SOLIST : Structure multi-couche pair-à-pair à faible consommation pour les réseaux de capteurs sans-fil

Yann Busnel

RenPar'18 / SympA'2008 / CFSE'6
Fribourg, Suisse, du 11 au 13 février 2008

SOLIST : Structure multi-couche pair-à-pair à faible consommation pour les réseaux de capteurs sans-fil.

Yann Busnel* (IRISA / Université de Rennes 1)

Institut de Recherche en Informatique et Systèmes Aléatoires,
Campus Universitaire de Beaulieu,
35042 Rennes Cedex - France – Yann.Busnel@irisa.fr

Résumé
Les réseaux de capteurs sans-fil (RCsF) connaissent depuis quelques années un intérêt croissant, tant dans la recherche qu dans l'industrie. Ceux-ci ouvrent une nouvelle voie aux applications réparties dans des milieux jusqu'alors impraticables, les capteurs s'auto-organisant eux-même afin de fournir des fonctionnalités telles que la collecte ou la dissémination d'information sur et dans le réseau, le routage de messages à des capteurs spécifiques, *etc.* En raison de leur petite taille, les ressources disponibles sur un capteur sont très limitées, et la gestion d'énergie devient donc une préoccupation de premier ordre.
Le cœur des systèmes de gestion de données dans les systèmes répartis repose sur certaines fonctionnalités de bases telles que la diffusion ou la recherche de capteurs particuliers. Dans cet article, nous proposons la mise en œuvre décentralisée d'une collection de primitive appelées *-cast (anycast, k-cast et broadcast). Pour cela, nous présentons et évaluons SOLIST, une structure multi-couche, largement inspirée des réseaux pair-à-pair structurés, limitant la consommation d'énergie dans le cadre des RCsF. Un type est associé à chaque capteur, et les primitives de *-cast sont mises en œuvre à la granularité des types, sans pour autant nécessiter une connaissance de ces types *à priori*, ni de leur répartition dans le réseau. Nous avons évalué SOLISTpar simulation et montré que cette structure fournit les primitives de *-cast en atteignant un bon compromis entre performance et consommation d'énergie.

Mots-clés : Réseaux de capteurs, réseaux structurés, primitive de diffusion, économie d'énergie

1. Introduction

Rechercher une information précise dans un réseau large-échelle non-structuré revient à chercher une aiguille dans une botte de foin sans détecteur de métaux ! La recherche de fichiers rares dans les systèmes de partage de fichier pair-à-pair (PàP) illustre parfaitement cette propriété. Alors que l'inondation est déconseillé dans les réseaux filaires, elle se révèle inexploitable dans les réseaux de capteurs sans-fil (RCsF), pour lesquels l'économie d'énergie est prédominante. En effet, les RCsF sont composés d'un grand nombre de petites entités ne possédant pas les mêmes ressources qu'un micro-ordinateur usuel. En raison de leur petite taille, ces objets, équipés d'un module de communication sans-fil et appelés *capteurs* par la suite, ne disposent que de faibles ressources (en terme de mémoire, processeur, batterie, *etc.*). La consommation d'énergie est donc une préoccupation de premier ordre dans la conception de ces réseaux.
Contrairement aux systèmes répartis classiques, les RCsF sont déployés et configurés en général pour les besoins d'une application spécifique, telle que la surveillance, l'inventaire, l'agrégation de données, *etc.* Cependant, certaines fonctionnalités de base sont communes à la plupart de ces applications. Nous avons identifié un ensemble de primitives de communication, celles-ci correspondant en quelques sortes aux briques de base de celles-ci. Nous appellerons **-cast* cet ensemble, composé des primitives de broadcast, anycast et k-cast. La primitive d'*anycast* consiste à trouver un nœud quelconque dans un sous-ensemble spécifique du réseau. La primitive de k-*cast* consiste a joindre, sans distinction, k nœuds d'un sous-ensemble du réseau. Enfin, la primitive de *broadcast* nécessite de contacter tous les nœuds de ce

* Travaux réalisés en collaboration avec Marin Bertier (IRISA / INSA Rennes) et Anne-Marie Kermarrec (INRIA Rennes – Bretagne Atlantique)

sous-sensemble. Par exemple, considérons une application d'inventaire, où chaque capteur représente un objet physique d'un type donné. Envoyer un message à tous les capteurs d'un tel type ou interroger le système sur la présence d'un objet d'un type spécifique sont des taches usuelles, issues de l'utilisation de la collection *-cast.

Nous considérons ici un système dans lequel chaque capteur possède un type associé. Dans cet article, nous proposons SOLIST (*Self-Organized Large-scale and lightweight Information-based Sensor Technology*), un réseau structuré multi-couches à faible consommation d'energie pour les RCsF incluant une implémentation efficace de la collection *-cast en terme d'économie d'énergie et de fiabilité[2]. SOLIST propose donc une interface générique pour les applications dans le contexte des RCsF (groupement par type et collection *-cast).

Afin de mettre en oeuvre les primitives de la collection *-cast, SOLIST se fonde sur une structure multi-couches légère. Les capteurs sont groupés ensemble, par type, au sein de sur-couches spécifiques. Les primitives de broadcast et k-cast sont mises en œuvre au niveau de ces sur-couches. SOLIST propose également une primitive efficace d'anycast, laquelle peut être utilisée seule pour les besoins de l'application, ou pour déterminer un point d'entrée dans une sur-couche particulière.

Cet article est organisé comme suit : Les sections 2 et 3 introduisent respectivement le modèle du système et l'interface générique. La structure de SOLIST est introduite dans la section 4. Étant donnée la contrainte d'espace de cet article, nous ne pouvons développer les détails de mise en œuvre de la collection *-cast dans SOLIST, mais celle-ci sera abordée succinctement en fin de section (paragraphe 4.4). Différents scénarios ont été simulés et comparés avec des mécanismes traditionnels (marche aléatoire et inondation). Nous présentons un sous-ensemble des résultats d'évaluation dans la section 5 avant de conclure en section 6. Une description plus précise des mises en œuvres, des évaluations et de l'état de l'art est disponible dans [2].

