

Routage par marche aléatoire à listes tabous

Karine Altisen, Stéphane Devismes, Pascal Lafourcade, Clément Ponsonnet

▶ **To cite this version:**

Karine Altisen, Stéphane Devismes, Pascal Lafourcade, Clément Ponsonnet. Routage par marche aléatoire à listes tabous. 13es Rencontres Francophones sur les Aspects Algorithmiques de Télécommunications (AlgoTel), May 2011, Cap Estérel, France. pp.14. hal-00588086

HAL Id: hal-00588086
https://hal.science/hal-00588086

Submitted on 22 Apr 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Routage par marche aléatoire à listes tabous †

K. Altisen[1], S. Devismes[1], P. Lafourcade[1] et C. Ponsonnet[1]

[1] *Université de Grenoble,* VERIMAG*, France, Email: firstname.lastname@imag.fr*

Nous proposons d'améliorer les marches aléatoires en utilisant des listes tabous. Notre objectif est de réduire le nombre de sauts pour atteindre un nœud particulier du réseau sans compromettre les autres avantages des marches aléatoires. Nos solutions permettent ainsi de résoudre efficacement le routage dans les réseaux de capteurs sans fil.

Keywords: Réseaux de capteurs sans fil, routage, marche aléatoire, liste tabou

1 Introduction

Une *marche aléatoire* distribuée consiste à router, de manière séquentielle et probabiliste, un message jusqu'à sa destination finale. Les *réseaux de capteurs sans fil* (WSN, pour Wireless Sensor Network) sont des réseaux multi-sauts, souvent à large échelle, où des nœuds, dont les ressources (mémoires, puissance de calcul, batterie) sont limitées, communiquent par le biais de radios.

Ici, nous utilisons les marches aléatoires pour résoudre un type particulier de routage, où un nœud appelé *puits* est le seul destinataire des messages. Les marches aléatoires sont particulièrement adaptées pour le routage dans les WSNs. En effet, elles ne nécessitent pas le maintien d'une infrastructure distribuée (*e.g.*, un arbre couvrant). De plus, les nœuds n'ont pas besoin de connaissances *a priori* sur le réseau (nombre de nœuds, diamètre, *etc.*). Ensuite, le coût en termes de calcul et de mémoire pour les nœuds est faible. Enfin, la taille des messages utilisés est elle-aussi faible. Cependant, tous ces avantages sont contrebalancés par le fait qu'un message routé *via* une marche aléatoire nécessite un grand nombre de sauts pour atteindre sa destination. Par exemple, en utilisant la marche aléatoire classique, notée `RW`, où le prochain saut à partir du nœud v est choisi de manière aléatoire uniforme parmi l'ensemble $\mathcal{N}_v$ de ses voisins, le temps d'atteinte (c'est-à-dire, le nombre moyen de sauts pour atteindre une destination) est en $O(n^3)$, où n est le nombre de nœuds du réseau. Cependant, cette complexité peut être améliorée. En effet, un choix aléatoire uniforme ne prend pas en compte les différences de degré des nœuds. Ainsi, en utilisant `RW`, un nœud v de degré δ_v sera en moyenne plus visité qu'un nœud u de degré δ_u si $\delta_v > \delta_u$. À partir de ce constat, Yamashita *et al* [SIY09] ont proposé une marche aléatoire, notée `YRW`, où la probabilité $P(u)$ (*cf.* figure 1) d'aller du nœud v à son voisin u suit une loi non-uniforme prenant en compte les degrés des nœuds. Cette loi qui favorise les nœuds de faibles degrés permet d'obtenir un temps d'atteinte en $O(n^2)$.

$$P(u) = \frac{\delta_u^{-1/2}}{\sum_{w \in \mathcal{N}_v} \delta_w^{-1/2}}$$

FIGURE 1: $P(u)$.

Dans cet article, nous proposons d'améliorer le temps d'atteinte des marches aléatoires en utilisant des listes tabous. Les listes utilisées seront de petite taille (une constante indépendante de n) afin de ne pas compromettre les avantages inhérents aux marches aléatoires. Nos solutions permettent ainsi de résoudre efficacement (en termes d'énergie) le routage tous-vers-1 dans les WSNs.

