un contrôleur de similarité (cf. Figure 6.2). Ces éléments possèdent les mêmes fonctionnalités que celles proposées pour la station de base.

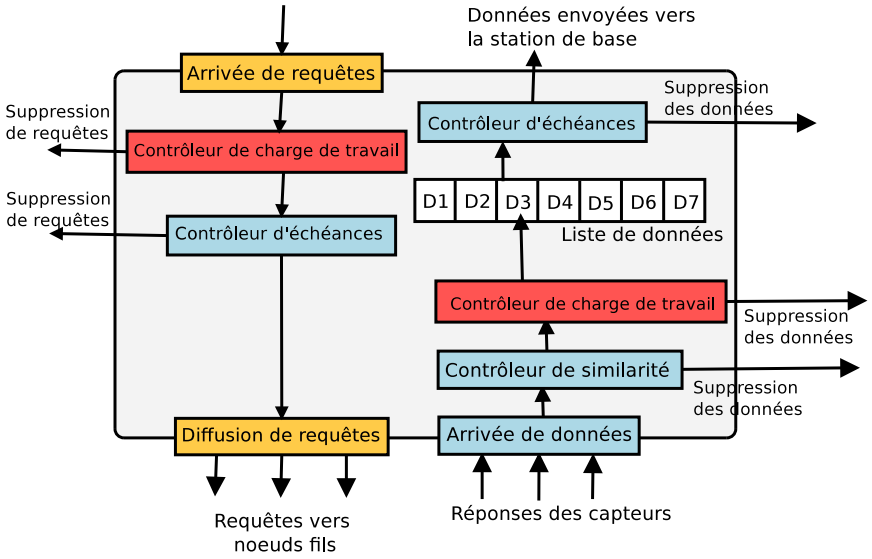


Figure 6.2: Architecture générale du nœud puit

6.3.3 Nœud émetteur

Il se compose des éléments suivants : (1) un contrôleur d'échéance des transactions et (2) un processeur de requêtes. Quand il reçoit une requête du nœud "puits", le contrôleur d'échéances vérifie la validité temporelle de la requête. Si l'échéance n'est pas dépassée, la

requête est exécutée par le processeur de requêtes. Ensuite, le nœud émetteur construit le message de réponse et l'envoie à son nœud parent.

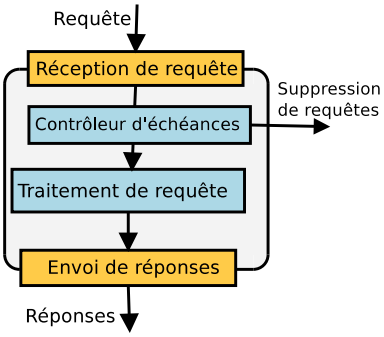


Figure 6.3: Architecture générale du nœud émetteur

6.4 Modèles du système

6.4.1 Modèle de réseau multisauts

On utilise un modèle de réseau multisauts, qui forme une architecture arborescente, composée de plusieurs niveaux. Chaque niveau représente un saut à un autre niveau vers la station de base. La station de base présente la racine de l'arbre. Elle est responsable de la diffusion des requêtes "utilisateur" dans le réseau, et de la collecte des réponses des nœuds de capteurs. Chaque niveau dans l'arborescence est composé de nœuds "puits" (nœuds de relais) et de nœuds émetteurs. Les nœuds "puits" sont sélectionnés par leurs voisins, en se basant sur un programme d'auto-sélection présenté dans le chapitre 5. Les nœuds "puits" sont considérés comme des nœuds parents, qui facilitent la transmission des requêtes entre la station de base et les nœuds enfants (nœuds émetteurs). Ils assurent aussi la collecte des réponses à partir des nœuds enfants pour les transmettre à la station de base. Un nœud "puits" peut être un nœud parent et un nœud enfant pour un autre nœud d'un niveau supérieur.

6.4.2 Modèle de données

Nous utilisons le modèle de données temporelles proposé dans le chapitre 5. Ce modèle est le suivant : d (d_{value}, d_{avi}, $d_{timestamp}$, d_{source}, d_{epoch}, d_{age}), où d_{value} représente la valeur de la donnée extraite du réseau, d_{avi} représente l'intervalle de validité absolue de la donnée, $d_{timestamp}$ indique la date de l'extraction de la donnée de l'environnement, d_{source} définit la source de données, d_{epoch} définit la période pendant laquelle les capteurs

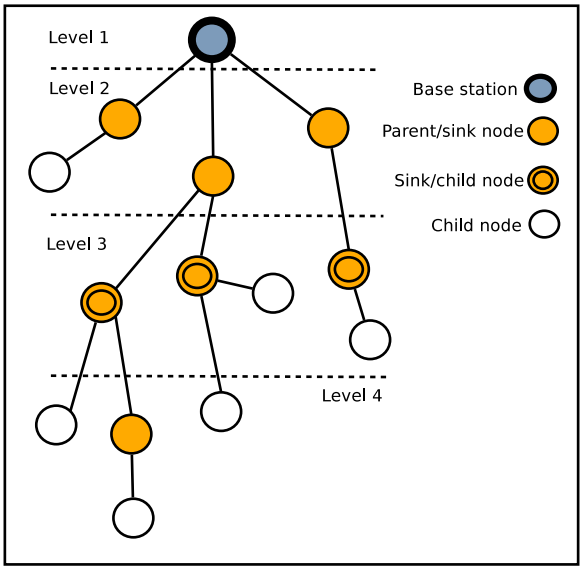


Figure 6.4: Modèle de réseau multisauts

ont envoyé leurs réponses, d_{age} indique l'âge de la donnée et permets à chaque capteur de déterminer si les données sont encore valides. Nous n'avons pas utilisé l'attribut d_{age} dans ce modèle de données, parce que cet attribut peut être calculé en utilisant l'attribut $d_{timestamp}$ et l'horodatage actuel. Ce modèle de données s'adapte au fonctionnement du contrôleur d'échéances, pour contrôler efficacement les contraintes temporelles des données et des transactions.

6.4.3 Langage de requêtes

Notre base de données de capteurs temps réel utilise le langage de requêtes "SQL-Like", proposé dans le chapitre 5. Il comprend différents opérateurs et clauses comme «SELECT», «FROM» et «WHERE», pour faciliter la construction des requêtes "utilisateur". Il fournit aussi de nouvelles clauses SQL nommées «DATA.AVI», qui permet à l'utilisateur de définir la validité temporelle des données, et « DEADLINE », qui permet à l'utilisateur de fixer la date d'échéance de la requête. La date d'échéance de la requête représente une date limite relative. Elle sera ajoutée à la date de création de la requête, pour former ensemble une échéance absolue (di). La requête 14 est un exemple de la syntaxe du langage SQL-Like, utilisée dans le système BDCTR.

Requête 14 (Forme générale de la requête utilisateur)**.**

```
SELECT attributes
FROM Sensor
WHERE conditions
EPOCH duration
DATA.AVI value
DEADLINE value
```

Dans la requête 14, la clause « SELECT » définit les données à collecter du RCSF. La clause « FORM » définit la source des données. La clause « WHERE » spécifie les conditions sur les données, la clause « EPOCH » définit la durée du cycle de collecte des données du RCSF. La clause « DATA.AVI » définit la validité temporelle de données demandées par l'utilisateur et enfin la clause « DEADLINE » définit la date d'échéance de la requête.

6.4.4 Politiques d'ordonnancement des requêtes

La politique d'ordonnancement des transactions définit la façon dont les priorités sont affectées aux opérations [Abbott and Garcia-Molina, 1992]. Elle garantit que les transactions respectent leurs échéances. Nous avons utilisé deux algorithmes d'ordonnancement : RM (Rate Monotonic) et EDF (Earliest Deadline First), afin de comparer leurs performances avec une transmission multi-requêtes.

RM est un algorithme d'ordonnancement en ligne basé sur des priorités fixes. Il sélectionne la tâche ayant la plus courte période parmi toutes les tâches, pour être exécutée en premier lieu. Les transactions doivent satisfaire la condition de faisabilité. Quand l'échéance est égale à la période, cette condition est donnée par (cf. Équation 6.1).

$$U = \sum_{i=1}^{n} (\frac{Ci}{Pi}) \leq n(2^{\frac{1}{n}} - 1) \tag{6.1}$$

EDF est un algorithme d'ordonnancement en ligne basé sur des priorités dynamiques. Il sélectionne la tâche ayant le plus court délai de toutes les tâches pour être exécutées en premier. Les transactions doivent satisfaire la condition de faisabilité. Quand l'échéance est égale à la période, cette condition est donnée par (cf. Équation 6.2).

$$U = \sum_{i=1}^{n} (\frac{Ci}{Pi}) \leq 1 \tag{6.2}$$

Les notations utilisées dans ce chapitre sont présentées dans le tableau 6.1 :

Symbole	Définition
Ci	Temps d'exécution de la transaction Ti
di	Échéance absolue de la transaction Ti
Di	Échéance relative de la transaction Ti
Pi	Période de la transaction Ti
n	Nombre de la transactions
U	Taux d'utilisation totale des transactions
Ai	Temps d'arrivée de la transaction Ti

Tableau 6.1: Symboles et définitions

6.4.5 Plan d'exécution des requêtes

6.4.5.1 Diffusion des requêtes "utilisateur"

L'utilisateur dispose d'une interface graphique pour saisir sa requête. Il peut utiliser la clause «DATA.AVI», pour définir la validité temporelle des données à collectés du réseau. Il peut utiliser la clause «DEADLINE», pour définir l'échéance de la requête. Une fois que la requête est saisie, le "QueryParser" vérifie sa syntaxe et sa sémantique. Ensuite, le système crée un paquet de transactions (cf. Figure 6.5 et Tableau 6.2). On considère que les capteurs sont synchronisés avec la station de base, qui est également synchronisée avec l'ordinateur de l'utilisateur. Les capteurs et la station de base utilisent un horodatage au niveau physique, ce qui génère un horodatage pour chaque message transmis.

Figure 6.5: Forme générale d'un paquet de transaction

Symbole	Définition
TID	Identifiant de la transaction.
TDeadline	Échéance absolue de la transaction.
Query	Requête "utilisateur".
Data	Réponse à la requête.
Data.AVI	Intervalle de validité absolue des données.

Tableau 6.2: Définitions des symboles utilisés dans un paquet de transaction

Nous supposons le scénario suivant : l'utilisateur demande les identifiants et les données des capteurs, ayant un intervalle de validité absolue inférieur ou égal à 300 secondes. La requête de l'utilisateur dispose d'un délai relatif (Di) égal à 600 secondes. Les contrôleurs d'échéances utilisent une échéance absolue (di) pour vérifier les délais de la requête.

$$di = Ai\_timestamps + Di \tag{6.3}$$

La requête 15 représente la requête de l'utilisateur. La transaction aura la forme présentée dans la figure 6.6 :

Requête 15 (Forme générale de la requête utilisateur).

```
SELECT idnode, temperature FROM sensors
WHERE temperature.AVI <= 600 sec
AND DEADLINE = 300 sec.
```

TID	di	Query	d1	d1.AVI	d2	d2.AVI
1	1447104001	SELECT nodeID, Temp FROM Sensors	nodeID	Null	Temp	300

Figure 6.6: Exemple d'une transaction

La valeur "Null" dans la figure 6.6 est utilisée lorsque les données requises ne disposent pas d'un intervalle de validité temporelle.