2. Pré-requis du système

Nous considérons un réseau composé de n capteurs sans-fil, répartis dans un espace géographique donnée. Bien qu'aucune hypothèse ne soit imposée sur la répartition des nœuds, la topologie doit assurer la connectivité du réseau. Nous ne considérons que des capteurs non-mobiles, mais ceux-ci peuvent devenir indisponible, en raison d'une défaillance ou de la consommation complète de leur batterie. De même, dans ce cas, l'hypothèse de connectivité doit rester valide, le système ne supportant pas le partitionnement du réseau. Chaque capteur ne peut communiquer qu'en émission à 1 saut avec les nœuds situés dans son rayon de transmission, via un canal de communication idéal (*i.e.* sans collision, ni perte de message). De plus, le temps de propagation d'un message entre deux nœuds du réseau doit être borné, cette borne étant connue.

Chaque nœud connaît sa position géographique relative dans un système de cordonnées virtuelles, ainsi que la taille de ce système. Nous ne proposons pas dans ces travaux une méthode de construction de ce système de coordonnées, mais, étant donnée un RCsF statique, les coordonnées relatives peuvent être calculées localement lorsque qu'un nœud rejoint le réseau, comme proposé dans [6, 9]. Aucun ensemble de type pré-défini n'est requis.

Afin de découvrir son voisinage, tout nœud diffuse localement et périodiquement un message d'existence. Dans la nécessité d'obtenir une communication multi-saut, tout protocole de routage géographique peut être utilisé sur le réseau, s'il vérifie qu'un nœud peut envoyer des messages à un autre nœud, ne connaissant que sa position dans le système de coordonnées virtuelles. Dans le cadre de nos expérimentations, le protocole GPSR [5] a été utilisé, permettant notamment de trouver efficacement le nœud le plus proche de coordonnées virtuelles données.

3. La collection *-cast

Une grande majorité des applications réparties reposent sur les mêmes fonctionnalités. Plus spécifiquement, dans les RCsF, il est souvent nécessaire de contacter des nœuds en fonction de leur catégorie. SOLIST fourni un ensemble de fonctionnalités de base, appelés collection *-cast et reposant sur une structure de groupe. A un temps donnée, chaque nœud possède un *type* représentant son état. Celui-ci peut

[2] Notre définition de la *fiabilité* correspond à la résolution actualisée d'une requête de la collection *-cast.

être soit *statique* (si le réseaux est composé de capteurs hétérogènes, par exemple, chaque forme de capteur sera lié à un identifiant particulier pour toute la durée de l'application), soit *dynamique* (représentant le niveau d'énergie restante, ou des valeurs de données captés). Dans le premier cas, la valeur du type est fixée de façon pérenne pour toute la durée de l'application, contrairement au second, dans lequel cette valeur peut changer éventuellement au cours de l'exécution. De plus, un nœud peut ne pas avoir de type (temporairement ou non) et ne participera alors qu'à la connectivité globale et au routage sur le réseaux. SOLIST assure que tous les nœuds de même type sont regroupé ensemble dynamiquement.
Nous avons identifié les trois briques de base suivante :

Primitive d'anycast ANYCAST(type) ;
Cette primitive permet d'obtenir le contact d'un nœud parmi tous les nœuds d'un type *donné.*
Par exemple, cette fonctionalité peut être utilisée afin de vérifier si au moins une instance d'un type donné existe dans le système, et dans l'affirmative, de localiser une de celles-ci.

Primitive de k**-cast** KCAST(type, k) ;
Cette primitive permet de contacter k *nœud d'un* type *donné, s'il en existe suffisamment dans le réseau. Si le groupe contient plus de* k *nœuds, la primitive renverra* VRAI*, ainsi que des informations optionnelles dépendantes de l'application, et* FAUX *sinon.*
Par exemple, un gestionnaire de stock a communément besoin de savoir si une marchandise est disponible en quantité suffisante. La primitive k-cast devient alors essentielle dans une application d'inventaire. Une autre application classique des RCsF est la surveillance de feus de forêt. Il peut être utile de connaître si le réseau contient au moins 10 capteurs ayant mesuré une température supérieure à 40 degrés Celsius ou une hygrométrie supérieure à 2 %vol. Dans ce cas, une valeur moyenne sur un échantillon de nœuds d'un type donné pourrait être requise. La primitive k-cast répond idéalement à ce type de demande.

Primitive de broadcast BROADCAST(type) ;
Cette primitive permet de contacter tous les nœuds d'un type *donné. Des informations optionnelles sur les noeuds de cet ensemble peuvent être attaché au message réponse, telles que le nombre de nœuds de ce* type*, la moyenne de leur valeur mesurée,* etc.
Dans SOLIST, la primitive de broadcast diffuse un message uniquement aux nœuds du même type. Ce service peut donc être vu comme du *multicast*. Ce type de fonctionnalité peut être utilisé pour disséminer de l'information à un ensemble de nœuds de type donné comme dans une application de publication/abonnement par exemple. Il peut également être utilisé pour comptabiliser le nombre d'entités d'un même type disponibles dans un stock.

Chaque nœud possède deux primitives supplémentaires :

Rejoindre le système JOIN(type) ;
Lors de l'arrivée d'un nœud dans un RCsF existant, celui-ci peut contacter un nœud appartenant déjà au réseau, et exécuter des tâches classiques d'initialisation, d'anonce de sa présence à son voisinage direct, de connexion à d'éventuelles structure, *etc.* Plus de détails sur cette primitive sont disponibles ci-après.

Quitter le système LEAVE(type) ;
Lors du départ déterminé d'un nœud, celui-ci doit effectuer une procédure de départ afin d'éviter les incohérences potentielles des structures, mettre à jour l'information sur ses voisins, *etc.* Si un nœud quitte le réseau sans exécuter cette procédure, il sera considéré comme défaillant.

Dans le cas de types dynamiques, à chaque changement de type d'un nœud, ce dernier commencera par quitter le réseau afin de quitter son groupe actuel, puis de revenir avec un type mis à jour et rejoindre un groupe différent, si besoin est.
SOLIST peut etre utilisé dans le contexte de nombreuses applications, comme mentionné ci-avant. Par exemple, nous pouvons citer les applications classiques suivantes :

Surveillance Dans ces applications, les utilisateurs ont besoin de connaître combien de capteurs sont dans un état donné, à un temps donné. Par l'utilisation du groupement dynamique (des capteurs dans le même état) et de la collection *-cast dans ces groupes, nous pouvons obtenir la plupart des informations sans recourir à l'utilisation d'un langage de requête particulier comme dans [7] ;