2 Algorithmes

Nous présentons plusieurs algorithmes de marche aléatoire utilisant des listes tabous. Nos algorithmes supposent des réseaux bidirectionnels, identifiés et connexes où il existe un unique *puits* ; les autres nœuds sont appelés *sources*. Les liens de communications sont supposés fiables mais asynchrones. Dans nos algorithmes, les choix probabilistes sont effectués en modifiant la loi utilisée dans `YRW` pour qu'elle prenne en compte les listes tabous. Nos algorithmes sont classés en deux familles, selon que les listes sont stockées dans le message (`TLM`) à router ou au niveau des nœuds (`TLCN`). Pour chaque famille, nous dérivons plusieurs algorithmes, qui diffèrent par la politique de mise à jour de la liste et par la borne sur la taille

† Ce travail a été financé par le projet ANR ARESA2.

$\forall u \in \mathcal{N}_v \setminus L, P^v_{\texttt{TLM}}(L)(u) = \dfrac{\delta_u^{-1/2}}{\sum_{w \in \mathcal{N}_v \setminus L} \delta_w^{-1/2}}$ $\forall u \in \mathcal{N}_v \cap L, P^v_{\texttt{TLM}}(L)(u) = 0$	$P^v_{\texttt{TLCN}}(Counter)(u) = \dfrac{Counter[u]^{-1} \times \delta_u^{-1/2}}{\sum_{w \in \mathcal{N}_v} Counter[w]^{-1} \times \delta_w^{-1/2}}$

FIGURE 2: Lois de probabilité utilisées dans `TLM` et `TLCN`.

de la liste. Ainsi, nous noterons, par exemple, `TLM` (*s*,`update`) un algorithme de la famille `TLM` où la liste contient au plus *s* éléments et la mise à jour se fait au moyen de la fonction `update`.

La mise à jour d'une liste *L*, de taille maximale *s* avec un élément *e* est notée `update(`*e*, *L*, *s*`)` . Nous imposons que chaque liste ne contienne que des éléments distincts. Deux politiques de mise à jour sont étudiées :

- `FIFO` : lorsque l'on souhaite ajouter un élément *e* à *L*, il est ajouté à la fin de la liste. Cependant, si *e* est déjà dans la liste, on supprime au préalable son occurrence de la liste. Sinon, si la liste est pleine, le plus ancien élément de la liste est supprimé avant l'insertion de *e*.
- `RAND` : dans ce cas, l'ordre de la liste est sans importance. Lorsque l'on souhaite ajouter *e* à *L*, *e* est inséré directement dans *L*, s'il n'y est pas déjà. Suite à cet ajout, si *L* a plus de *s* éléments, un élément est choisi de manière aléatoire uniforme (peut-être *e*) pour être supprimé de *L*.

Dans les algorithmes `TLM`, une liste tabou est stockée dans chaque message. Elle contient une partie des nœuds déjà visitée par le message. Autant que possible le message essaie d'éviter de revisiter les nœuds présents dans sa liste. Ainsi, nous définissons la probabilité de visiter le voisin *u* à partir du nœud *v* par la fonction $P^v_{\texttt{TLM}}(L) : \mathcal{N}_v \to [0;1]$ où $\mathcal{N}_v$ est l'ensemble des voisins de *v* et *L* est la liste tabou (*c.f.*, la partie gauche de la figure 2). Noter que si $L = \emptyset$, $P^v_{\texttt{TLM}}(L)$ est la loi de probabilité utilisée par `YRW`. Si $\mathcal{N}_v \subseteq L$ (la liste tabou contient tous les voisins de *v*), $P^v_{\texttt{TLM}}(L)$ n'est pas défini : nous utilisons la loi de `YRW` à la place.

À partir de la loi $P^v_{\texttt{TLM}}(L)$, nous avons défini l'algorithme `TLM` (*s*,`update`). Le code de `TLM` pour les nœuds sources est fourni dans l'algorithme 1, le code pour le puits est trivial. Dans cet algorithme, lorsqu'un nœud source *v* a une nouvelle donnée à router (lignes 1-3), cette donnée est encapsulée dans un message. Dans ce message, une liste est initialisée avec *v* (le premier nœud visité). Puis, le message est envoyé vers un voisin de *v* choisi par $P^v_{\texttt{TLM}}(\emptyset)$. Quand un noeud *v* reçoit un message du noeud *u* (lignes 4-9), il sélectionne le noeud suivant vers lequel envoyer le message, met à jour la liste avec *v*, puis envoie le message. Le noeud suivant est tiré aléatoirement parmi les voisins de *v* qui ne sont pas présents dans la liste tabou du message *L* en utilisant la loi de probabilité $P^v_{\texttt{TLM}}(L)$. Si tous les voisins sont déjà dans la liste, le noeud suivant est choisi aléatoirement parmi tous les voisins de *v* en utilisant $P^v_{\texttt{TLM}}(\emptyset)$. Lorsque le message atteint le puits, celui-ci délivre la donnée à la couche applicative.

