Algorithme de gestion de groupe pour réseaux ad hoc fortement dynamiques

Bertrand Ducourthial, Sofiane Khalfallah, Franck Petit

Algorithme de gestion de groupe pour réseaux ad hoc fortement dynamiques

Bertrand Ducourthial*, Sofiane Khalfallah* and Franck Petit**

*Laboratoire Heudiasyc UMR CNRS 6599
Université de Technologie de Compiègne, France
prénom.nom@hds.utc.fr

**Laboratoire MIS
Université Picardie Jules Verne, Amiens
prénom.nom@u-picardie.fr

Nous proposons un service de gestion de groupe adapté aux réseaux ad hoc fortement dynamiques tels que les réseaux de véhicules. Ce service maintient un groupe restreint dans un certain diamètre D_{max} dépendant de critères applicatifs. Notre solution fonctionne dans un environnement asynchrone et ne requiert pas que les communications soient fiables. L'algorithme que nous proposons est auto-stabilisant, c'est-à-dire qu'il construit des groupes satisfaisant les contraintes quelque soit la configuration initiale. De plus, lorsqu'un nœud s'ajoute ou se retire d'un groupe stable, l'effet de cette modification est corrigé sur tous les nœuds du groupe en un temps optimal, soit $O(D_{max})$ unités de temps.

Keywords: réseaux ad hoc, gestion de groupe, réseau fortement dynamique, tolérance aux fautes, auto-stabilisation

1 Introduction

Contexte. Un réseau ad hoc mobile ne possède pas d'infrastructure fixe. Il est fortement dynamique lorsque les nœuds apparaissent et disparaissent des voisinages avec une fréquence élevée, engendrant une topologie très instable. Les réseaux ad hoc fortement dynamiques conduisent à de nouveaux problèmes algorithmiques, car les solutions classiques ne sont généralement plus praticables. Un exemple de tel réseau est donné par les réseaux inter-véhicules. Ces réseaux font actuellement l'objet de nombreuses études, motivées par les problématiques de sécurité routière, de gestion des infrastructures de transport, d'aide à la conduite ou de services aux passagers. Parmi les applications inter-véhicules envisagées, certaines nécessitent un service de gestion de groupe (messagerie, perception coopérative, jeux inter-véhicules...). En effet, il est nécessaire de savoir distinguer les véhicules voisins qui participent de ceux qui n'y participent pas dans le but d'assurer la survie de telles applications malgré la dynamique du réseau. Par exemple, si tout message émis par un voisin est relayé dans la messagerie inter-véhicules, la discussion aurait peu d'intérêt car il serait impossible à certains véhicules de répondre à temps.

État de l'art. Dans [SBB02, Sch06], les auteurs proposent de résoudre le problème de gestion de groupe dans les réseaux dynamiques en utilisant un service générique de consensus muni d'un détecteur de défaillances. Cependant, ces travaux supposent que le nombre de participants soit connu d'avance. Une autre approche pour mettre en place un service de gestion de groupe dans ce type de réseaux consiste à mettre en place des algorithmes auto-stabilisants. De tels algorithmes ont la propriété de converger vers une vue stable du groupe malgré les pannes transitoires ou définitives qui pourraient survenir. Dans [DSW06], un service auto-stabilisant de gestion de groupe adapté aux réseaux mobiles est proposé. Cette solution se base sur une circulation aléatoire (*random walk*) d'un agent contenant des informations sur la constitution du groupe. Cette solution ne permet cependant pas de limiter le diamètre du groupe.

Contribution. Nous proposons un *service de gestion de groupe* restreint dans un certain diamètre D_{max} dépendant de critères applicatifs. Nous nous plaçons dans le cadre d'un réseau de véhicules à communication WiFi, bien que notre algorithme puisse s'adapter à d'autres contextes. Notre solution, dont le principe est décrit dans la section 2, se fonde sur le maintien sur chaque nœud d'une liste ordonnée des antécédents dont la longueur est limitée à D_{max}. Cette liste est construite de manière *auto-stabilisante* par un r-opérateur [DT03], c'est-à-dire que quelque soit la configuration initiale du système, la liste maintenue sur chaque nœud finit par être conforme à la définition du groupe en un temps fini. Cette construction est formellement

décrite dans la section 3. De plus, l'ajout ou le retrait d'un nœud d'un groupe stable ne conduit pas à la restabilisation de l'ensemble du système car l'effet de cette modification est corrigé sur tous les nœuds du groupe en temps optimal, soit $O(D_{max})$ unités de temps. L'algorithme que nous proposons est donc *adaptatif en temps* [KPS99].

Notre solution est constituée de deux protocoles en couche. Le premier est un algorithme de gestion de voisinage dynamique décrit dans la section 4. Le second protocole est décrit dans la section 5. Il s'appuie sur le protocole de gestion du voisinage et constitue l'algorithme de gestion de groupe proprement dit. Nous concluons cet article dans la section 6.