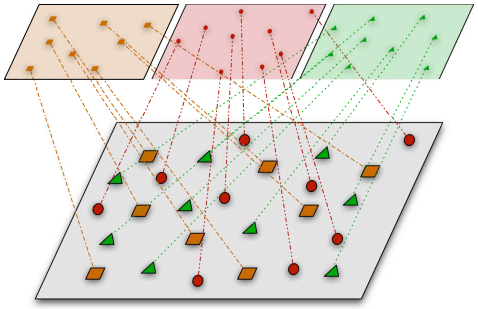

FIG. 1 – Projection des groupes en sur-couche

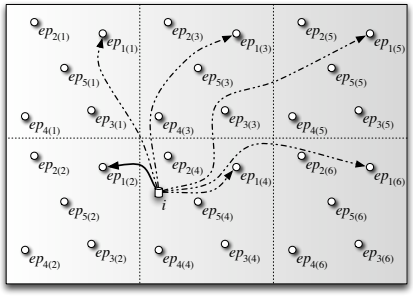

FIG. 2 – Exemple d'une division de l'espace en 3×2 cellules.
Le nœud i *envoi sa requête de recherche de* LIGH-1-LAYER *au point d'entrée le plus proche :* $ep_{1(2)}$

Gestion de stock Les applications de ce type doivent permettre au gestionnaire d'obtenir dynamiquement la réponse à trois questions fondamentales : y-a-t-il un élément d'un type donné dans le stock ? Y-en a-t-il en quantité suffisante ? Combien d'éléments exactement y-a-t-il ? La collection *-cast répond directement à ces questions (respectivement les primitives d'anycast, de k-cast et de broadcast) ;

Nous pouvons également citer quelques utilisations génériques de SOLIST :

Diffusion Cette application est inhérente à la collection *-cast. Les primitives de k-cast et de broadcast fournissent respectivement les fonctionnalités de *multicast* bornée ou non. Pour contacter différents groupes de nœuds dans le réseau, il suffit d'émettre une requête de broadcast pour chacun de ces groupes.

Agrégation En utilisant la primitive de broadcast avec réponse présentée ci-dessus, les informations disponibles sur tout un groupe peuvent être collectées et agrégées simplement vers l'émetteur de la requête.

4. Fonctionnement de SOLIST

Dans cette partie, nous présentons l'architecture de SOLIST.

4.1. Une structure multi-couche

Dans l'optique de fournir efficacement la collection *-cast, SOLIST se fonde sur une structure multi-couches.
Premièrement, tous les nœuds font partie d'une couche de base, utilisée pour assurer la connectivité globale du réseau et le routage géographique[3]. Cette couche sera appelée *couche de routage* par la suite. Elle permet de contacter une destination quelconque en utilisant uniquement ses coordonnées virtuelles, présentées dans la partie 2.
De plus, au-dessus de cette couche de routage, SOLIST met en œuvre différentes sur-couches logiques légères (une par groupe existant dans le réseau). La structure de ces sur-couches est présentée ci-après, en section 4.2. Cette structure multi-couche fourni un groupement logique des noeuds du réseau. Par exemple, dans la figure 1, n nœuds de trois types différents sont disposés dans un RCsF. À chacun des types données correspond une sur-couche spécifique (*i.e.* celle des triangles, des ronds et des carrés), au-dessus de la couche de routage. Cette dernière est utilisée pour les communications entre deux nœuds d'une même sur-couche, situés hors de portée directe.

[3] Le protocole de routage utilisé dans notre implémentation est une version allégée de GPSR [5]

4.2. Structure d'une sur-couche : LIGH-t-LAYER

SOLIST fourni également une structure de sur-couche à faible consommation énergétique, par découpage rectangulaire de l'espace. Nous avons observé précédemment de nombreuses similitudes entre les RCsF et les systèmes PàP, en terme de propriété et de fonctionnalités [1]. Notre structure de sur-couche s'est donc naturellement fondée sur les propositions de réseaux logiques structurés PàP, notamment les tables de hachage distribuées classiques telles que CAN [11]. Dans [3], Castro *et al.* propose une comparaison des réseaux structurés PàP dans le contexte du *multicast*. La structure de CAN nous est apparue comme idéale pour les primitives identifiées des RCsF. CAN divise l'espace en zones logiques de responsabilité, réparties sur les différents nœuds du système. Chaque nœud possède alors un voisinage virtuel au sein de cette structure, correspondant aux nœuds responsables des zones adjacentes à celle du nœud donné. Inspirés par cette technique, nous avons modifié sensiblement la structure et la construction de CAN afin de l'adapter au contexte des RCsF. Les principales différences résident dans les contraintes de communication et d'économie d'énergie, qui doivent impérativement être prises en compte. Nous n'avons donc conservé que le découpage logique de l'espace. Les coordonnées virtuelles d'un nœud restent les mêmes quelque soit la couche considérée[4] (couche de routage et sur-couche de groupe). Les zones de responsabilités d'un nœud ne sont utilisées que pour l'obtention d'une implémentation efficace des primitives de k-cast et broadcast. Nous appellerons ces sur-couches LIGH-t-LAYER par la suite[5].

Ces sur-couches sont construites graduellement et dynamiquement. Le premier nœud d'un type t' donné rejoignant le réseau devient *responsable* de la totalité de l'espace du LIGH-t'-LAYER correspondant. A l'arrivée d'un autre nœud de type t' dans le réseau, ce dernier va contacter le nœud responsable de la zone contenant ses coordonnées, en utilisant un routage glouton dans le LIGH-t'-LAYER. Cette dernière zone sera alors divisée en deux, chacun des deux nœuds devenant responsable de la partie correspondante à ses coordonnées, à l'instar de CAN. Une nouvelle frontière est dès lors créée entre les deux zones. Celle-ci sera estampillée par un identifiant incrémental, dans l'objectif de conserver l'ordonnancement de la création de frontière. Ce dernier permet de réorganiser la structure en cas de départ ou de défaillance d'un nœud, comme indiqué ci-dessous. En résumé, à chaque arrivée d'un nœud, le nœud responsable de la zone correspondante la divise en deux et créé une nouvelle frontière estampillée entre le nouveau nœud et lui-même.

Chaque nœud appartenant à une sur-couche spécifique doit maintenir une liste de voisin appelée *vue* dans le LIGH-t-LAYER correspondant. Chaque entrée de cette vue contient (1) l'identifiant du voisin logique ; (2) ses coordonées dans SOLIST ; (3) les coordonnées des extrémités de la frontière commune ; et (4) l'estampille de cette frontière.