La transaction est transmise à la station de base. Lorsque la station de base reçoit la transaction, un contrôleur de similarité compare la requête avec l'historique des requêtes. Si la requête est déjà utilisée, la station de base utilise simplement les mesures précédentes, stockées temporairement dans le cache. Si une mesure plus récente est nécessaire, la requête est placée dans la liste active des requêtes à envoyer (cf. Figures 6.7 et 6.1). Ensuite, *le contrôleur d'échéances* compare l'échéance absolue dans la requête avec l'heure actuelle. Si l'heure actuelle est inférieure à l'échéance, la requête est transmise au planificateur. Sinon, elle est supprimée parce qu'elle a raté son échéance. Si les transactions (requêtes) satisfont la condition de faisabilité, elles seront triées selon le principe de fonctionnement du programme d'ordonnancement utilisé par le système. Ensuite, la première requête dans la liste est diffusée aux nœuds "puits", qui sont couverts par le rayon de transmission de la station de base. Chaque transaction est envoyée. Ensuite, elle est retirée de la liste active et du planificateur après avoir été traitée. L'algorithme 3 représente le protocole d'ordonnancement Rate Monotonic pour la station de base et le tableau 6.3 représente les fonctions utilisées dans l'algorithme 3.

Tous les nœuds "puits" qui sont couverts par la portée de transmission de la station de base reçoivent la transaction. Le contrôleur d'échéances, au niveau du nœud "puits", compare l'échéance de la transactions avec l'heure actuelle. Si l'heure actuelle est inférieure à l'échéance de la transaction, la transaction sera transmise à l'ordonnanceur, sinon, la transaction est abandonnée. Lorsque la transaction est dans l'ordonnanceur, un test de faisabilité est effectué pour vérifier si elle satisfait la condition de faisabilité. Les opérations seront triées par la suite selon l'algorithme d'ordonnancement utilisé par le système. Ensuite, la première transaction dans la liste sera envoyée aux nœuds enfants (cf. Figure

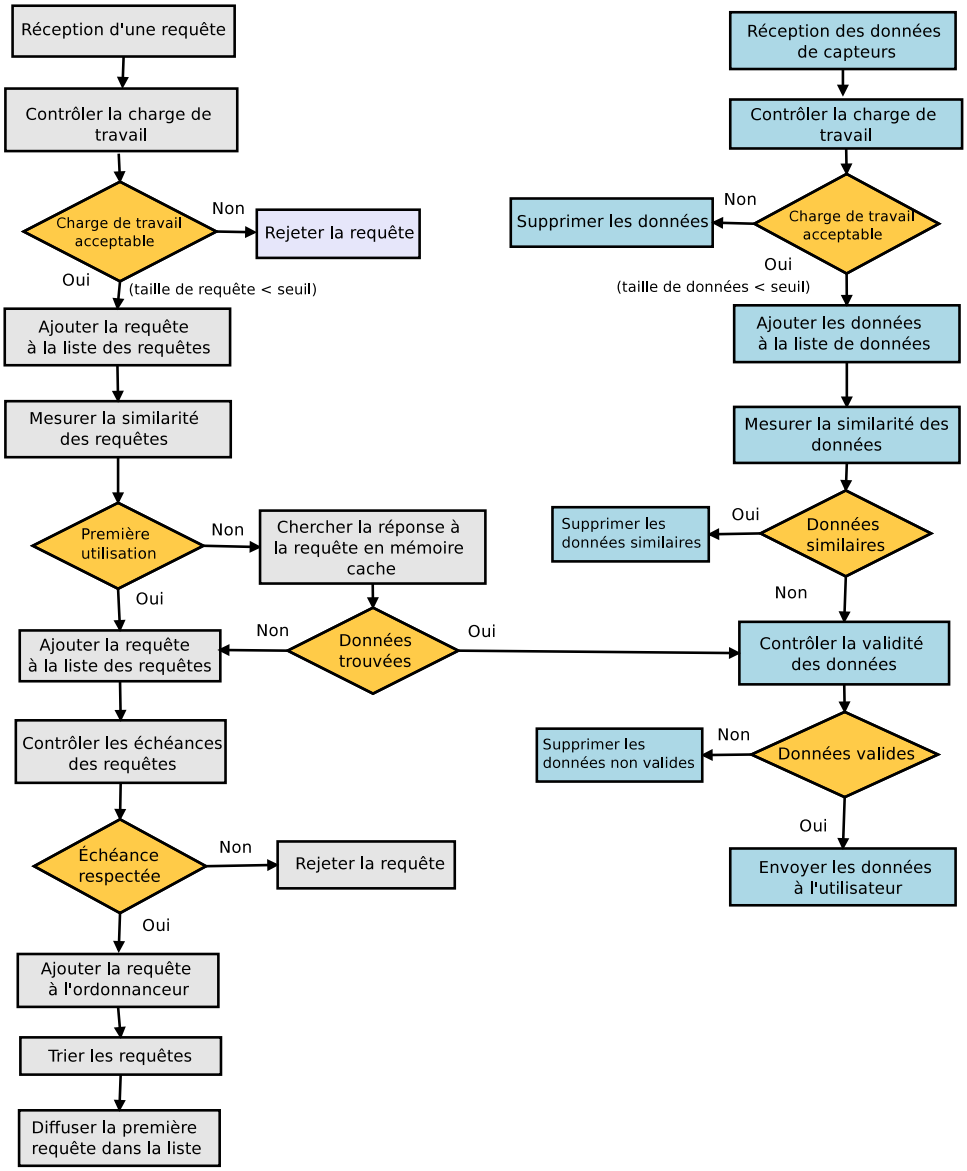


Figure 6.7: Diagramme d'activité de la station de base

6.8). Une fois que le nœud "puits" a reçu les réponses de ses nœuds enfants, la requête sera retirée de planificateur. Sinon, le nœud "puits" émet de nouveau la requête à ses nœuds enfants. De cette façon, même si le paquet est perdu, le nœud enfant aura la requête et sa réponse constituera un rappel pour le nœud "puits".

Lorsque le nœud enfant reçoit la transaction, il vérifie son échéance. Si la transaction a raté son échéance, elle sera automatiquement abandonnée. Cette tâche ne peut pas être finalisée par tous les nœuds en même temps. Sinon, le nœud enfant extrait la requête du paquet, puis il l'exécute. Chaque nœud gère les requêtes reçues indépendamment des autres nœuds. Les opérations sont effectuées jusqu' à la fin.

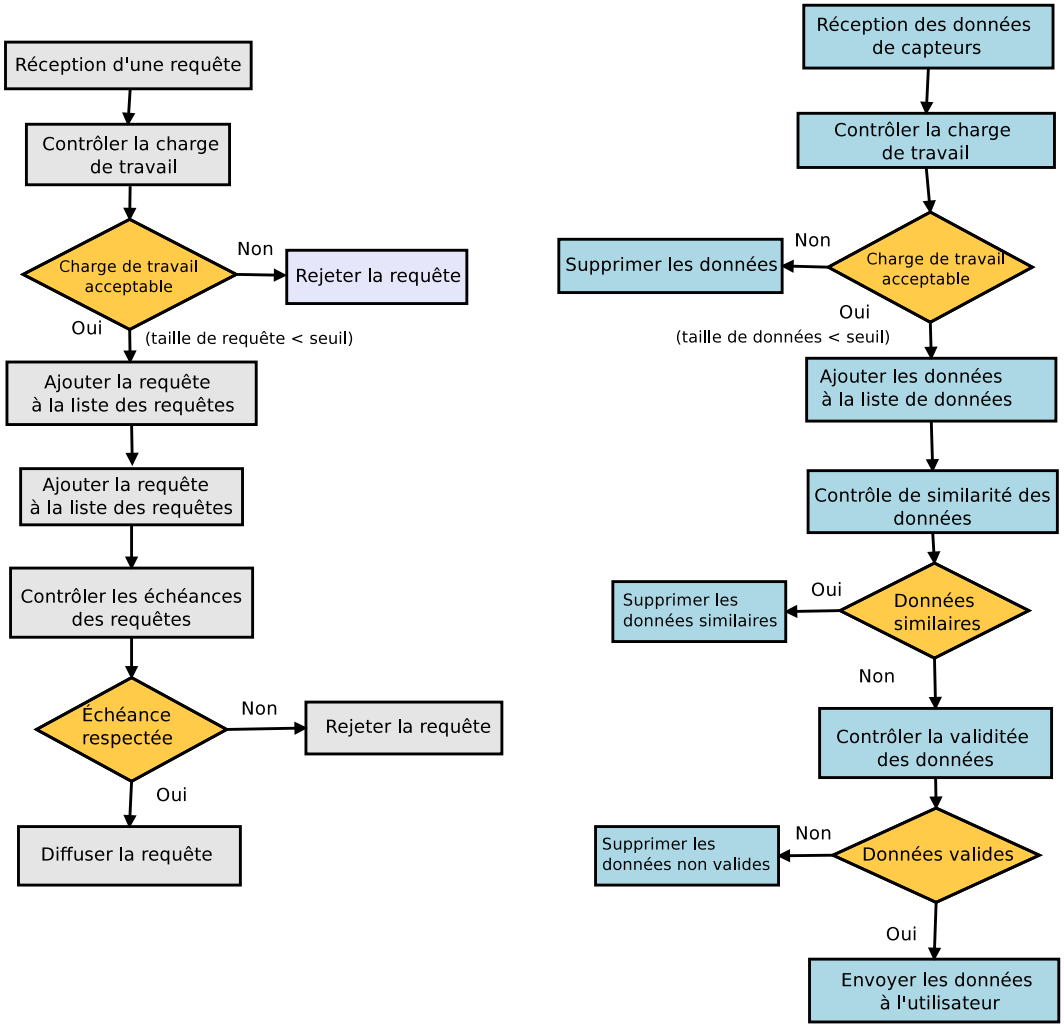


Figure 6.8: Diagramme d'activité du nœud "puits"

6.4.5.2 Collection des réponses aux requêtes :

Cette étape est consacrée à la collecte des réponses aux requêtes. Le nœud enfant construit un message de réponse composé des éléments présentés dans la figure 6.9 :

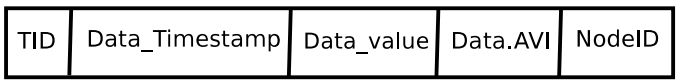


Figure 6.9: Forme générale d'un message réponse

Le nœud enfant construit une réponse. Ensuite, il l'envoie à son nœud parent. Lorsque le nœuds parent reçoit la réponse de son nœud enfant à l'heure d'arrivée A_i, il vérifie sa validité temporelle, en se basant sur le temps d'arrivée du message et sur la date d'extraction des données. Si le résultat est inférieur au temps de validité temporelle fixé par l'utilisateur, les réponses seront transmises à la station de base (cf. Équation 6.4).

Algorithm 3: Algorithme d'ordonnancement (RM) de la station de base

```
while 1 do
    if ev = serial_line_event_message then
        if QueryHistory (NewQuery) = 0 then
            InsertQueryInlist (ArrivalT, NewQuery);
            if RM_Feasibility(ArrivalT) = 1 then
                ArrivalT← Sort_by PI(ArrivalT);
                FirstT ← Get_FirstT (ArrivalT);
                Period ← Get_period (FirstT);
                Etimer_set(&periodic,Period);
                SendQuery(FirstT);
                Delete_FirstT(ArrivalT);
            else
                Read_From_Cache(NewQuery);
            end
        end
    end
end
```

Fonction	Explication
Sort by PI(Transaction list)	Ordonner les transactions suivant la politique de l'algorithme RM.
Get FirstT (Transaction list)	Récupérer le premier élément dans la liste de transactions.
Get period (Transaction)	Récupérer la période d'envoie de la requête.
Etimer set(periodic,Period)	Régler la minuterie sur la période d'envoie de la requête.
Send Query(Transaction)	Envoyer la requête dans le RCSF.

