Un algorithme d'élection de leader cross-layer pour réseaux mobiles ad hoc (résumé)

Arnaud Favier, Nicolas Guittonneau, Luciana Arantes, Anne Fladenmuller, Pierre Sens

Compas'2019 : Parallélisme/ Architecture / Système
LIUPPA - IUT de Bayonne, France, du 24 au 28 juin 2019

Résumé : un algorithme d'élection de leader cross-layer pour réseaux mobiles ad hoc

Arnaud Favier[‡], Nicolas Guittonneau[‡], Luciana Arantes[‡], Anne Fladenmuller, Pierre Sens[‡] *

Sorbonne Université, CNRS, [‡]Inria
Laboratoire LIP6
4 place Jussieu, 75252 Paris - France
{arnaud.favier, nicolas.guittonneau, luciana.arantes, anne.fladenmuller, pierre.sens}@lip6.fr

---

Résumé

Un MANET (*mobile ad hoc network*) est un réseau dynamique décentralisé qui s'autoconfigure continuellement, composé de nœuds pouvant se déplacer librement, rejoindre et quitter le système à tout moment. Les nœuds ne connaissent initialement que leur identifiant. Seuls les nœuds situés dans le rayon de transmission l'un de l'autre communiquent directement. En revanche, toute paire de nœuds peut communiquer via un ensemble de nœuds relais intermédiaires.

L'élection de leader est un composant essentiel des systèmes répartis permettant par exemple, de résoudre le problème du consensus. L'élection consiste à trouver une entente sur l'identité d'un unique nœud, considéré comme leader par l'ensemble des nœuds du réseau. Lors d'un départ ou d'une défaillance du leader, les autres nœuds détectent son indisponibilité et déclenchent une nouvelle élection.

De nombreux travaux portent sur l'élection de leader dans les systèmes statiques [6, 8]. Dans un système dynamique, [1] proposent un algorithme adapté aux topologies mobiles et reposant sur des TVG [3]. D'autres algorithmes spécifiques aux MANET utilisent une vision partielle du réseau, le plus souvent pour élire le nœud ayant le plus faible identifiant [7, 5, 4, 2, 9].

Notre algorithme obtient une vision globale à partir de la liste des voisins que chaque nœud maintient et échange. Périodiquement, les nœuds envoient un checksum de leur vision pour détecter les incohérences. Les communications sont en broadcast uniquement. Différents critères permettent de choisir le leader comme son ancienneté dans le réseau ou sa centralité.

L'algorithme est conçu pour les réseaux mobiles génériques et possède une implémentation cross-layer adaptée aux MANET. Notre algorithme exploite la connexion sans fil et les informations maintenues par la couche MAC pour améliorer les performances des échanges.

L'algorithme a été implémenté sur une plateforme de dix Raspberry Pi mobiles et de premiers résultats sont prometteurs. Dans le futur, le choix du leader sera amélioré en prenant en compte la stabilité des nœuds. Des expérimentations dans le simulateur OMNeT++ permettront de tester le passage à l'échelle avec différents patterns de mobilité.

Mots-clés : Systèmes distribués, élection de leader, MANET, cross-layer

---

∗. Le texte a été relu par Luciana Arantes et Pierre Sens.

Bibliographie

1. Arantes (L.), Greve (F.), Sens (P.) et Simon (V.). – Eventual leader election in evolving mobile networks. – In *International Conference On Principles Of Distributed Systems*, pp. 23–37. Springer, 2013.
2. Bhoir (S.) et Vidhate (A.). – A modified leader election algorithm for manet. *International Journal on Computer Science and Engineering*, vol. 5, n2, 2013, p. 78.
3. Casteigts (A.), Flocchini (P.), Quattrociocchi (W.) et Santoro (N.). – Time-varying graphs and dynamic networks. – In *International Conference on Ad-Hoc Networks and Wireless*, pp. 346–359. Springer, 2011.
4. Ingram (R.), Radeva (T.), Shields (P.), Viqar (S.), Walter (J. E.) et Welch (J. L.). – A leader election algorithm for dynamic networks with causal clocks. *Distributed computing*, vol. 26, n2, 2013, pp. 75–97.
5. Ingram (R.), Shields (P.), Walter (J. E.) et Welch (J. L.). – An asynchronous leader election algorithm for dynamic networks. – In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pp. 1–12. IEEE, 2009.
6. Le Lann (G.). – Distributed systems-towards a formal approach. – In *IFIP congress*volume 7, pp. 155–160. Toronto, 1977.
7. Malpani (N.), Welch (J. L.) et Vaidya (N.). – Leader election algorithms for mobile ad hoc networks. – In *Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, pp. 96–103. ACM, 2000.
8. Peleg (D.). – Time-optimal leader election in general networks. *Journal of parallel and distributed computing*, vol. 8, n1, 1990, pp. 96–99.
9. Rahman (M.), Abdullah-Al-Wadud (M.) et Chae (O.). – Performance analysis of leader election algorithms in mobile ad hoc networks. *Int'l J. of Computer Science and Network Security*, vol. 8, n2, 2008, pp. 257–263.

1. Annexe

Algorithm 1: Cross-layer Leader Election

```
1  Initialisation of node i :
2      network[i].neighbors ← {i}
3      network[i].clock ← 0
4  Upon reachable peer j :
5      network[i].neighbors ← network[i].neighbors ∪ {j}
6      network[i].clock ← network[i].clock + 1
7      Broadcast (network)
8  Upon unreachable peer j :
9      edit ← {⟨i, ∅, {j}, network[i].clock, network[i].clock + 1⟩}
10     network[i].neighbors ← network[i].neighbors \ {j}
11     network[i].clock ← network[i].clock + 1
12     Broadcast (edit)
13 Upon reception of network_j from peer j :
14     edit ← {}
15     for each peer p in network_j do
16         if network[p] is empty then
17             edit ← edit ∪ {⟨p, p.neighbors, ∅, 0, p.clock⟩}
18             network[p].neighbors ← p.neighbors
19             network[p].clock ← p.clock
20         else if p.clock > network[p].clock then
21             added ← p.neighbors \ network[p].neighbors
22             removed ← network[p].neighbors \ p.neighbors
23             edit ← edit ∪ {⟨p, added, removed, network[p].clock, p.clock⟩}
24             network[p].neighbors ← p.neighbors
25             network[p].clock ← p.clock
26     if edit is not empty then
27         Broadcast (edit)
```

```
28 Upon reception of edit from peer j :
29     for each ⟨source, added, removed, old_clock, new_clock⟩ in edit do
30         if added is not empty then
31             if old_clock equals 0 OR old_clock equals network[source].clock then
32                 network[source].neighbors ← network[source].neighbors ∪ added
33         if removed is not empty then
34             if old_clock equals network[source].clock then
35                 network[source].neighbors ← network[source].neighbors\removed
36         if network[source].neighbors was updated then
37             network[source].clock ← new_clock
38     if network was updated then
39         Broadcast (edit)
40 Repeat forever :
41     Broadcast (Checksum(network))
42     Wait Δ seconds
43 Upon reception of checksum from peer j :
44     if checksum not equals Checksum(network) then
45         Broadcast (network)
46 Upon invocation of Leader() :
47     accessible ← {i}
48     for each peer p reachable by i do
49         accessible ← accessible ∪ {p}
50     return Min(accessible)
```