La figure 3 présente les différents cas d'évolution de la structure d'un LIGH-t-LAYER à partir de sa création. Le nœud A de type t rejoint le réseau en premier, et devient donc responsable de la totalité de l'espace du LIGH-t-LAYER. Dans la figure 3.*a*, B contacte A, qui divise sa zone en deux et envoi à B, les coordonnées de la zone de ce dernier ainsi que celles de leur frontière commune. Puis, à l'arrivée du nœud C, B divise à son tour sa propre zone, et estampille la nouvelle frontière par un entier strictement plus grand, comme l'illustre la figure 3.*b*. Après plusieurs arrivées, la figure 3.*c* présente l'état de la structure du LIGH-t-LAYER avec 9 nœuds (identifiés de A à I). La plus grande estampille de frontière est 5.

La partie inférieure de la figure 3 présente comment conserver la cohérence de la structure en cas de départ (figure 3.*d*) ou de défaillance (figure 3.*e*). Un départ ou une défaillance est traité de la même façon par le système. Quand un nœud quitte le LIGH-t-LAYER (sur un départ définitif ou un changement de type), celui-ci envoi sa vue au nœud situé de l'autre coté de la frontière la plus récente. Ce dernier devient alors responsable de l'union des deux zones, met à jour sa propre vue (avec des nouveaux voisins potentiels ou des allongements de frontières) et finalement, informe les voisins concernés du changement d'état de la structure.

Par exemple, dans la figure 3.*d*, le nœud C doit quitter le LIGH-t-LAYER. C envoi donc sa vue à I, qui se situe derrière la frontière d'estampille 3 (3 étant la plus grande estampille). I devient donc responsable des deux zones (celles de I et de C). I met à jour sa vue en étendant sa frontière commune avec B, et,

[4] Les fonctionnalités de hachage et de répartition uniforme des nœuds par le calcul de nouvelles coordonnées logiques dans CAN sont coûteuses et ne sont pas nécessaire dans SOLIST.

[5] t correspond à l'identifiant du type des nœuds regroupés dans cette sur-couche.

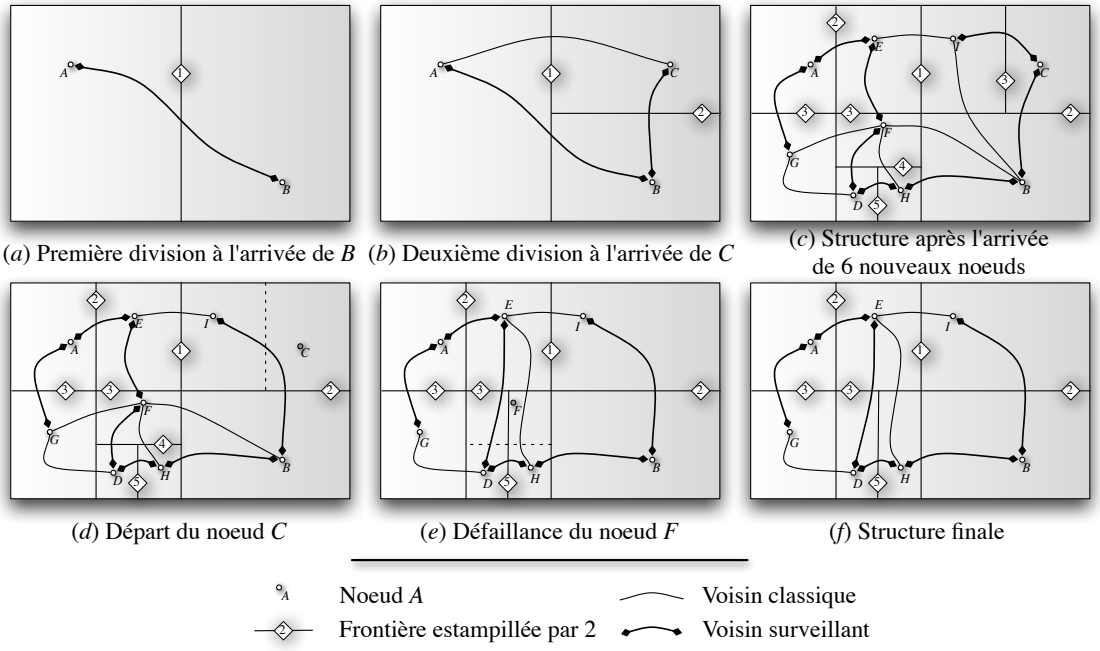

FIG. 3 – Evolution de la structure du LIGH-t-LAYER suivant différents évenements.

enfin, prévient ce dernier de cette modification.

La détection de défaillance est effectuée par un mécanisme de surveillance commune, selon les hypothèses du modèle, décrites dans la partie 2. Le temps maximal de propagation d'un message étant connu, il ne peut pas y avoir de fausse suspicion. A chaque étape d'arrivée dans le LIGH-t-LAYER, les deux nœuds impliqués dans la division d'une zone initient une surveillance symétrique. Périodiquement, chaque nœud du LIGH-t-LAYER envoi un message *jeSuisVivant!* à son (ou ses) surveillant(s). Si celui-ci n'en recoit pas pendant un laps de temps déterminé, il vérifie l'état de son voisin par un message *toujoursEnVie?*. S'il n'obtient toujours aucune réponse, le surveillant considère le nœud comme fautif et le supprime du LIGH-t-LAYER.

En cas de défaillance du nœud F comme dans la figure 3.*e* par exemple, celui-ci doit être retiré de SOLIST. F ne peut évidemment pas envoyer sa vue à un de ses voisins, comme le nécessite la procédure de départ volontaire. Dans ce cas, D, qui est un des nœuds surveillants de F, envoie un message *reconstitutionDeVue* qui va tourner autour de la zone de F, afin de reconstruire la vue de ce dernier à partir des informations disponibles sur chacun de ses voisins. Avec cette vue ainsi reconstituée, D exécute la procédure standard de départ, contactant chaque nœud au-delà de la frontière la plus récente (dans ce cas, H et lui-même), et ceux-ci allongent leur zone avec la partie correspondante de l'ancienne zone de F. La figure 3.*f* présente l'état de la structure du LIGH-t-LAYER après l'exécution de ces départs.

4.3. Relier les deux mondes : Les points d'entrée

Chaque LIGH-t-LAYER étant autonome et indépendant en terme de fonctionnement de chacun des autres LIGH-t-LAYER ainsi que de la couche de routage, il reste le problème de lier toutes ces couches les unes aux autres. Comment atteindre, à partir de n'importe quel nœud du RCsF, un LIGH-t-LAYER spécifique sans autre information que l'identifiant t du type voulu ?