Tableau 6.3: Tableau de fonctions pour l'algorithme 3

Symbole	Définition
TID	identifiant de la transaction.
Data_Timestamp	horodatage d'extraction des données
Data_value	valeur des données demandées.
NodeID	identifiant du nœud.
Data.AVI	intervalle de validité absolue des données.

Tableau 6.4: Définitions des symboles utilisés dans un message de réponse

$$A_i - DTimestamp < Data\_avi \tag{6.4}$$

Lorsque un nœud "puits" reçoit les mêmes réponses de ses nœuds fils, il ne transmet qu'un seul message de réponse à la station de base qui sera vérifié au niveau de chaque nœud "puits", jusqu'à ce qu'il atteigne la station de base. Lorsque un nœud "puits" reçoit une

réponse différente, il ne la transmet à la station de base que lorsque son intervalle de validité est vérifié. Sinon, la réponse à la requête est abandonnée.

6.5 Évaluation des performances

Nous avons utilisé le simulateur de réseau de système ContikiOS, nommé "Cooja" [Österlind *et al.*, 2006], pour évaluer notre système de bases de données de capteurs temps réel. Cooja fournit différents modèles de capteurs tels que Tmote Sky et MicaZ. Nous avons défini un nombre de nœuds de type Tmote Sky dans les simulations qui varie entre 10 et 40 nœuds. Nous avons choisi une portée de transmission de 40 mètres de manière que la portée de transmission de la station de base couvre les nœuds "puits", et la portée de transmission des nœuds "puits" couvre les nœuds émetteurs. Le tableau 6.5 présente les paramètres de simulation utilisées dans nos expériences.

Paramètres	**Valeur**
Modèle de capteur	Tmote sky
Zone de simulation	100m*100m
Nombre de nœuds	10 à 40
Portée de transmission d'un nœud	40m
Taille maximale d'un paquet	128 bytes
Vitesse de transmission	250 kbps
Couche MAC	CSMA/CA
Couche radio (duty-cycle)	ContikiMac
Modele de propagation radio	Unit Disk Graph Medium (UDGM) Distance Loss

Tableau 6.5: Paramètres de simulation

Notre première expérience consiste à envoyer plusieurs requêtes à la station de base selon une période fixe. Nous avons testé deux performances : (1) le taux de réussite des transactions et (2) le nombre de données valides, sans ordonnancement et avec ordonnancement en utilisant les deux algorithmes Rate-Monotonic (RM) et Earliest deadline first (EDF). Nous avons utilisé trois modèles de requêtes. Le premier modèle Q1, comprend une clause de délai de validité absolue sur les données ayant une valeur égale à l'intervalle des requêtes.

```
Q1: SELECT idnode, temperature FROM sensors
    WHERE DEADLINE = query interval.
```

La deuxième modèle de requêtes Q2 contient une clause de validité temporelle sur les données inférieure ou égale à 20 secondes :

```
Q2: SELECT idnode, temperature FROM sensors
    WHERE temperature.AVI <= 20 sec.
```

Le troisième modèle de requête Q3 contient une clause de validité temporelle sur les données inférieure ou égale à 20 secondes et une clause de délai de validité des requêtes ayant une valeur égale à l'intervalle des requêtes.

```
Q3: SELECT idnode, temperature FROM sensors
    WHERE temperature.AVI <= 20 sec AND DEADLINE = query interval.
```

6.5.1 Transmission multi-requêtes avec un intervalle d'envoi fixe

6.5.1.1 Mesure du taux de réussite des transactions

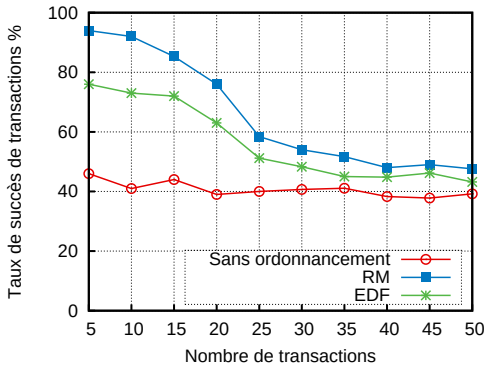
Nous considérons qu'une transaction est réussie lorsqu'elle respecte son échéance, à savoir que la station de base reçoit la réponse à la requête à la date définie par l'utilisateur sans manquer son échéance. Le taux de réussite des transactions est calculé comme suit :

$$SRatio = \frac{CommitT}{SubmittedT} \tag{6.5}$$

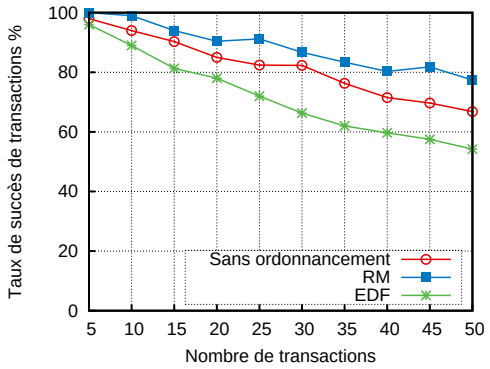
où $CommitT$ indique le nombre de transactions qui ont respecté leurs échéances et $SubmittedT$ indique le nombre de transactions soumises durant la période d'échantillonnage.

Pour observer le comportement du système avec des requêtes périodiques et voir l'impact de l'ordonnancement sur le taux de réussite des transactions, nous avons créé un générateur de transactions, connecté à la station de base. L'utilisateur envoie des requêtes périodiques à la station de base, suivant un intervalle défini. D'abord, nous avons choisi un intervalle égal à 10 secondes, puis à 20 secondes et enfin à 30 secondes. Nous avons testé le système sans et avec ordonnancement, avec des requêtes dont l'échéance relative est égale à la période d'envoi. Les testes sont réalisés sur 10 nœuds et avec un nombre de transactions de 5, 10, 20, 30, 40 et 50. Les résultats des simulations sont présentés dans les figures 6.10a, 6.10b et 6.10c.

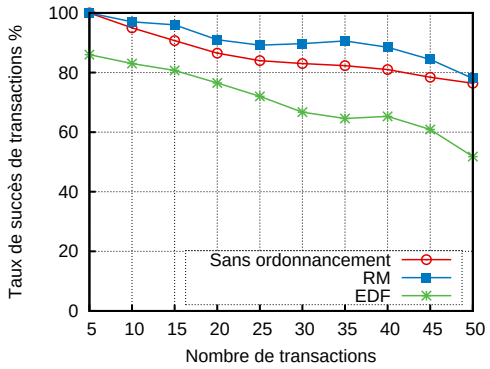
Nous notons dans la figure 6.10a, que l'algorithme d'ordonnancement EDF est moins performant que RM et que ce dernier donne le meilleur taux de réussite des transactions. Lorsque le système n'utilise pas un algorithme d'ordonnancement, la première transaction arrivée est la première envoyée aux capteurs. De cette manière, les transactions sont exécutées indépendamment de leurs priorités, ce qui affecte le temps d'exécution d'autres transactions.
L'algorithme d'ordonnancement EDF est basé sur des priorités dynamiques. Il sélectionne les transactions ayant l'échéance la plus courte pour être exécutées en premier. On remarque qu'il retourne de meilleurs résultats avec un petit nombre de transactions. Cependant, ses performances diminuent avec un nombre élevé de transactions. La durée



(a) Intervalle d'envoi de 10 secondes



(b) Intervalle d'envoi de 20 secondes



(c) Intervalle d'envoi de 30 secondes

Figure 6.10: Taux de succès de transactions avec différents intervalles d'envoi

d'ordonnancement des transactions augmente avec le nombre de transactions dans la fil d'attente. Ceci peut influer sur la durée d'exécution des transactions et au final, sur le temps de réponse de la requête.
L'algorithme d'ordonnancement RM est basé sur un ordonnancement à priorités statiques pour des tâches périodiques, quand la période égale l'échéance. Il considère que les transactions avec une courte période sont les plus prioritaires. Les transactions sont exécutées jusqu'à la fin de leurs échéances sans préemption, ce qui augmente leurs taux de succès.

Nous pouvons constater aussi que l'intervalle d'envoi des requêtes a un impact sur le taux de réussite des transactions. Le taux de succès des transactions utilisant un intervalle d'envoi égal à 10 secondes (cf. Figure 6.10a) est inférieur au taux de succès des transactions avec d'autres intervalles d'envoi (cf. Figures 6.10b et 6.10c). Nous expliquons ces résultats comme suit : le temps nécessaire pour envoyer les données à partir du premier niveau jusqu'au deuxième niveau dans l'arbre du réseau (cf. Figure 6.4) demande une durée d'envoi supérieure à 10 secondes. Les nœuds situés dans le niveau 1, c'est à dire les nœuds avec un seul saut vers la station de base peuvent communiquer directement avec cette dernière et peuvent envoyer leurs réponses en respectant leurs échéances. Cependant, les

nœuds situés au niveau 2, reçoivent les requêtes avec des retards, ce qui peut affecter la date d'arrivée des transactions. Nous pouvons conclure qu'il est important d'estimer les délais de transmission entre les différents niveaux dans l'arbre du réseau, pour donner la date limite appropriée aux transactions.

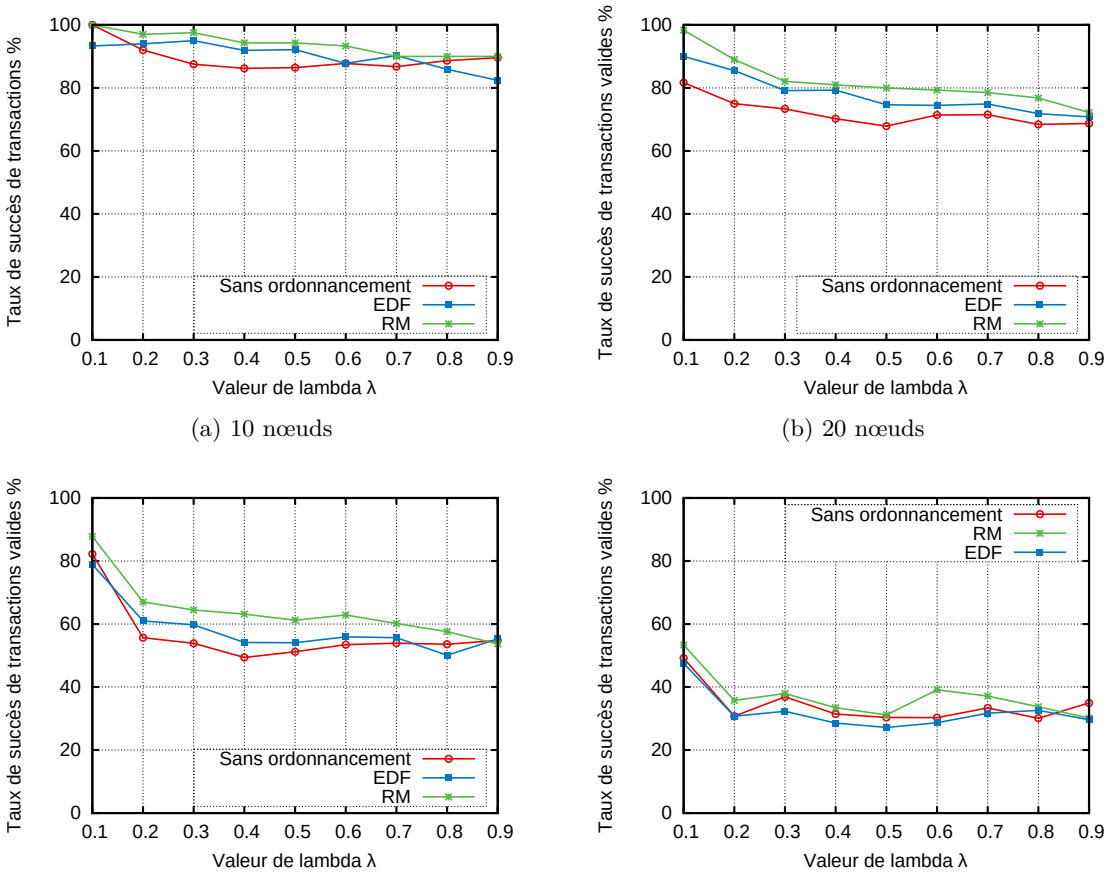
Nous notons dans la figure 6.10b que le taux de réussite des transactions est meilleur lorsque nous avons augmenté l'intervalle d'envoi des requêtes à 20 secondes, comparé à la figure 6.10a. Nous pouvons interpréter ces résultats comme suit. D'abord, le temps nécessaire pour recevoir des requêtes ou envoyer des réponses a augmenté. En conséquence, les nœuds situés dans le niveau 2 peuvent envoyer leurs réponses, en respectant les délais des transactions. L'algorithme EDF prend beaucoup de temps pour planifier les transactions dans la file d'attente, ce qui peut affecter les échéances des transactions. Nous pouvons conclure alors que l'algorithme RM est meilleur que l'algorithme EDF avec des transactions périodiques. Nous relevons que nous avons également des résultats similaires avec un intervalle d'envoi des requêtes égal à 30 secondes (cf. Figure 6.10c).