Dans les algorithmes `TLCN`, chaque nœud *v* dispose d'une liste tabou $L[u]$ par voisin *u*. La liste $L[u]$ contient des identités de messages ayant été envoyés de *v* vers *u*. Nous maintenons les listes de telle sorte qu'un nœud n'apparaisse au plus que dans une seule liste tabou : lorsqu'un nœud source *v* reçoit un message *m*, *v* choisit la nouvelle destination de *m*, supprime *m* de la liste où il était précédemment si une telle liste existe, puis met à jour la liste correspondant à la destination de *m*. Ainsi, lorsque l'identifiant d'un message *m* apparaît dans la liste $L[u]$ d'un nœud *v*, cela signifie que la dernière fois que *v* a reçu *m*, il l'a renvoyé vers *u*. Ceci permet à un nœud *v* de détecter qu'un message *m* a suivi un cycle : lorsque *v* reçoit *m* de *z* et que *m* est présent dans $L[u]$, *v* détecte que *m* vient de suivre un cycle contenant les liens (z,v) et (v,u). Nous utilisons cette information pour privilégier les liens sur lesquels peu de cycles ont été détectés. Pour cela, chaque nœud *v* associe un compteur $Counter[u]$ à chacun de ses voisins *u*. Le compteur $Counter[u]$ cumule le nombre de fois où *v* a détecté qu'un message avait suivi un cycle contenant le lien (v,u). Ainsi, nous définissons une nouvelle loi de probabilité, $P^v_{\texttt{TLCN}}(Counter)$ (*c.f.*, partie droite de la figure 2), de telle sorte que chaque nœud *v* choisisse de préférence un voisin *u* avec une valeur de compteur faible.

À partir de la loi $P^v_{\texttt{TLCN}}(Counter)$, nous avons défini l'algorithme `TLCN` (*s*,`update`). Le code de `TLCN` pour les nœuds source est fourni dans l'algorithme 1, le code pour le puits est trivial. Dans cet algorithme, les compteurs sont initialisés à 1 pour éviter les divisions par zéro. Ensuite, lorsqu'un nœud source *v* a une nouvelle donnée *m* à router, il encapsule *m* dans un message étiqueté par $tag = (v, id)$ où *id* est un numéro de séquence qui identifie le message parmi tous les messages envoyés par le noeud *v*. Ensuite, *v* choisit la

destination u du message en utilisant $P^v_{\texttt{TLCN}}(Counter)$. Enfin, $L[u]$ est mise à jour avec l'identifiant (v,id).

Lorsqu'un nœud source v reçoit un message $\langle m,tag\rangle$ du nœud u (lignes 12-15), il met à jour ses compteurs. Si le message était déjà présent dans une liste $L[z]$, v incrémente $Counter[u]$ et $Counter[z]$ et tag est supprimé de $L[z]$. Puis, la liste du prochain destinataire du message est mise à jour. Ce dernier est choisi aléatoirement suivant $P^v_{\texttt{TLCN}}$. Lorsque le puits reçoit un message, il délivre la donnée à la couche applicative.

Algorithme 1 Algorithmes TLM et TLCN pour les nœuds sources

```
TLM (s,update), code for every source node v
 1: On request to route m
 2:     randomly select next ∈ N_v using probability law P^v_TLM(∅)
 3:     send ⟨m,[v]⟩ to next
 4: Upon receiving ⟨m,L⟩ from u
 5:     if N_v \ L = ∅ then
 6:         randomly select next ∈ N_v using probability law P^v_TLM(∅)
 7:     else
 8:         randomly select next ∈ N_v \ L using probability law P^v_TLM(L)
 9:     update (v, L, s) ; send ⟨m,L⟩ to next
```

```
TLCN (s,update), code for every source node v
 1: Variables :
 2:     id : integer
 3:     L[i] : array of lists defined for all i ∈ N_v
 4:     Counter[i] : array of integer defined for all i ∈ N_v
 5: Initialization :
 6:     id ← 1
 7:     For all i ∈ N_v do L[i] ←⊥ ; Counter[i] ← 1
 8: On request to route m
 9:     randomly select next ∈ N_v using probability law P^v_TLCN(Counter)
10:     send ⟨m,(v,id)⟩ to next
11:     update((v,id),L[next],s) ; id++
12: Upon receiving ⟨m,tag⟩ from u
13:     if ∃z ∈ N_v : tag ∈ L[z] then
14:         Counter[u]++ ; Counter[z]++
15:         remove (tag, L[z])
16:     randomly select next ∈ N_v using probability law P^v_TLCN(Counter)
17:     send ⟨m, tag⟩ to next ; update(tag,L[next],s)
```