Nous avons calqué l'espace de coordonnée virtuelle sur la couche de routage (*cf.* partie 2), et utilisons celui-ci comme proposé dans des travaux récent [10]. Nous utilisons deux fonctions de hachage communes à tous les nœuds du système. Le but de ces fonctions est de répartir des *points d'entrées virtuels* dans l'espace de coordonnées, uniformément en fonction du nombre de type existant. Soit t un identifiant de type donné, pour lequel un nœud recherche le LIGH-t-LAYER correspondant ; $f_x : \mathbb{N} \mapsto [0; 1[$ et

$f_y : \mathbb{N} \mapsto [0;1[$ les deux fonctions de hachage ; et wsX, wsY la taille respectivement horizontale et verticale de l'espace de coordonnées. Les coordonnées du point d'entrée correspondant au LIGH-t-LAYER (nommés ep_t) sont calculés de la manière suivante :

$$ep_t = (x, y) \text{ TQ } \begin{cases} x &= f_x(t) \times wsX \\ y &= f_y(t) \times wsY \end{cases} \quad (1)$$

Comme f_x et f_y sont communes à tous les nœuds du réseaux, les coordonnées des points d'entrées sont uniques pour tout type. Un point d'entrée est donc représenté par ses coordonnées, identique pour tous les nœuds. Malheureusement, il est probable qu'aucun nœud physique ne se trouve exactement au point correspondant à ces coordonnées. Le protocole de routage géographique permettant de déterminer le nœud physique le plus proche d'un point donné dans l'espace virtuel (*cf.* partie 2), nous appellerons dès lors indistinctement *point d'entrée* le point de coordonnées ep_t obtenues par l'équation 1 et le nœud le plus proche de celui-ci. Ce dernier doit connaître au moins un nœud de type t (et donc, connaître l'existence du LIGH-t-LAYER). Si ce LIGH-t-LAYER existe, ce nœud doit être en mesure de donner l'identifiant et les coordonnées d'un nœud de ce LIGH-t-LAYER. Par la suite, les nœuds référencés sur les points d'entrée seront appelés *nœud de contact*.

La disposition de ces points d'entrées dans l'espace peut amener un nœud excentré à envoyer une requête vers un point d'entrée situé à l'exacte opposée du réseau. Afin d'éviter qu'un message n'ait à traverser entièrement le réseau et de diminuer la charge des nœuds les plus proches d'un point d'entrée, l'espace des coordonnées est divisé selon une grille : m divisions horizontales (par rapport à la coordonnée x) et n divisions verticales (par rapport à la coordonnée y). L'espace des coordonnées est alors calqué sur chacune de ces sous-divisions, appelés *cellules* par la suite, mais uniquement dans le cadre du calcul des coordonnées des points d'entrées. Lorsqu'un nœud doit atteindre un LIGH-t-LAYER, il envoi sa requete au point d'entrée le plus proche de lui, lequel lui renverra les informations d'un nœud de contact dans le LIGH-t-LAYER.

La figure 2 présente une topologie $m \times n = 3 \times 2$ contenant 5 types déclarés. Pour chaque cellule, les 5 points d'entrées $(et_1, \ldots, et_5)$ sont répliqués à la même position relative. Dans ce cas, le nœud i émet une requête afin d'accéder au LIGH-1-LAYER (sur-couche du groupe de type 1). i connaît, grâce aux fonctions de hachages communes, les coordonnées des 6 points d'entrées (flèches pleine et pointillées), mais ne transmet sa requête qu'au plus proche d'entre eux (ici, $ep_{1(2)}$ représenté par une flèche pleine). Plus formellement, soit $d(\cdot,\cdot) : (\mathbb{R} \times \mathbb{R})^2 \mapsto \mathbb{R}$ la distance entre deux points de l'espace de coordonnées sur la couche de routage, et csX, csY la taille des cellules respectivement horizontales et verticales. L'équation 2 présente le calcul pour trouver le point d'entrée de type t le plus proche. Soit les deux ensembles suivants :

$$\begin{cases} \Gamma_{x,t} = \{(k_x + f_x(t)) \times csX | k_x \in [0..m-1]\} \\ \Gamma_{y,t} = \{(k_y + f_y(t)) \times csY | k_y \in [0..n-1]\} \end{cases}$$

Les coordonnées du point d'entrée le plus proche sont les suivantes :

$$ep_t = (x, y) \quad (2)$$

$$\text{TQ} \quad d(i,(x,y)) = \min_{\substack{x' \in \Gamma_{x,t} \\ y' \in \Gamma_{y,t}}} d(i,(x',y'))$$

Afin de mettre à jour dynamiquement les informations disponibles sur les points d'entrées, à chaque arrivée d'un nœud dans un LIGH-t-LAYER, celui-ci en informe le point d'entrée de type t le plus proche de sa position. Ainsi, ce point d'entrée connaît le nœud de contact le plus proche de lui, sans avoir a interroger le LIGH-t-LAYER. Cela permet à l'initiateur d'une requête d'obtenir un nœud de contact qui ne soit pas trop éloigné. De même, en cas de départ ou de défaillance, les points d'entrées sont informés de ce départ afin de mettre à jour leurs informations sur les nœuds de contact. Néanmoins, comme le choix du nœuds de contact se fait localement sur chaque point d'entrée, un nœud n'est pas informé s'il a été choisi comme nœud de contact. Tout départ d'un type t donné doit donc être notifié à tous les points d'entrée de ce type.

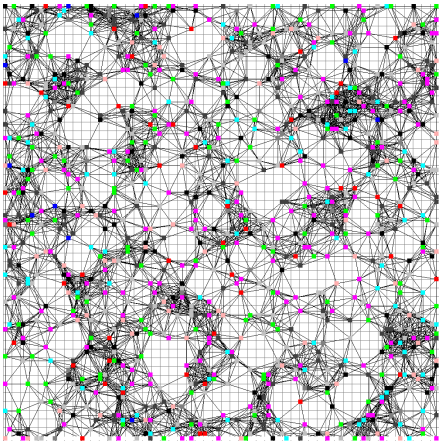

FIG. 4 – Exemple d'une topologie réseau avec 1000 nœuds et 10 types.

Opération	nAh
Transmission d'un paquet	20,000
Reception d'un paquet	8,000
Ecoute radio durant une 1 ms	1,250
Lecture de données	1,111
Ecriture/Suppression de données	83,333

FIG. 5 – Consommation d'énergie nécessaire pour la réalisation de diverses opérations de bases sur un capteur MICA, proposé dans [8].