Nous avons augmenté le nombre de nœuds pour tester les performances du notre système. Nous avons fait varier le nombre de nœuds de 10 à 40, avec une période d'envoi égale à 30 secondes, car cette période a donné de meilleurs résultats dans les tests précédents. Nous avons utilisé la loi de Poisson pour générer un nombre aléatoire de transactions, en nous basant sur le paramètre λ, qui représente le taux d'arrivée des transactions. Le tableau 6.6 présente le rapport entre la valeur de λ et le nombre de transactions. Les résultats des

λ	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
Nombre de transactions	3	10	12	21	28	36	40	44	50

Tableau 6.6: Nombre de transactions générées avec un intervalle d'envoi fixe

simulations sont présentés dans les figures 6.11a, 6.11b et 6.11c. Nous pouvons remarquer dans les figures 6.11a, 6.11b et 6.11c qui le taux de réussite des transactions diminue à chaque fois qu'ont augmente le nombre de nœuds. Nous notons aussi que l'algorithme RM donne toujours de meilleurs résultats que l'algorithme EDF. Les performances de l'algorithme EDF baissent lorsqu'on augmente le nombre de transactions et elles se dégradent encore plus lorsqu'on augmente le nombre de œuds. Nous remarquons aussi que notre système de base de données de capteurs donne de mauvais résultats sans algorithme d'ordonnancement, mais ne présente pas de piques.



(a) 10 nœuds

(b) 20 nœuds

(c) 30 nœuds

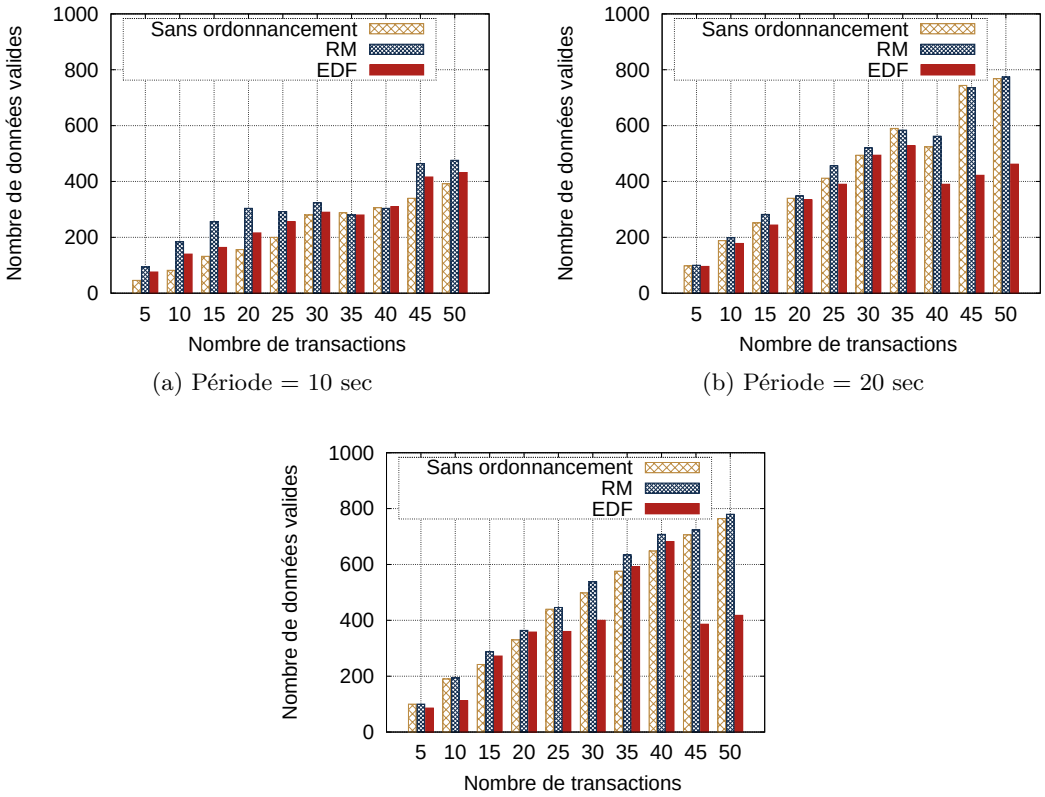
(d) 40 nœuds

Figure 6.11: Taux de succès de transactions

6.5.1.2 Nombre de données valides

Dans cette expérience, nous étudions l'impact de l'algorithme d'ordonnancement sur le nombre de données valides. Nous avons utilisé la requête Q2 avec trois intervalles d'envoi (10, 20 et 30 secondes). Nous avons fixé l'intervalle de validité absolue des données inférieur ou égal à 20 secondes, car nous avons remarqué que le temps nécessaire pour envoyer des données d'un niveau à l'autre nécessite 10 secondes. Comme notre arbre de réseau est composé de deux niveaux, nous avons un intervalle de validité absolue des données égal à 20 secondes. Les résultats des simulations sont présentés dans les figures 6.12a, 6.12b et 6.12c.

Les figures 6.12a, 6.12b et 6.12c montrent que l'algorithme d'ordonnancement RM donne le meilleur nombre de données valides. RM utilise des priorités fixes qui permettent aux nœuds de capteurs de terminer leurs transactions. De la même manière, une collecte des données sans ordonnancement assure une exécution des transactions jusqu'à la fin, ce



(a) Période = 10 sec

(b) Période = 20 sec

(c) Période = 30 sec

Figure 6.12: Nombre de données valides

qui améliore le nombre de données valides reçues. En revanche, EDF est un algorithme à priorité dynamique, qui peut changer au cours de son exécution. L'exécution d'une requête sur un nœud de capteur peut être entravée par l'arrivée d'une nouvelle requête avec une échéance plus courte. En conséquence, la requête ne peut être exécutée jusqu'à la fin. Nous pouvons constater aussi que l'algorithme d'ordonnancement EDF donne de meilleurs résultats avec un petit nombre de transactions, et avec un grand intervalle d'envoi. Nous pouvons conclure que l'algorithme d'ordonnancement RM est bien adapté à notre système avec des requêtes périodiques, car il donne de meilleurs résultats dans les trois intervalles d'envoi, ainsi que lorsque nous augmentons le nombre de transactions.

6.5.2 Transmission multi-requêtes avec un intervalle d'envoi dynamique

Les requêtes des utilisateurs sont transmises aléatoirement à la station de base. Nous utilisons un générateur de transactions basé sur le processus de Poisson avec λ, le taux d'arrivée des transactions *utilisateur*. La charge de travail de la station de base varie en fonction de λ. Lorsque $\lambda = 0{,}1$, le nombre de transactions d'utilisateurs arrivant au

système lors d'une expérience est d'environ 3 transactions. Lorsque $\lambda = 0{,}9$, ce nombre est d'environ 50 transactions (cf. tableau 6.7). Nous avons testé les performances du notre

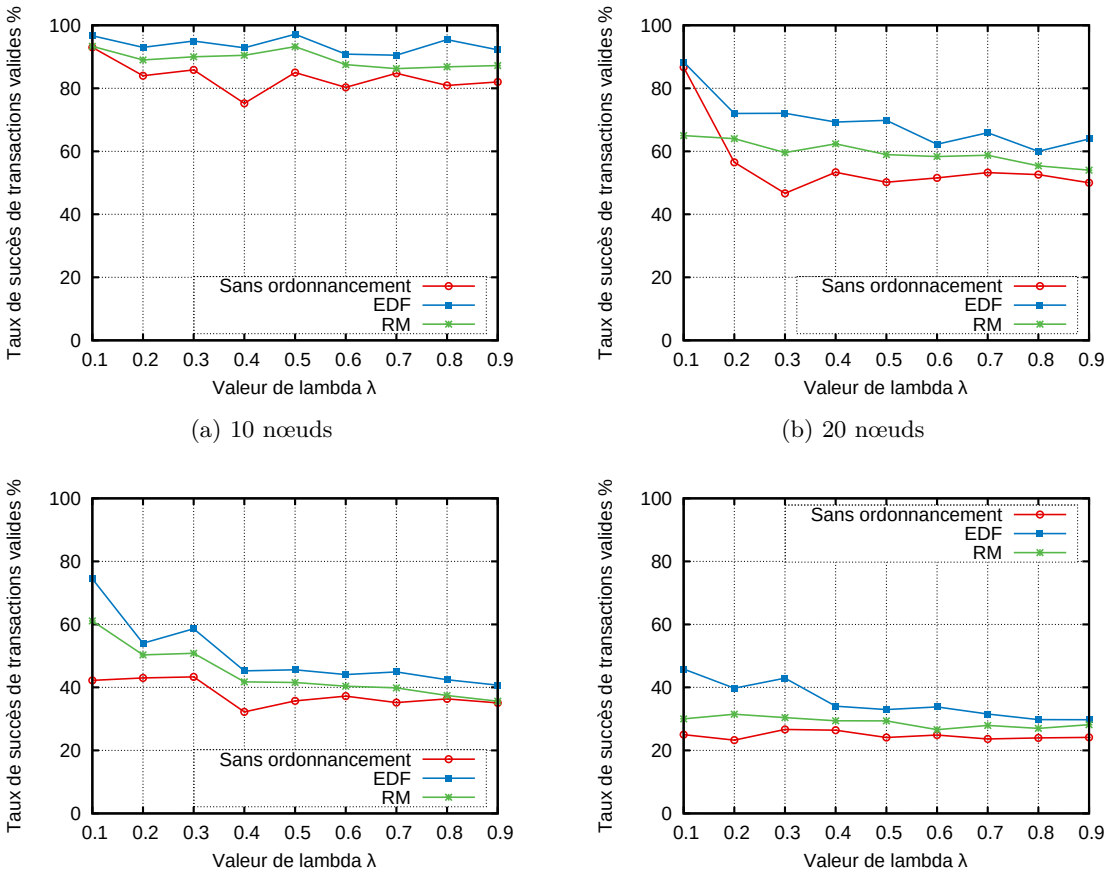
λ	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
Nombre de transactions	3	10	12	21	28	36	40	44	50

Tableau 6.7: Nombre de transactions générées avec un intervalle d'envoi dynamique

système afin de déterminer le taux de succès de transactions en variant le nombre d'arriver des transactions à la base de données.

6.5.2.1 Mesure du taux de succès des transactions

Dans cette expérience nous avons varié le nombre de transactions entre 3 et 50 ainsi que le nombre de capteurs dans le réseau entre 10 et 40 nœuds. Nous avons utilisé une portée de transmission égale à 40 mètres, car nous avons remarqué qu'avec cette portée, le taux de succès des transactions augmente.