3 Expérimentations

Nous avons comparé par simulation les performances de nos algorithmes avec celles de RW et YRW. Nous présentons ici les résultats obtenus en termes de temps d'atteinte et de volume de données échangées, c'est-à-dire la somme des tailles de messages générés pour router une donnée. Pour cela, nous avons utilisé le simulateur pour WSNs, Sinalgo[‡]. Dans nos simulations, les capteurs sont déployés de manière aléatoire uniforme sur une surface carrée et la topologie du réseau correspond à un graphe à disques unitaires de rayon r : deux nœuds sont voisins si leur distance est inférieure ou égale à r. Dans nos expérimentations, la transmission des messages est fiable ; les nœuds et les liens de communications sont asynchrones ; les réseaux sont connexes et contiennent un seul puits ; enfin, chaque noeud source génère 100 données à router, l'intervalle entre deux générations étant choisi aléatoirement entre 1 et 10 unités de temps. De plus, chaque nœud vérifie s'il a reçu des messages tous les 30 unités de temps.

Nous avons testé nos algorithmes avec les deux politiques de mise à jour, ainsi que plusieurs bornes sur la taille des listes. Nous remarquons sur les simulations que plus la taille de la liste est grande meilleurs sont les résultats. Cependant, à partir de 15, le gain que nous avons observé en termes de temps d'atteinte devient négligeable. Concernant le volume, les meilleurs résultats sont obtenus avec une borne égale à 1 dans TLM alors que pour TLCN, la borne n'influe pas sur le volume. Dans la suite, nous présentons les résultats obtenus avec les bornes 1 et 15 pour TLM et seulement 15 pour TLCN.

Temps d'atteinte : Nous avons tout d'abord fait varier la taille des réseaux, de 50 à 200 nœuds en fixant le degré moyen à 8. Les résultats sont présentés dans la figure 3 à gauche. Nous observons que tous nos protocoles sont significativement meilleurs que RW et YRW ; c'est en particulier vrai pour TLM avec une borne de 1. En fait, la taille 1 permet à tout message routé en utilisant TLM de ne pas repartir directement en arrière, ce que les marches aléatoires RW et YRW permettent. Nous observons aussi que pour TLM, la politique FIFO est bien meilleure que la politique RAND, tandis que la politique de mise à jour n'a pas d'influence sur les résultats obtenus avec TLCN. Finalement, notons que les meilleurs résultats sont obtenus avec TLCN (15,FIFO) et TLCN (15,RAND).

Nous avons ensuite étudié l'impact du degré moyen du réseau sur nos algorithmes. Ainsi, nous avons considéré des réseaux de 400 nœuds dont le degré moyen varie entre 16 et 46. Les résultats sont présentés dans la figure 3 à droite. Nos expériences montrent que les performances relatives de nos protocoles, de RW et de YRW restent les mêmes que précédemment. L'augmentation du degré moyen implique une réduction du diamètre du réseau, ce qui explique la baisse du temps d'atteinte observée. Enfin, nous pouvons remarquer que l'écart positif entre les protocoles TLCN et les autres est encore plus important que précédemment.