4.4. La collection *-cast dans SOLIST

En raison de la contrainte d'espace, nous ne pouvons présenter en détail la réalisation des primitives de la collection *-cast dans SOLIST. Dans [2], nous proposons un algorithme d'anycast, fondé sur le mécanisme de recherche d'un LIGH-t-LAYER et des points d'entrées correspondants. Nous introduisons également un algorithme de broadcast non-redondant à faible consommation d'énergie, utilisant la structure des LIGH-t-LAYER, ainsi qu'un algorithme de k-cast déterministe correspondant à un parcours en profondeur de l'arbre de diffusion dans un LIGH-t-LAYER (une approche probabiliste est proposée mais non évaluée dans [2] à cause à la fiabilité relative de ses résultats). Un court état de l'art et une comparatif avec les travaux relatifs sont également disponibles dans [2].

5. Evaluations

Dans cette partie, nous présentons l'environnement de simulation, les protocoles utilisés à des fins de comparaison et enfin, l'évaluation de SOLIST.

5.1. Environnement de simulation

Dans l'optique d'évaluer SOLIST, nous avons utilisé SeNSim [12], un simulateur d'application pour RCsF, développé par l'équipe-projet ASAP/INRIA Rennes – Bretagne Atlantique.

Nous avons simulé plusieurs topologies de réseaux différentes, respectant le modèle défini préalablement (*cf.* partie 2). Nous avons utilisé une version allégée du protocole de routage géographique GPSR : le calcul des *graphes planaires* étant trop coûteux, et désireux d'évaluer le coût énergétique de SOLIST, notre version de GPSR ne contient que le routage glouton avec la *règle de la main droite* en présence d'un trou de densité sur une route[6].

Un exemple de topologie utilisée est présenté en figure 4 : un réseau de 1 000 nœuds, répartis parmi 10 types statiques (10 nœuds de type 1, 30 de type 2, 50 de type 3, ..., 190 de type 10), dans un espace de 96×96 mètres carré. Pour chacun des scénarios présenté ci-après, chaque nœud arrive dans le réseau, émet une requête de k-cast et une de broadcast (et donc, par définition, 3 requêtes d'anycast), avant de finalement quitter le réseau. Les dates d'arrivée, de départ et de requête sont générés aléatoirement, ainsi que les valeurs de t et de k (ces dernières sont piochés respectivement dans $[1;12]$ et $[1;200]$ afin d'obtenir des résultats hétérogènes). Chaque simulation a une durée de 10 000 temps discrets.

[6] La contrainte d'espace ne nous permet pas de développer ce point - *cf.* [5] pour plus de détails.

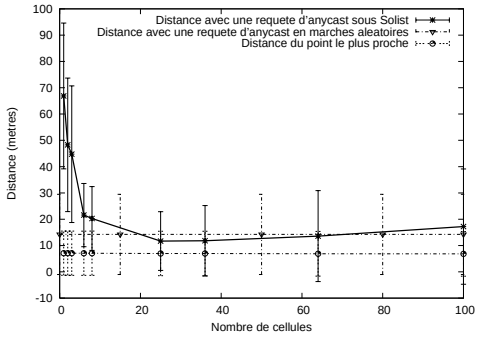

FIG. 6 – Distance moyenne entre l'émetteur d'un anycast et le nœud de contact correspondant.

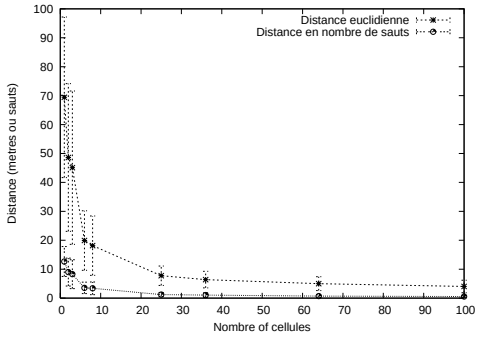

FIG. 7 – Distance moyenne entre l'émetteur d'une requête d'anycast et le point d'entrée le plus proche.

5.2. Présentation des algorithmes de comparaison

Afin d'évaluer objectivement SOLIST, il est nécessaire de comparer ses résultats avec l'existant. Étant donnée que l'interfacage générique et la structuration de RCsF n'ont pas été proposé de cette facon (*cf.* [2]), nous avons utilisés deux protocoles classiques afin de comparer deux des trois membres de la collection *-cast : la *marche aléatoire* pour la primitive anycast, et l'*inondation* pour celle de broadcast. Divers autres algorithmes de broadcast ont été proposés tels que LPBcast [4] mais ceux-ci sont probabilistes, contrairement à celui proposé dans cet article.

Nous avons évalué SOLIST selon deux grands axes présentés dans les paragraphes suivants : la fiabilité des primitives de la collection *-cast et la consommation d'énergie.

5.3. Fiabilité des primitives

Nous avons évalué la primitive d'anycast selon deux métriques. Tout d'abord, la figure 6 présente la distance moyenne et l'écart type de celle-ci entre l'émetteur de la requête et le nœud de contact correspondant, en fonction du nombre de cellules dans SOLIST. La borne inférieure (*Distance du point le plus proche*) correspond à la distance moyenne entre l'émetteur de la requête et le nœud le plus proche du type requis. La distance moyenne par l'utilisation de marches aléatoires est également présentée ici. Pour de grande cellules dans SOLIST, correspondant à un nombre restreint de cellules, les nœuds de contact sont situés assez loin de l'émetteur, ceux-ci étant les plus proches des points d'entrée et non de l'émetteur. Mais, plus la taille des cellules diminuent, plus les nœuds de contact sont proches. Un effet de bord peut être observé pour un très grand nombre de cellule, la courbe d'anycast dans SOLIST remontant légèrement. En effet, lorsqu'un nœud rejoint un LIGH-t-LAYER, celui-ci informe le point d'entrée le plus proche de son arrivée. À partir d'une forte diminution de la taille des cellules, d'autres points d'entrée peuvent être plus proche de ce nouveau nœud que de tous les autres de ce LIGH-t-LAYER, mais ils ne seront pas informés de cette arrivée. Nous avons préféré ne pas inonder les points d'entrée à chaque arrivée afin de limiter la consommation d'énergie globale du système.