(a) 10 nœuds

(b) 20 nœuds

(c) 30 nœuds

(d) 40 nœuds

Figure 6.13: Taux de succès des transactions avec un intervalle d'envoi dynamique

Dans les figures 6.13a, 6.13b, 6.13c, 6.13d, nous remarquons que le taux de succès des transactions diminue avec l'augmentation du nombre de nœuds. Avec une augmentation du nombre de nœuds, la charge de travail au niveau des nœuds "puits" augmente et elle réduit la capacité des capteurs à traiter les transactions. L'utilisation des ordonnanceurs a amélioré le taux de succès des transactions. L'algorithme EDF donne le meilleur taux de succès par rapport à l'algorithme RM, qui s'adapte bien aux tâches périodiques.

6.6 Comparaison avec d'autres solutions selon quelques critères

Dans cette section, nous nous sommes intéressés d'abord à comparer les fonctionnalités de notre système avec d'autres bases de données de capteurs, selon certains critères, le modèle de base de données, la durée de vie de la requête, les fonctions d'agrégation, l'exécution multi-requêtes, la synchronisation et le routage. Ensuite, nous avons choisi la base de données de capteurs TinyDB pour comparer ses performances avec les performances de notre système. Nous avons choisi le facteur de performance consommation d'énergie.

6.6.1 Comparaison des fonctionnalités

6.6.1.1 Modèle de base de données

Les bases de données de capteurs comme TinyDB et Cougar utilisent l'approche base de données comme un modèle, afin d'améliorer la collecte des données dans un RCSF. Elles utilisent des techniques de traitement de requêtes pour réduire le nombre de transmission des données ainsi que la consommation d'énergie dans le réseau. Notre base de données de capteurs temps réel utilise un modèle de base de données temps réel pour assurer le respect des contraintes temporelles dans le RCSF.

6.6.1.2 Durée de vie de requêtes

Les BDCSF prennent en charge la durée de vie de la requête. Elles utilisent des clauses dans un langage similaire à SL (SQL-Like) comme `<DUREE>` ou `<EPOCH>` pour spécifier la durée entre deux exécutions de requêtes, ou pour déterminer la période pendant laquelle les données sont agrégées. Elles ne définissent pas un modèle de données temporelles pour gérer les données de capteurs. Notre base de données de capteurs temps réel utilise un modèle de données temporelles pour aider les contrôleurs d'échéances à déterminer les données et les transactions qui ont raté leurs échéances. En outre, elle fournit de nouvelles clauses SQL-Like pour l'utilisateur, pour ajuster les contraintes temporelles liées aux données et aux transactions.

6.6.1.3 Fonctions d'agrégations

TinyDB et Cougar supportent les fonctions d'agrégation pour réduire le nombre de données transmises dans le RCSF. Nous n'avons pas encore mis en œuvre des fonctions d'agrégation dans notre BDCSFTR. Nous prévoyons d'ajouter ces fonctions pour améliorer les performances du système. Actuellement, nous avons utilisé les contrôleurs d'échéances sur les nœuds "puits" et la station de base, afin d'éliminer les données non valides et réduire le nombre de données transmises dans le RCSF.

6.6.1.4 Exécution multi-requêtes

Cougar et MaD-Wise ne supportent pas une exécution multi-requêtes. TinyDB supporte une exécution multi-requêtes, mais ne prend pas en charge l'ordonnancement. Notre système supporte une exécution multi-requêtes et fournit un ordonnancement en temps réel avec des règles de priorité attribuées aux transactions.

6.6.1.5 Synchronisation et routage

Notre système utilise la synchronisation des nœuds pour aider les contrôleur d'échéances à vérifier les échéances des requête et l'intervalle de validité temporelle des données. En outre, il construit les chemins vers la station de base en utilisant un routage basé sur la latence. TinyDB utilise un protocole de synchronisation sur chaque nœud qui définit le début et la fin de chaque période de collecte des données. Il utilise la méthode d'arbre de routage sémantique pour obtenir des informations sur les capacités des nœuds afin de déterminer quels sont les nœuds qui peuvent donner une réponse à la requête de l'utilisateur. Cette méthode permet de réduire le nombre de nœuds actifs et d'améliorer la durée de vie du réseau. Notre système utilise un modèle de réseau basé sur une topologie arborescente. Il sélectionne les nœuds "puits" et définit des chemins entre les capteurs et la station de base en se basant sur la latence minimale entre deux nœuds. Cette méthode permet d'améliorer le temps de transmission des requêtes ainsi que le temps de collecte des données.

6.6.2 Comparaison avec TinyDB

6.6.2.1 Consommation d'énergie

Nous avons utilisé l'extension *PowerTOSSIM* [Shnayder *et al.*, 2004] du simulateur de TinyOS *TOSSIM* [Levis *et al.*, 2003], afin de calculer la consommation d'énergie du système TinyDB. Nous avons utilisé des capteurs virtuels de type MICA2. *PowerTOSSIM*

permet de déterminer la consommation énergétique d'une application sous TinyOS, telle que TinyDB, en utilisant les formules 6.6 et 6.7.

$$E(mW) = V * I * t \tag{6.6}$$

$$E(mJ) = mW * ts \tag{6.7}$$

Avec :

E : la consommation énergétique.
V : la tension utilisée par le capteur.
I : le courant utilisé par le capteur.
t : le temps d'accès au capteur.
ts : le temps écoulé en secondes.

Modèle de capteur	Tmote Sky	Mica2
CPU (active)	1.8 2.4 mA	7.6 8.0mA
LPM	0.0545 1.2 mA	3.2 3.3mA
Radio (RX)	19.7 mA	8.0 9.6 mA
Radio (TX)	17.7 mA	17 25 mA
Tension	3 v	3 v

Tableau 6.8: Modèle d'énergie du Tmote Sky et Mica2 [ZHU, 2013]

Nous avons utilisé le mécanisme (Energest power) [Dunkels *et al.*, 2007] avec le simulateur Cooja sous le système Contiki, pour estimer la consommation d'énergie pendant l'exécution de notre modèle, en nous basant sur la formule 6.7. Nous nous référons aux modèles d'énergie du capteur (Tmote Sky) pour notre modèle et le capteur (Mica2) pour TinyDB (cf. Tableau 6.8). Les résultats des simulations sont présentés dans la figure 6.14.

Nous notons dans la figure 6.14 que notre système de base de données de capteurs temps réel consomme moins d'énergie que TinyDB. TinyDB utilise un arbre de routage sémantique pour sélectionner les nœuds qui peuvent donner une réponse à la requête parmi l'ensemble de nœuds dans le réseau. De plus, les processus d'exécution des requêtes et de collecte des réponses consomment beaucoup d'énergie. En revanche, notre système utilise le processus de sélection des nœuds "puits", basé sur la latence, qui consomme moins d'énergie que le processus de construction d'arbre de routage sémantique de TinyDB. De plus, les contrôleurs d'échéances au niveau des nœuds "puits" réduisent la quantité de données à traiter, et en conséquence, réduisant ainsi la quantité d'énergie à consommer.

Nous remarquons également qu'en utilisant l'algorithme d'ordonnancement RM, la con-

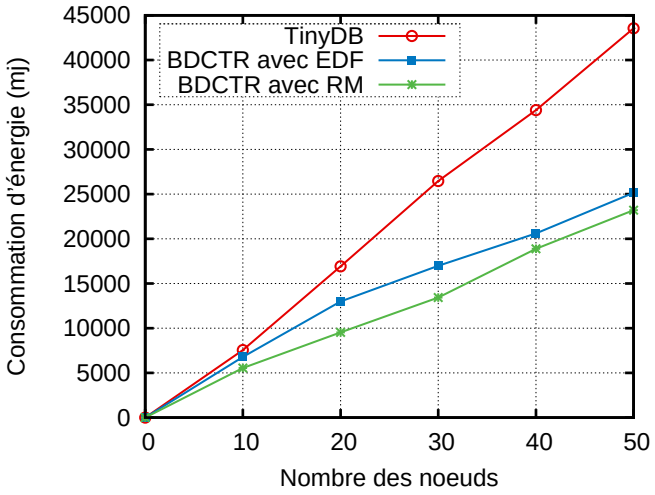


Figure 6.14: Consommation d'énergie

sommation d'énergie de notre système de base de données de capteurs temps réel est inférieure à la consommation en utilisant l'algorithme d'ordonnancement EDF. EDF étant un algorithme à priorité dynamiques, il vérifie à chaque instant les transactions qui se rapprochent de leurs échéances. Pour cette raison, il consomme beaucoup d'énergie pour la sélection des transactions avec des échéances courtes ainsi que pour les planifier dans la file d'attente. Par contre, l'algorithme d'ordonnancement RM est à priorité fixe. Il ne s'occupe pas de connaitre si une transaction se rapproche de son échéance, qui réduit la consommation d'énergie du processus de planification.

6.7 Conclusion

Dans ce chapitre, nous avons conçu une nouvelle base de données de capteurs temps réel pour améliorer la cohérence temporelle des données dans un RCSF ainsi que le taux de succès des transactions. Nous avons proposé une nouvelle architecture de bases de données de capteurs temps réel, qui implémente des contrôleurs d'échéance et des ordonnanceurs temps réel pour attribuer des priorités aux transactions. L'objectif est de (i) réduire la charge de travail au niveau de la station de base et (ii) d'augmenter le nombre de données valides et le taux de succès des transactions. L'algorithme d'ordonnancement RM a montré de meilleures performances avec des transactions périodiques. L'algorithme EDF a donné de meilleurs résultats avec des transactions non périodiques. Nous considérons que les performances de notre système de base de données de capteurs en temps réel sont obtenues avec des requêtes périodiques. Notre système a montré de meilleures performances en terme de consommation d'énergie que le système TinyDB, grâce au processus de sélection des nœuds "puits" et aux contrôleurs d'échéances, qui ont réduit le nombre de données

non valides dans le réseau. Nous prévoyons d'améliorer ses performances en utilisant les transactions relaxées [Sadeg and Saad-Bouzefrane, 2000].

Chapitre 7

Conclusion et perspectives

Dans cette thèse, nous avons étudié les contraintes temporelles dans les bases de données de capteurs sans fil. Nous avons entamé cette étude par un état de l'art, dans lequel nous avons présenté les systèmes de bases de données de capteurs, leurs architectures et leurs langages de requêtes. Nous avons présenté par la suite les contraintes temporelles dans ces systèmes ainsi que les différents protocoles temps réel dans les réseaux de capteurs sans fil (RCSF).

Notre première contribution consiste à étudier les contraintes temporelles dans les bases de données des capteurs et dans les bases de données classiques. Cette étude nous a permis d'identifier leurs faiblesses et les enjeux sur lesquels nous pouvons intervenir. Nous avons déterminé que l'architecture d'une base de données de capteurs sans fil n'est pas conçue pour respecter les contraintes temporelles, car les systèmes actuels se focalisent plutôt sur la minimisation de la consommation d'énergie dans le réseau. Néanmoins, la topologie du réseau et le protocole de routage peuvent jouer un rôle important dans l'amélioration du temps de transmission des requêtes et du temps de collecte des données.