‡. http://disco.ethz.ch/projects/sinalgo/

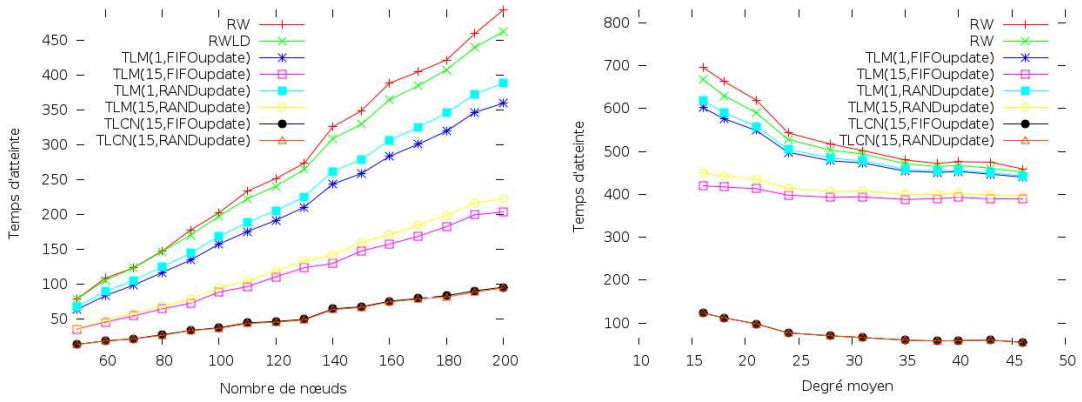


FIGURE 3: Temps d'atteinte

De toutes ces expérimentations, nous pouvons conclure que la meilleure politique pour `TLM` est `FIFO`. Cette politique empêche les messages de suivre tout cycle élémentaire de taille inférieure à la taille de la liste plus 1. Inversement, `RAND` permet d'éviter les cycles de toutes tailles avec une probabilité positive : tout nœud peut rester arbitrairement longtemps dans la liste, cependant il peut aussi rester moins longtemps dans la liste qu'avec une politique `FIFO`. Ainsi, `FIFO` traite plus efficacement les petits cycles et `RAND` les grands. Nous pouvons alors conclure qu'éviter les petits cycles a plus d'impact sur le temps d'atteinte.

Concernant `TLCN`, nous pouvons remarquer que la politique de mise à jour n'a pas d'impact sur les résultats. Nous avons observé que `TLCN` était influencé par deux paramètres : la taille des cycles élémentaires et la charge du réseau. En effet, s'il existe trop de messages dans le réseau à un instant donné, alors la probabilité qu'un message disparaisse d'une liste avant de revenir dans un nœud est importante, dans ce cas le cycle n'est pas détecté. Dans le pire des cas, les performances de `TLCN` deviennent presque identiques à `YRW`. Ainsi, pour obtenir de bons résultats, la taille de la liste doit être corrélée au ratio entre le temps moyen entre deux générations de messages et le temps de transmission moyen.

Volume : Dans `RW` et `YRW`, chaque message contient uniquement la donnée à router d. Nous notons $|d|$ la taille de d et s la taille maximale des listes. Pour simplifier, nous supposons que les identifiants et numéro de séquence sont des entiers et que la taille d'un entier vaut 1.

Dans `TLM`, chaque message contient une donnée d et au plus s identités, il est donc de taille au plus $|d|+s$. Dans `TLCN`, chaque message contient une donnée d, l'identité de la source et un numéro de séquence, il a donc pour taille $|d|+2$. Nos résultats expérimentaux pour le volume sont donnés dans la figure 4 pour des réseaux de 50 à 200 nœuds avec un degré moyen de 8. Nous avons fixé $|d|$ à 1, pour étudier nos algorithmes au pire cas. Remarquons que toutes les versions de `TLM` sont moins efficaces en volume que `RW` et `YRW` : le gain en nombre de sauts ne compense pas le surcoût en taille de message. Au contraire, même si la taille des messages de `TLCN` est plus importante que celle de `RW` et `YRW`, le volume reste meilleur en utilisant `TLCN` : le gain en nombre de sauts compense largement le surcoût en taille de message.

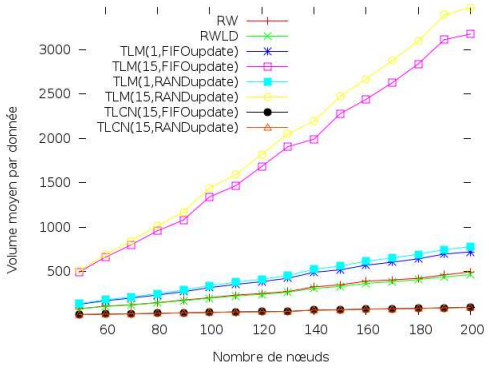


FIGURE 4: Volume d'informations échangées

Références

[SIY09] Izumi Kubo Satoshi Ikeda and Masafumi Yamashita. The hitting and cover times of random walks on finite graphs using local degree information. *Theoretical Computer Science*, 410 :94–100, 2009.