D'autre part, la figure 7 présente la distance moyenne et l'écart type de celle-ci entre tout nœud du système et le point d'entrée le plus proche, en fonction du nombre de cellules dans SOLIST. La distance euclidienne n'est pas seulement représentée comme en figure 6, mais accompagnée du nombre de saut nécessaire pour contacter un point d'entrée. Cette figure montre que pour les topologies données, à partir de 6 cellules dans SOLIST, le nombre moyen de sauts est inférieur à 3 sauts, illustrant la basse consommation d'énergie nécessaire à l'obtention d'un nœud de contact.

Concernant les primitives de broadcast et k-cast, la fiabilité des résultat est simple à analyser, correspondant à un comportement binaire : Si une requête entraîne l'obtention d'une réponse, alors respectivement tous ou k nœuds ont été contactés. Sinon, aucune réponse ne sera renvoyé. Ce dernier cas n'apparaît que rarement et uniquement si le LIGH-t-LAYER est en cours de maintenance (départ ou défaillance d'un nœud). Considérant le système en régime stable, le taux de succès d'une requête de k-cast

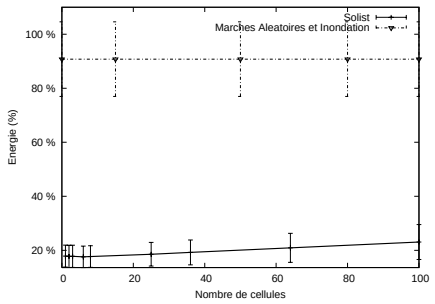

FIG. 8 – Consommation moyenne d'énergie en fin de simulation.

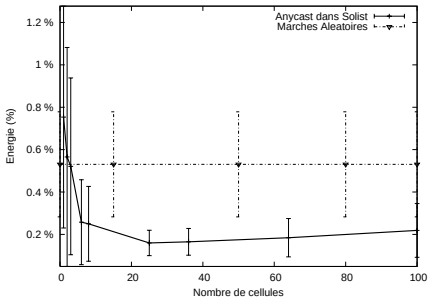

FIG. 9 – Consommation moyenne d'énergie pour la primitive de anycast.

ou de broadcast est de 100 %. En cas de maintenance d'un LIGH-t-LAYER, l'avancement de la primitive sera retardé en attendant un retour en régime stable. Le seul cas d'échec d'une requête advient donc uniquement lorsqu'un nœud est défaillant et que son nœud surveillant ne l'a pas encore détecté. Dans ce cas, la requête sera transférée et la réponse attendue indéfiniment. Un délai de garde, ou des accusés de réception de requête, peuvent être instaurés pour éviter ce cas de figure. Ces optimisations n'ont pas été incluses dans les simulations présentées dans cet article.

5.4. Consommation d'énergie

Nous nous intéressons maintenant à la consommation d'énergie nécessaire au fonctionnement de SOLIST, comparée avec celle nécessaire à l'utilisation de marches aléatoires et d'inondation. Afin de comparer équitablement ces approches, nous avons effectués les simulations en imitant une consommation réelle d'énergie, proposé dans [8], dont les chiffres, issus de mesures sur des capteurs MICA réels, sont fournis dans le tableau 5. Chaque capteur possède au début de l'expérimentation une batterie chargé de capacité 2,200 mAh.

En premier lieu, intéressons-nous à la consommation totale du système au cours d'une simulation. La figure 8 présente, pour différents nombre de cellules dans SOLIST, la consommation d'énergie moyenne en fin de simulation. Cette figure illustre l'efficacité de SOLIST en terme d'économie d'énergie, comparativement aux marche aléatoires combiné avec l'inondation. Nonobstant, la courbe de consommation de SOLIST remonte sensiblement pour un grand nombre de cellules. Ceci est du à la croissance de l'énergie nécessaire à la maintenance du système, comme nous le verrons ci-après.

La figure 9 présente pour les mêmes configuration que précédemment la consommation moyenne d'énergie dans l'utilisation de la primitive d'anycast. Celle-ci montre l'intérêt de SOLIST à partir d'une configuration à 3 cellules, en comparaison avec les marche aléatoires. Néanmoins, avec une configuration à très petites cellules, le nœud de contact reçu ne corresponds pas forcement au plus proche, rehaussant légèrement la courbe énergétique en fonction du nombre de cellules. Cependant, l'énergie nécessaire pour l'anycast dans SOLIST reste inférieure à 35 % de celle utilisé en moyenne avec des marches aléatoires.

Concernant la primitive de broadcast, la figure 10 présente l'énergie pour les mêmes configurations de SOLIST, comparativement à l'utilisation de l'inondation. Etant donnée que la structure des LIGH-t-LAYER ne sont pas corrélés avec le nombre de cellules dans SOLIST, la consommation d'énergie moyenne est constante pour chaque courbe. Comme attendu, cette figure illustre l'intérêt de ne contacter que les nœuds nécessaires à la diffusion sur un type donné plutôt que d'impliquer tous les nœuds du réseau.

Enfin, la consommation d'énergie moyenne dédiée à la maintenance de la cohésion de la structure est présenté en figure 11 en fonction du nombre de cellules dans SOLIST. Sachant que les primitives d'arrivée et de réorganisation en cas de départ ou de défaillance sont gérées localement, l'énergie nécessaire reste logiquement réduite quelque soit la configuration (moins de 0,25 % de l'énergie totale disponible sur un nœud). En contrepartie, la primitive de départ demande une part de plus en plus importante d'énergie à mesure que le nombre de cellule augmente. Ceci est conséquence de l'accroissement linéaire du nombre de messages de départ envoyés aux points d'entrées, pour la mise à jour sur ces derniers de

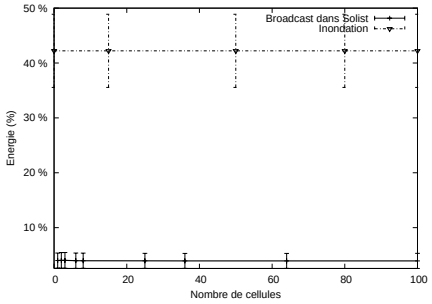

FIG. 10 – Consommation moyenne d'énergie pour la primitive de broadcast.

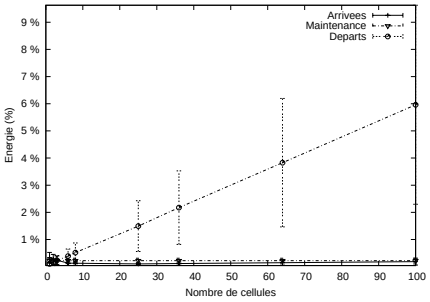

FIG. 11 – Consommation moyenne d'énergie pour la maintenance de la structure.