Dans la deuxième contribution, nous nous sommes concentrés sur la mise en place d'un nouveau modèle de requêtes, utilisant deux nouvelles clauses SQL et un algorithme de contrôle de la validité temporelle des données et des échéances des requêtes au niveau de chaque nœud. Nous avons testé les performances du modèle de requêtes avec deux modèles de réseaux. Le premier est un modèle dynamique utilisant une méthode d'auto-sélection des nœuds "puits". L'auto-sélection est basée sur la latence entre les capteurs. Pour ce faire, nous avons proposé d'ajouter une nouvelle fonction objective pour le protocole de routage RPL, afin d'utiliser les chemins avec une latence minimale entres les capteurs. Le

deuxième modèle est statique. Les nœuds sont fixés manuellement par l'utilisateur. Le modèle dynamique a donné de meilleurs résultats par rapport au modèle statique en terme du temps de réponse aux requêtes ainsi que du nombre de données valides reçues par la station de base. Nous avons testé le modèle de requêtes avec trois approches conçues pour maintenir la validité temporelle des données, qui sont "One-One", "Half-Half" et "More-Less". Les résultats ont montré que l'algorithme "One-One" est le plus adapté à notre modèle de traitement des requêtes car il donne le meilleur taux de données valides par rapport aux autres approches ("More-Less" et "Half-Half"). Cependant, nous avons remarqué durant les simulations une augmentation de la charge de travail au niveau des nœuds "puits", engendrée par le flux de données reçues de la station de base (les requêtes) et aussi par le flux des réponses des capteurs. Pour résoudre ce problème, nous avons pensé utiliser un système de gestions de données temps réel pour les RCSF.

Dans la troisième contribution, nous avons conçu une base de données de capteurs temps réel avec une nouvelle architecture. Elle implémente des contrôleurs d'échéances et des ordonnanceurs temps réel, afin de réduire la charge de travail au niveau de la station de base, ainsi qu'au niveau des nœuds "puits" et pour augmenter le nombre de données valides, ainsi que le taux de succès des transactions. Nous avons analysé les performances du système avec des requêtes périodiques et des requêtes non périodiques. Les résultats des simulations ont montré que les requêtes périodiques donnent un meilleur taux de réussite des transactions par rapport aux requêtes non périodiques. Nous avons également testé la consommation d'énergie de notre système de base de données de capteurs temps réel, qui a montré de meilleures performances par rapport au système de base de données de capteurs "TinyDB".

Perspectives

Ces dernières années, l'internet des objets a intégré plusieurs domaines. Il s'agit de connecter différents objets de manière à ce qu'ils puissent fournir des informations. Le concept de l'internet des objets est très proche de celui de réseau de capteurs sans fil d'un point de vue technique. Les RCSF sont constitués de plusieurs capteurs sans fil qui collectent des données pour superviser des processus industriels et analyser des phénomènes naturels. Cependant, l'internet des objets connecte plusieurs objets et différents réseaux. Les objets se connectent à travers une passerelle (gateway) au réseau Internet pour envoyer les données collectées au serveurs de stockage distant (cloud).

Cette thèse ouvre plusieurs perspectives pour mener de futurs travaux de recherche sur l'internet des objets. L'accès concurrentiel des objets connectés à la passerelle engendre des pertes de données ainsi que des retards de transmission. Les objets connectés peuvent

être considérés comme des capteurs hétérogènes qui fournissent temporairement ou en permanence des données. La gestion de ce grand nombre de flux de données nécessite la mise en place d'un système de gestion bases de données temps réel pour gérer les accès concurrents et les transmissions en temps réel des objets connectés ainsi que l'hétérogénéité des objets et des réseaux, les sources de données multiples et la quantité de données reçues par la passerelle.

Parmi les améliorations que nos contributions dans cette thèse peuvent apporter au contexte de l'internet des objets, on peut citer les points suivants : (1) l'utilisation d'un langage de requête, comme SQL, pour interroger les différents objets connectés dans différents réseaux, afin de résoudre le problème d'hétérogénéité des réseaux. (2) Notre système de bases de données temps réel peut gérer les priorités des transactions dans un environnement connecté. Il permet aussi de réduire le nombre de transmissions par le filtrage de données redondantes provenant de différents sites.

Publications

** Chapitre de livre

1 - Abderrahmen Belfkih, Bruno Sadeg, Claude Duvallet, Laurent Amanton, Book Series: Advances in Sensors: Reviews, Book title: Sensors, Transducers, Signal Conditioning and Wireless Sensors Networks, Publisher: (IFSA) Publishing, Vol. 3, chapter:18, Pages:362-380, 2016.

** Revue internationale

1 - Abderrahmen Belfkih, Bruno Sadeg, Claude Duvallet, Laurent Amanton, Comparative Study of Temporal Aspects of Data Collection Using Remote Database vs. Abstract Database in WSN, Sensors & Transducers Journal, Vol. 189, Issue 6, pp.71-81, June 2015.

** Revue nationale

1 - Abderrahmen Belfkih, Bruno Sadeg, Claude Duvallet, Laurent Amanton. Les bases de données dans les réseaux de capteurs sans fil Technique et Science Informatiques. Volume 33/9-10. 2014. pp.739-776.

** Conférences internationales avec comité de lecture

1 - Abderrahmen Belfkih, Bruno Sadeg, Claude Duvallet and Laurent Amanton. A New Query Processing Model for Maintaining Data Temporal Consistency in Wireless Sensor Networks. Proceedings of the IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, April 2015. Pages 1-6.

2 - Abderrahmen Belfkih, Bruno Sadeg, Claude Duvallet and Laurent Amanton. Study of timing properties on data collection and query processing techniques in wireless sensor networks. Proceedings of the 4st International Conference on Sensor Networks (SENSORNETS), Angers/France, February 2015, Pages 77-84.

3 - Abderrahmen Belfkih, Bruno Sadeg, Claude Duvallet and Laurent Amanton. Study of temporal constraints for data management in wireless sensor networks. Proceedings of the 8th Junior Researcher Workshop on Real-Time Computing (JRWRTC), Versailles/France, Octobre 2014, Pages 29-32.

Références

[Abbott and Garcia-Molina, 1992] Robert K. Abbott and Hector Garcia-Molina. Scheduling real-time transactions: A performance evaluation. *ACM Trans. Database Syst.*, 17(3):513–560, sep 1992.

[Ahmed and Fisal, 2008] Adel Ali Ahmed and Norsheila Fisal. A real-time routing protocol with load distribution in wireless sensor networks. *Computer Communications*, 31(14):3190–3203, 2008.

[Ali *et al.*, 2011] N.A. Ali, M. Drieberg, and P. Sebastian. Deployment of micaz mote for wireless sensor network applications. In *IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE)*, pages 303–308, Dec 2011.

[Allen, 1983] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, November 1983.

[Alsboui *et al.*, 2012] Maguy Tariq Alsboui, Abdelrahman AB Uarqoub, Mohammad Hammoudeh, Zuhair Bandar, and Andy Nisbet. Information extraction from wireless sensor networks: System and approaches. *Sensors and Transducers journal*, 14(2):1–17, March 2012.

[Amato *et al.*, 2010] Giuseppe Amato, Stefano Chessa, and Claudio Vairo. Mad-wise: A distributed stream management system for wireless sensor networks. *Software: Practice and Experience*, 40(5):431–451, apr 2010.

[Bechkit *et al.*, 2011] Walid Bechkit, Abdelmadjid Bouabdallah, and Yacine Challal. Un prototype de réseaux de capteurs sans fil pour l'agriculture et le contrôle de l'environnement. In *CFIP 2011 - Colloque Francophone sur l Ingénierie des Protocoles*, Sainte Maxime, France, 2011.

[Becker *et al.*, 2010] M. Becker, Bernd-Ludwig Wenning, Carmelita Goerg, Reiner Jedermann, and Andreas Timm-Giel. Logistic applications with wireless sensor networks. In *Processing of the Workshop on Hot Topics in Embedded Networked Sensors (HotEMNETS 2010)*, pages 6–10, Killarney, Ireland, June 2010.

[Belfkih *et al.*, 2015a] Abderrahmen Belfkih, Bruno Sadeg, Claude Duvallet, and Laurent Amanton. A new query processing model for maintaining data temporal consistency in wireless sensor networks. In *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, April 7-9, 2015*, pages 1–6, 2015.

[Belfkih *et al.*, 2015b] Abderrahmen Belfkih, Bruno Sadeg, Claude Duvallet, and Laurent Amanton. Study of timing properties on data collection and query processing techniques in wireless sensor networks. In *SENSORNETS 2015 - Proceedings of the 4rd International Conference on Sensor Networks, Angers, France, 11-13 February, 2015*, pages 77–84, 2015.

[Bhaskar and Sitharama, 2004] Krishnamachari Bhaskar and Iyengar Sitharama. Distributed bayesian algorithms for fault-tolerant event region detection in wireless sensor networks. *IEEE Transactions on Computers*, 53:241–250, 2004.

[Billet, 2015] Benjamin Billet. *Système de gestion de flux pour l'Internet des objets intelligents*. PhD thesis, Université de Versailles Saint-Quentin-En-Yvelines, 2015.

[Bonnet *et al.*, 2001] Philippe Bonnet, Johannes Gehrke, and Praveen Seshadri. Towards Sensor Database Systems. In *Proceedings of the International Conference on Mobile Data Management*, pages 3–14, London, UK, 2001. Springer-Verlag.

[Breheret *et al.*, 2000] Laurent Breheret, Frédéric Schettini, Eric Bernauer, and Magali Barbier. Traitements des données de Trafic. Technical report, Direction de la Recherche et des Affaires Scientifiques et Techniques, 2000.

[Caccamo *et al.*, 2002] M. Caccamo, L.Y. Zhang, Lui Sha, and G. Buttazzo. An implicit prioritized access protocol for wireless sensor networks. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 39–48, 2002.

[Chaiporn *et al.*, 2000] Jaikaeo Chaiporn, Srisathapornphat Chavalit, and Shen Chienchung. Squerying and tasking in sensor networks. In *Proceedings of the International Symposium on Aerospace/Defense Sensing, Simulation, and Control*, pages 184–197, 2000.

[Chen *et al.*, 2010] Deji Chen, Mark Nixon, and Aloysius Mok. Wireless and Real-Time Industrial Process Control. In *WirelessHART™*, pages 201–221. Springer, 2010.

[Chipara *et al.*, 2006] O. Chipara, Z. He, Guoliang Xing, Qin Chen, Xiaorui Wang, Chenyang Lu, J. Stankovic, and T. Abdelzaher. Real-time power-aware routing in sensor networks. In *IEEE International Workshop on Quality of Service*, pages 83–92, June 2006.

[Chipara *et al.*, 2007] O. Chipara, C. Lu, and G. C. Roman. Real-time query scheduling for wireless sensor networks. In *IEEE International Real-Time Systems Symposium (RTSS)*, pages 389–399, Dec 2007.

[Choi *et al.*, 2006] Jae Young Choi, Jongwook Lee, Kamrok Lee, Sunghyun Choi, Wook Hyun Kwon, and Hong Seong Park. Aggregation Time Control Algorithm for Time constrained Data Delivery in Wireless Sensor Networks. In *Proceedings of the Vehicular Technology Conference*, pages 563–567. Spring, 2006.

[Corporation, 2006] Moteiv Corporation. Tmote sky low power wireless sensor module, 2006.

[Corporation, 2015] Oracle Corporation. Mysql 5.1 reference manual, 2015.

[Dawborn and Khoury, 2010] Tim Dawborn and Raymes Khoury. Corona developers guide. Technical report, University of Sydney, 2010.

[Deering and Hinden, 1998] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification, December 1998.

[Draves *et al.*, 2004] Richard Draves, Jitendra Padhye, and Brian Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, pages 114–128, New York, NY, USA, 2004. ACM.

[Dunkels *et al.*, 2004] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Annual IEEE International Conference on Local Computer Networks*, pages 455–462, Tampa, Florida, USA, 2004.

[Dunkels *et al.*, 2007] Adam Dunkels, Fredrik Osterlind, Nicolas Tsiftes, and Zhitao He. Software-based on-line energy estimation for sensor nodes. In *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets '07, pages 28–32, New York, NY, USA, 2007. ACM.

[Dunkels, 2011] Adam Dunkels. The ContikiMAC Radio Duty Cycling Protocol. Technical Report T2011:13, Swedish Institute of Computer Science, dec 2011.

[Duvallet *et al.*, 1999] Claude Duvallet, Zoubir Mammeri, and Bruno Sadeg. Les SGBD temps réel. *Technique et Science Informatiques*, 18(5):479–516, 1999.

[Egea-Lopez *et al.*, 2005] E. Egea-Lopez, J. Vales-Alonso, A. S. Martinez-Sala, P. Pavon-Marino, and J. Garcia-Haro. Simulation tools for wireless sensor networks. *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, pages 2–9, 2005.

[Elias *et al.*, 2012] A.G.F. Elias, J.J.P.C. Rodrigues, L.M.L. Oliveira, and B.B. Zarpelao. A Ubiquitous Model for Wireless Sensor Networks Monitoring. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 835–839, July 2012.

[Elson *et al.*, 2002] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Operating Systems Review*, 36(SI):147–163, 2002.

[Felemban *et al.*, 2006] E. Felemban, Chang-Gun Lee, and E. Ekici. Mmspeed: Multipath multi-speed protocol for qos guarantee of reliability and timeliness in wireless sensor networks. *IEEE Transactions on Mobile Computing*, 5(6):738–754, 2006.

[Fung *et al.*, 2002] Wai Fu Fung, David Sun, and Johannes Gehrke. COUGAR: the network is the database. In *Proceedings of the international conference on Management of data*, pages 621–621, New York, NY, USA, 2002. ACM.

[Ganeriwal *et al.*, 2003] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems*, pages 138–149, New York, NY, USA, 2003. ACM.

[Gehrke and Madden, 2004] Johannes Gehrke and Samuel Madden. Query processing in sensor networks. *IEEE Pervasive Computing*, 3(1):46–55, 2004.

[Govindan *et al.*, 2002] R. Govindan, J. Hellerstein, W. Hong, S. Madden, M. Franklin, and S. Shenker. The sensor network as a database. Technical Report 02-771, Usc information sciences institute, 2002.

[Group, 2013] The PostgreSQL Global Development Group. The world's most advanced open source database, 2013.

[Gürgen *et al.*, 2006] Levent Gürgen, Claudia Roncancio, Cyril Labbé, and Vincent Olive. Transactional issues in sensor data management. In *Proceedings of the Workshop on Data Management for Sensor Networks*, pages 27–32, New York, NY, USA, 2006. ACM.

[Guyoton, 1991] Fabrice Guyoton. *Sismicité et structure lithosphérique des Alpes occidentales*. PhD thesis, Université Joseph Fourier – Grenoble I, 1991.

[Haiying *et al.*, 2004] Zhou Haiying, Mean Hou Kun, Ponsonnaille Jean, Gineste Laurent, and Coudon Julien. Remote continuous cardiac arrhythmias detection and monitoring. *Transformation of Healthcare with Information Technologies*, 5:112–120, 2004.

[HART Communication Foundation, 2014] HART Communication Foundation. WirelessHART. http://fr.hartcomm.org/hcp/tech/wihart/wireless_how_it_works.html, 2014.

[Harte *et al.*, 2005] S. Harte, A. Rahman, and K.M. Razeeb. Fault tolerance in sensor networks using self-diagnosing sensor nodes. *The IEEE International Workshop on Intelligent Environment*, pages 7–12, 2005.

[He *et al.*, 2003] Tian He, John A. Stankovic, Chenyang Lu, and Tarek Abdelzaher. Speed: A stateless protocol for real-time communication in sensor networks. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 46–55, Washington, DC, USA, 2003. IEEE Computer Society.

[Hoeller *et al.*, 2010] N. Hoeller, C. Reinke, J. Neumann, and S. Groppe. Stream-based xml template compression for wireless sensor network data management. In *Multimedia and Ubiquitous Engineering (MUE), 2010 4th International Conference on*, pages 1–9, Aug 2010.

[Iino *et al.*, 2010] Yutaka Iino, Takeshi Hatanaka, and Masayuki Fujita. Dynamic topology optimization for dependable sensor networks. In *IEEE International Conference on Control Applications*, pages 274–279, 2010.

[Jaikaeo *et al.*, 2000] Chaiporn Jaikaeo, Chavalit Srisathapornphat, and Chien-Chung Shen. Sensor information networking architecture. In *Proceedings of the International Workshop on Parallel Processing*, pages 23–30, 2000.

[Jason *et al.*, 2000] Hill Jason, Szewczyk Robert, Woo Alec, Hollar Seth, Culler David, and Pister Kristofer. System architecture directions for networked sensors. *SIGPLAN Notice*, 35(11):93–104, nov 2000.

[Jun Zheng, 2009] Abbas Jamalipour Jun Zheng. *Wireless Sensor Networks: A Networking Perspective*. Wiley-IEEE Press, 1st edition, 9 2009.

[Kanzaki *et al.*, 2010] Akimitsu Kanzaki, Takahiro Hara, Yoshimasa Ishi, Tomoki Yoshihisa, Yuuichi Teranishi, and Shinji Shimojo. X-Sensor: Wireless Sensor Network Testbed Integrating Multiple Networks. In *Wireless Sensor Network Technologies for the Information Explosion Era*, volume 278 of *Studies in Computational Intelligence*, pages 249–271. Springer Berlin Heidelberg, 2010.

[Kellner, 2010] Simon Kellner. Flexible Online Energy Accounting in TinyOS. In *Proceedings of the International Workshop in Real-World Wireless Sensor Networks*, pages 62–73, Colombo, Sri Lanka, 2010. Springer.

[Khoury *et al.*, 2010] Raymes Khoury, Tim Dawborn, Bulat Gafurov, Glen Pink, Edmund Tse, Quincy Tse, K. Almiani, Mohamed Gaber, Uwe Rohm, and Bernhard Scholz. Corona: Energy-efficient multi-query processing in wireless sensor networks. In Hiroyuki Kitagawa, Yoshiharu Ishikawa, Qing Li, and Chiemi Watanabe, editors, *Database Sys-*

tems for Advanced Applications, volume 5982 of *Lecture Notes in Computer Science*, pages 416–419. Springer Berlin Heidelberg, 2010.

[Kitagawa *et al.*, 2010] Hiroyuki Kitagawa, Yousuke Watanabe, Hideyuki Kawashima, and Toshiyuki Amagasa. Stream-based real world information integration framework. In Takahiro Hara, VladimirI. Zadorozhny, and Erik Buchmann, editors, *Wireless Sensor Network Technologies for the Information Explosion Era*, volume 278 of *Studies in Computational Intelligence*, pages 173–204. Springer Berlin Heidelberg, 2010.

[Kofoed, 2007] Leif Morten Kofoed. Enhancing sensor network programming:Extending TinyDB with HAVING and aggregation, and investigating TinyDB reliability. Master's thesis, University of Oslo, Oslo, Norvège, 2007.

[Kreibich, 2010] Jay A. Kreibich. *Using SQLite*. O'Reilly Media, 2010.

[Krishnakumar, 2010] T Krishnakumar. Integrated Routing and Query Processing in Wireless Sensor Networks. *International journal of applied engineering research*, 1(1):115–123, 2010.

[Kushalnagar *et al.*, 2007] N. Kushalnagar, G. Montenegro, and C. Schumacher. 6LoWPAN: Overview, Assumptions, Problem Statement and Goals, February 2007.

[LE, 2008] Hung-Cuong LE. *Optimisation d'accès au médium et stockage de données distribuées dans les réseaux de capteurs*. PhD thesis, Université de franche-comté, 2008.

[Levis *et al.*, 2003] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proceedings of the International Conference on Embedded Networked Sensor Systems*, pages 126–137, New York, NY, USA, 2003. ACM.

[Levis *et al.*, 2011] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. The Trickle Algorithm, rfc 6206, Internet Engineering Task Force (IETF). mar 2011.

[Li *et al.*, 2008] Jinbao Li, Zhipeng Cai, and Jianzhong Li. Data Management in Sensor Networks. In Yingshu Li, MyT. Thai, and Weili Wu, editors, *Wireless Sensor Networks and Applications*, pages 287–330. Springer US, 2008.

[Li *et al.*, 2009] Yanjun Li, Chung Shue Chen, Ye-Qiong Song, Zhi Wang, and Youxian Sun. Enhancing real-time delivery in wireless sensor networks with two-hop information. *Industrial Informatics, IEEE Transactions on*, 5(2):113–122, 2009.

[Locke, 2001] C. Douglas Locke. Applications and system characteristics. In *RealTime Database Systems*, pages 17–26, 2001.

[Lu *et al.*, 2002] Chenyang Lu, Brian M. Blum, Tarek F. Abdelzaher, John A. Stankovic, and Tian He. RAP: A real-time communication architecture for large-scale wireless sensor networks. In *Proceedings of Real-Time and Embedded Technology and Applications Symposium*, pages 55–66, 2002.

[Macedo *et al.*, 2009] Mário Macedo, António Grilo, and Mário Nunes. Distributed latency-energy minimization and interference avoidance in {TDMA} wireless sensor networks. *Computer Networks*, 53(5):569 – 582, 2009.

[Madden *et al.*, 2003a] Madden, Hellerstein Joe, and Hong Wei. *TinyDB: In-Network Query Processing in TinyOS, version 0.4*, 2003.

[Madden *et al.*, 2003b] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the international conference on Management of data*, pages 491–502, New York, NY, USA, 2003. ACM.

[Madden *et al.*, 2005] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, mar 2005.

[Madden, 2009] Samuel Madden. In-network query processing. In *Encyclopedia of Database Systems*, pages 1538–1543. Springer US, 2009.

[Makhoul, 2008] Abdallah Makhoul. *Réseaux de capteurs : localisation, couverture et fusion de données*. PhD thesis, Université de Franche-Comté, 2008.

[Mall, 2009] Rajib Mall. *Real-Time Systems: Theory and Practice*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2009.

[Mammeri, 1998] Zoubir Mammeri. Expression et dérivation des contraintes temporelles dans les applications temps réel. *Journal Européen des Systèmes Automatisés, JESA-APII,*, 32(5-6):609–644, 1998.

[Maroti *et al.*, 2004] Miklos Maroti, Branislav Kusy, Gyula Simon, and Akos Ledeczi. The flooding time synchronization protocol. In *Proceedings of the International ACM Conference on Embedded Networked Sensor Systems*, pages 39–49, Baltimore, MD, USA, 2004. ACM.

[Mazzer and Tourancheau, 2008] Yannis Mazzer and Bernard Tourancheau. Réseaux de micro-contrôleurs à faible consommation d'énergie embarquant des capteurs : Premières expériences et développement d'une nouvelle interface paramétrable et programmable. Rapport technique 6450, INRIA, 2008.

[Mouradian *et al.*, 2014] Alexandre Mouradian, Isabelle Augé-Blum, and Fabrice Valois. RTXP: A localized real-time mac-routing protocol for wireless sensor networks. *Computer Networks*, 67:43–59, 2014.

[Noh *et al.*, 2008] Kyoung-Lae Noh, Erchin Serpedin, and Khalid A. Qaraqe. A new approach for time synchronization in wireless sensor networks: Pairwise broadcast synchronization. *IEEE Transactions on Wireless Communications*, 7(9):3318–3322, 2008.

[Noël and Servigne, 2004] Guillaume Noël and Sylvie Servigne. Po-Tree: un système d'indexation spatio-temporel temps-réel . In Cassini, editor, *Conférence CASSINI-SIGMA 2004 : Géomatique et Analyse Spatiale*, pages 120–121, jun 2004.