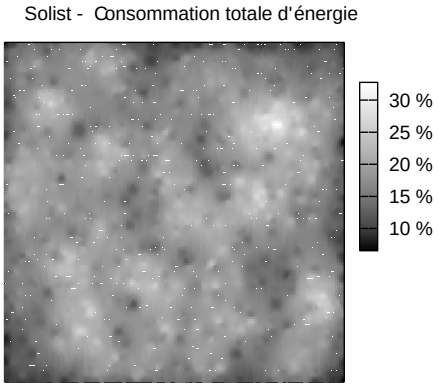

FIG. 12 – Consommation d'énergie à la fin de la simulation avec SOLIST pour la topologie présentée en figure 4 avec une configuration à 2×4 cellules.

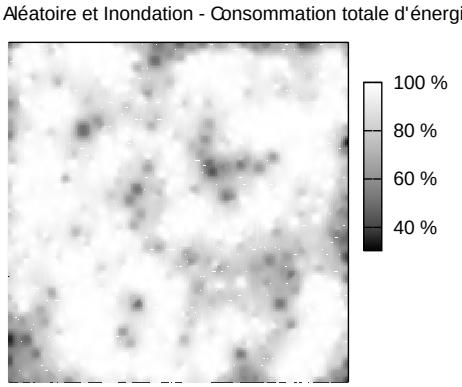

FIG. 13 – Consommation d'énergie à la fin de la simulation avec marches aléatoires et inondations pour la topologie présentée en figure 4.

l'information concernant les nœuds de contact. Par exemple, avec une configuration à 10×10 cellules, durant la simulation, chaque nœud devra quitter le réseau ou subir une défaillance. 100 000 messages de départ sont alors générés à travers le réseau uniquement pour la maintenance de la structure.
À titre d'exemple, les figures 12 et 13 présentent l'état du réseau en terme d'énergie consommée à la fin d'une simulation, sur la topologie présentée en figure 4, respectivement pour une utilisation de SOLIST avec une configuration à 2×4 cellules et pour une utilisation de marches aléatoires et inondations. Elles permettent de se rendre compte de l'équilibrage de charge sur le réseau. La majorité des nœuds souffre d'une pénurie d'énergie sur la figure 13. Au contraire, l'utilisation de SOLIST sur le même scénario, avec 1 000 requêtes de k-cast supplémentaires, montre une consommation d'énergie maximale inférieure à 33 %. Nous pouvons toutefois observer des zones à forte densité. Les nœuds de ces zones ont en effet été plus sollicités en raison d'un plus grand nombre de messages ayant transités – ou été générés – par ceux-ci au cours de la simulation.
Ces résultats montrent que SOLIST permet de devancer les approches classiques utilisées ici, en terme de consommation d'énergie. Pour la topologie de réseau utilisée dans ces évaluations (1 000 nœuds répartis en 10 types), la configuration à 8 cellules est idéale pour obtenir le meilleur compromis entre efficacité et consommation d'énergie.

6. Conclusion

Dans cet article, nous avons proposé SOLIST, une solution générique *tout-en-un* pour fournir l'implémentation de la collection *-cast (*i.e.* anycast, k-cast et broadcast) dans les RCsF statiques. Celle-ci est une architecture de système, faible consommation, pour les RCsF large-échelle. SOLIST est composé d'un ensemble fini de sur-couches (les LIGH-t-LAYER) permettant de mettre en œuvre cette interface commune, fondé sur le groupement de nœud de même type. Une mise en œure efficace de la collection *-cast en terme de fiabilité des résultats et de consommation d'énergie est fournie.
SOLIST a été évalué par simulation pour chacune des fonctionnalités fournies. Nous avons comparé SOLIST avec d'autres algorithmes classiques : les marches aléatoire pour l'anycast et l'inondation par arbre de diffusion pour le broadcast. Les résultats de cette étude illustrent la forte économie d'énergie due à l'utilisation de SOLIST par rapport à des algorithmes standards.

Bibliographie

1. Yann Busnel, Marin Bertier, Eric Fleury, et Anne-Marie Kermarrec. GCP : Gossip-based Code Propagation for large-scale mobile wireless sensor networks. In *The First ACM International Conference on Autonomic Computing and Communication Systems (Autonomics'07)*, Roma, Italy, Oct. 2007.
2. Yann Busnel, Marin Bertier, et Anne-Marie Kermarrec. SOLIST : A lightweight multi-overlay structure for wireless sensor networks. Rapport de recherche, INRIA, Rennes, France, Oct. 2007.
3. Miguel Castro, Michael B. Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, et Alec Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (Infocom'03)*, San Francisco, CA, USA, Fév. 2003.
4. Patrick Eugster, Sidath Handurukande, Rachid Guerraoui, Anne-Marie Kermarrec, et Petr Kouznetsov. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4) :341–374, 2003.
5. Brad Karp et H. T. Kung. GPSR : Greedy Perimeter Stateless Routing for Wireless Networks. In *ACM/IEEE International Confrence on Mobile Computing and Networking (MobiCom 2000)*, Boston, MA, USA, Août 2000.
6. YoungMin Kwon, Kirill Mechitov, Sameer Sundresh, Wooyoung Kim, et Gul Agha. Resilient localization for sensor networks in outdoor environments. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 643–652, Colombus, OH, USA, Juin 2005.
7. Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, et Wei Hong. TinyDB : an acquisitional query processing system for sensor networks. *ACM Transactions on Database System*, 30(1) :122–173, 2005.
8. A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, et J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *The First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, pages 88–97, Atlanta, GE, USA, Sep. 2002.
9. R. O'Dell et Roger Wattenhofer. Theoretical aspects of connectivity-based multi-hop positioning. *Theoretical Computer Science*, 344 :47–68, Juin 2005.
10. Sylvia Ratnasamy, Deborah Estrin, Ramesh Govindan, Brad Karp, Scott Shenker, Li Yin, et Fang Yu. Data-Centric Storage in Sensornets. In *Proceedings of the 2002 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'02)*, New York, NY, USA, 2002.
11. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, et Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'01)*, pages 161–172, Août 2001.
12. Simulateur SeNSim. `http ://sensim.gforge.inria.fr/`, ASAP Research Project, INRIA Rennes, FR, 2005-2007.