[Noël and Servigne, 2005] Guillaume Noël and Sylvie Servigne. Indexation multidimensionnelle de bases de données capteur temps-réel et spatiotemporelles. *Ingénierie des Systèmes d'Information*, 10(4):59–88, oct 2005.

[Noël, 2006] Guillaume Noël. *Indexation dans les bases de données capteurs temps réel*. PhD thesis, Institut National des Sciences Appliquées de Lyon, 2006.

[Olivier Hersent, 2014] Omar Elloumi Olivier Hersent, David Boswarthick. *Formation des réseaux 6LowPan : RPL et MLE*. Dunod, Mai 2014.

[Österlind *et al.*, 2006] Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-Level Sensor Network Simulation with COOJA. In *LCN 2006, The 31st Annual IEEE Conference on Local Computer Networks*, pages 641–648, Tampa, Florida, USA, November 2006.

[Ozdemir and Xiao, 2009] Suat Ozdemir and Yang Xiao. Secure Data Aggregation in Wireless Sensor Networks: A Comprehensive Overview. *Computer Networks*, 53(12):2022–2037, August 2009.

[Paradis and Han, 2007] Lilia Paradis and Qi Han. A survey of fault management in wireless sensor networks. *J. Netw. Syst. Manage.*, 15(2):171–190, June 2007.

[Patil and Patil, 2011] Nandini S. Patil and P. R. Patil. Data aggregation in wireless sensor network. *International Journal Of Service Computing And Computational Intelligence*, 1(1):7–10, 2011.

[Perkins *et al.*, 2003] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing, July 2003.

[Pignagnoli *et al.*, 2011] L. Pignagnoli, F. Chierici, P. Favali, L. Beranzoli, D. Embriaco, S. Monna, , F. D'Oriano, and N. Zitellini. Tsunami early warning system: Deep sea measurements in the source area. *Marine Research at CNR*, 2011.

[Pinto *et al.*, 2013] P. Pinto, A. Pinto, and M. Ricardo. End-to-end delay estimation using rpl metrics in wsn. In *Wireless Days (WD), 2013 IFIP*, pages 1–6, Nov 2013.

[Radoi *et al.*, 2012] Ion Emilian Radoi, Aditi Shenoy, and D. K. Arvind. Evaluation of routing protocols for internet-enabled wireless sensor networks. In *ICWMC 2012, The International Conference on Wireless and Mobile Communications*, pages 56–61, 2012.

[Rahman and Hussain, 2007] Md. Ashiqur Rahman and Sajid Hussain. Energy efficient query processing in wireless sensor network. In *AINA Workshops*, pages 696–700, 2007.

[Ramamritham *et al.*, 2004] Krithi Ramamritham, Sang H. Son, and Lisa Cingiser Dipippo. Real-time databases and data services. *Real-Time Syst.*, 28(2-3):179–215, November 2004.

[Ramamritham, 1993] Krithi Ramamritham. Real-Time Databases. *Distrib. Parallel Databases*, 1(2):199–226, April 1993.

[Rezayat *et al.*, 2009] P. Rezayat, M. Mahdavi, M. Ghasemzadeh, and M. AghaSarram. A novel real-time routing protocol in wireless sensor networks. In *Current Trends in Information Technology (CTIT), 2009 International Conference on the*, pages 1–6, 2009.

[Sadeg and Saad-Bouzefrane, 2000] Bruno Sadeg and Samia Saad-Bouzefrane. Relaxing correctness criteria in real-time dbmss. In *Proceedings of the ISCA 15th International Conference Computers and Their Applications, March 29-31, 2000, New Orleans, Louisiana, USA*, pages 64–67, 2000.

[Sadeg, 2004] Bruno Sadeg. *Contributions a la gestion des transactions dans les SGBD temps réel.* Habilitation à diriger des recherches en informatique, Université du Havre, Novembre 2004.

[Saifullah *et al.*, 2011] Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. End-to-End Communication Delay Analysis in WirelessHART Networks. Technical Report WUCSE-2011-86, All Computer Science and Engineering Research, Washington, 2011.

[Sarkar, 2012] Subir Kumar Sarkar. *Wireless Sensor and Ad Hoc Networks Under Diversified Network Scenarios.* Artech House, 2012.

[Servigne and Noël, 2008] Sylvie Servigne and Guillaume Noël. Real time and spatiotemporal data indexing for sensor based databases , 2008.

[Sharaf *et al.*, 2003] Mohamed A. Sharaf, Jonathan Beaver, Alexandros Labrinidis, and Panos K. Chrysanthis. Tina: A scheme for temporal coherency-aware in-network aggregation. In *International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDe '03, pages 69–76, New York, NY, USA, 2003. ACM.

[Shelby *et al.*, 2014] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP), June 2014.

[Shen *et al.*, 2001] Chien-Chung Shen, C. Srisathapornphat, and C. Jaikaeo. Sensor information networking architecture and applications. *Personal Communications, IEEE*, 8(4):52–59, Aug 2001.

[Shnayder *et al.*, 2004] Victor Shnayder, Mark Hempstead, Bor-Rong Chen, and Matt Welsh. PowerTOSSIM: Efficient Power Simulation for TinyOS Applications. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Baltimore, Maryland, USA, 2004.

[Srisathapornphat *et al.*, 2001] Chavalit Srisathapornphat, Chaiporn Jaikaeo, and Chien chung Shen. Sensor information networking architecture and applications. *IEEE Personal Communications*, 8:52–59, 2001.

[Stankovic and Ramamritham, 1989] John A. Stankovic and K. Ramamritham, editors. *Tutorial: Hard Real-time Systems.* IEEE Computer Society Press, Los Alamitos, CA, USA, 1989.

[Stankovic *et al.*, 1999] J.A. Stankovic, Sang Hyuk Son, and J. Hansson. Misconceptions about real-time databases. *Computer*, 32(6):29–36, Jun 1999.

[Suriyachai *et al.*, 2010] Petcharat Suriyachai, James Brown, and Utz Roedig. Time-critical data delivery in wireless sensor networks. In *Distributed Computing in Sensor Systems*, pages 216–229. Springer, 2010.

[Tan *et al.*, 2006] Wee Lum Tan, OnChing Yue, and Wing Cheong Lau. Performance Evaluation of Differentiated Services Mechanisms Over Wireless Sensor Networks. In *Vehicular Technology Conference*, pages 1–5, Sept 2006.

[Tarannum, 2010] Suraiya Tarannum. Energy conservation challenges in wireless sensor networks: A comprehensive study. *Wireless Sensor Network*, 2(6):483–491, 2010.

[Trigoni *et al.*, 2007] Niki Trigoni, Alexandre Guitton, and Antonios Skordylis. *Learning from Data Streams: Processing Techniques in Sensor Networks.* Springer, November 2007.

[Tsiftes and Dunkels, 2011] Nicolas Tsiftes and Adam Dunkels. A database in every sensor. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, pages 316–332, New York, NY, USA, 2011. ACM.

[Tsiftes *et al.*, 2009] Nicolas Tsiftes, Adam Dunkels, Zhitao He, and Thiemo Voigt. Enabling large-scale storage in sensor networks with the coffee file system. In *Proceedings*

of the International Conference on Information Processing in Sensor Networks, pages 349–360, Washington, DC, USA, 2009. IEEE Computer Society.

[Val *et al.*, 2008] Thierry Val, Eric Campo, and Adrien Van Den Bossche. Technologie zigbee / 802.15.4 protocoles, topologies et domaines d'application. *Techniques de l'ingénieur*, TE7508, 2008.

[Vasseur *et al.*, 2012] JP. Vasseur, M. Kim, K. Pister, N. Dejean, and D. Barthel. Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks, mar 2012.

[Vucinic *et al.*, 2013] Malisa Vucinic, Bernard Tourancheau, and Andrzej Duda. Performance comparison of the RPL and LOADng routing protocols in a Home Automation scenario. In *WCNC*, pages 1974–1979, 2013.

[Watteyne *et al.*, 2006] Thomas Watteyne, Isabelle Augé-Blum, and Stéphane Ubéda. Dual-mode real-time mac protocol for wireless sensor networks: A validation/simulation approach. In *Proceedings of the First International Conference on Integrated Internet Ad Hoc and Sensor Networks*. ACM, 2006.

[Winter *et al.*, 2012] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPV6 Routing Protocol for Low-Power and Lossy Networks, mar 2012.

[Woo *et al.*, 2003] Alec Woo, Terence Tong, and David Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 14–27. ACM, 2003.

[Xiong and Ramamritham, 2004] Ming Xiong and Krithi Ramamritham. Deriving deadlines and periods for real-time update transactions. *IEEE Trans. Comput.*, 53(5):567–583, may 2004.

[Yao and Gehrke, 2002] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31:9–18, sep 2002.

[Yutaka *et al.*, 2009] Iino Yutaka, Hatanaka Takeshi, and Fujita Masayuki. Event-predictive control for energy saving of wireless networked control system. In *Proceedings of the American Control Conference*, pages 2236–2242, Piscataway, NJ, USA, 2009.

[ZHU, 2013] Nanhao ZHU. *Simulation and Optimization of Energy Consumption on Wireless Sensor Networks*. PhD thesis, École Centrale de Lyon, 2013.

Résumé : Dans ce travail, nous nous focalisons sur l'ajoût de contraintes temporelles dans les Bases de Données de Capteurs Sans Fil (BDCSF). La cohérence temporelle d'une BDCSF doit être assurée en respectant les contraintes temporelles des transactions et la validité temporelle des données, pour que les données prélevées par les capteurs reflètent fidèlement l'état réel de l'environnement. Cependant, les retards de transmission et/ou de réception pendant la collecte des données peuvent conduire au non respect de la validité temporelle des données. Les travaux réalisés dans cette thèse portent principalement sur trois contributions : (1) nous faisons une étude comparative des propriétés temporelles entre la collecte périodique des données et une approche de traitement des requêtes utilisant une BDCSF, nommée "TinyDB", (2) nous proposons un modèle de traitement des requêtes temps réel, qui utilise deux nouvelles clauses SQL et un algorithme de contrôle de la validité temporelle des données, (3) nous avons conçu une BDCSF temps réel, basée sur les techniques décrites dans la deuxième contribution. Elle implémente des contrôleurs d'échéances et des ordonnanceurs temps réel. Nous avons testé le système avec des requêtes périodiques et non périodiques, qui a montré de meilleurs performances durant les simulations que l'approche "TinyDB", largement utilisée dans la recherche sur les BDCSF.

Mots-clé: Réseau de capteurs sans fil, base de données de capteurs, contraintes temporelles, système temps réel, langage de requêtes.

Abstract: In this thesis, we are interested in adding real-time constraints in the Wireless Sensor Networks Database (WSNDB). Temporal consistency in WSNDB must be ensured by respecting the transaction deadlines and data temporal validity, so that sensor data reflect the current state of the environment. However, delays of transmission and/or reception in a data collection process can lead to not respect the data temporal validity. Besides, the concurrent accesses to data by user queries (i) generate both an overload in the base station and delays in the responses, and (ii) weakened the system's ability to provide accurate answers to users, that reflect the actual state of the environment. The work carried out in this thesis focus on three contributions: (1) we have done a comparative study of temporal properties between (i) a periodic data collection and (ii) a query processing approach using sensor database, named "TinyDB", (2) we have proposed a new real-time query processing model. It uses two new SQL clauses and a control algorithm to ensure data temporal validity, and (3) we have designed a real-time sensor database, based on the techniques described in the second contribution. It implements deadline controllers and real-time schedulers to assign priorities to the transactions. We tested the system with periodic and non periodic transactions, which showed better performances than TinyDB approach, which is largely used in research area of WSNDB.

Keywords: Wireless sensor network, sensor database, time constraints, real-time system, query language.