2 Principe du service de gestion de groupe

Notre algorithme utilise la *liste ordonnée des antécédents*, notée par la suite *listAnt*. Pour un nœud v, *listAnt* est la liste constituée de v, suivi de ses antécédents à distance 1, puis de ses antécédents à distance 2 et ainsi de suite. La constitution de la liste ordonnée des antécédents de chaque nœud permet de former des groupes. En effet, lorsqu'un nœud u décide de créer un groupe, il diffuse la liste (u). En recevant ce message, tout nœud v voisin de u voulant adhérer au groupe diffuse la liste (v,u). De la même manière, un voisin w de v finit par diffuser la liste (w,v,u), à moins qu'il ne soit lui-même également voisin de u, auquel cas il diffuse la liste (w,uv). La liste continue ainsi à s'agrandir jusqu'à ce qu'elle atteigne une taille maximale : un nœud ne pourra diffuser une liste de taille excessive, ce qui empêchera l'adhésion au groupe de nœuds qui en seraient trop éloigné.

Outre cette phase d'adhésion, l'algorithme inclut également une phase d'exclusion, de manière à conserver la contrainte sur le diamètre en cas d'étirement du groupe (lié à la mobilité des nœuds). Dans le cas précédent, avec un diamètre maximal de 1, si w se déplace de sorte qu'il ne soit plus voisin de u, il s'interdira d'émettre la liste (w,v,u) et déduira qu'il n'est plus dans le groupe de v.

Après avoir été exclu du groupe, le nœud w continue à émettre régulièrement la liste (w) qui est reçue par v tant que celui-ci est à sa portée. Cependant, afin de limiter le diamètre du groupe (dans notre exemple, égal à 1), v ne doit pas propager ce message vers les autres membres du groupe (c'est à dire u). Pour cela, le mécanisme précédent est complété par un système de marquage des nœuds, représenté dans la suite en soulignant un nœud marqué. Ainsi, lorsque v reçoit la liste émise par w, constatant qu'il n'appartient pas à la liste, il diffuse la liste $(v,u\underline{w})$. À la réception du message, u n'intègre pas le noeud w dans sa liste, ce dernier étant marqué. Le nœud w ne prend donc pas en compte la liste de v tant qu'il est éloigné de u. La composition du groupe se stabilise donc sur u et v.

Supposons maintenant que le nœud w se rapproche à nouveau de u et devienne son voisin. Le nœud w reçoit alors régulièrement les listes (u,v) et (v,u) émises respectivement par u et v. Conformément au mécanisme de marquage décrit ci-dessus, w finit par émettre la liste $(w,\underline{uv})$. À leur tour, les nœuds u et v finissent par émettre respectivement les listes (u,vw) et (v,uw). Par la suite, w émettra (w,uv). Le groupe sera alors stabilisé sur u, v et w.

3 Construction auto-stabilisante de la liste des antécédents

En modélisant les algorithmes répartis par des opérateurs algébriques, des propriétés de terminaison et de stabilisation globales peuvent être assurées en vérifiant simplement des propriétés locales de l'opérateur. Pour stabiliser un calcul réparti malgré les circuits dans le réseau, la propriété d'idempotence est requise ($x \cdot x = x$). Cependant les opérateurs des demi-groupes idempotents (*e.g.*, $\min(x,y)$) ne supportent pas les pannes transitoires. En utilisant un endomorphisme (*e.g.*, $x \mapsto x+1$), ces opérateurs sont généralisés en r-opérateurs (*e.g.*, $\min(x,y+1)$). Le demi-groupe idempotent abélien apparaît alors comme un cas particulier des r-demi-groupes, obtenu avec l'endomorphisme identité $x \mapsto x$ [Duc07]. Un r-opérateur est r-associatif ($x \triangleleft (y \triangleleft z) = (x \triangleleft y) \triangleleft r(z)$), r-commutatif ($r(x) \triangleleft y = r(y) \triangleleft x$), r-idempotent ($r(x) \triangleleft x = r(x)$) et admet un élément neutre à gauche ($x \triangleleft e_{\triangleleft} = x$).

Sous certaines conditions, un r-demi-groupe induit un demi-groupe et cela donne une méthode pour construire un r-opérateur [Duc07] : trouver un demi-groupe idempotent abélien $(\mathbb{S},\oplus)$ puis un endomorphisme $r : \mathbb{S} \to \mathbb{S}$. Ces deux structures admettent une relation d'ordre. Un r-opérateur idempotent vérifie

$\forall x \in \mathbb{S}, x \preceq_{\oplus} x$ où $\preceq_{\oplus}$ est la relation d'ordre du demi-groupe induit. Lorsqu'on a $\forall x \in \mathbb{S}, x \prec_{\oplus} x$, le *r*-opérateur est dit strictement idempotent. Il a été montré dans [DDT06] que des tâches statiques auto-stabilisantes peuvent être accomplies dans le modèle à passage de messages non fiables dès lors que l'ordre induit par un *r*-opérateur strictement idempotent est un ordre total. Dans ce même article, une extension pour réseaux WiFi est discutée. L'algorithme de voisinage que nous proposons ci-dessous correspond à cette discussion. Cette dernière reste donc valide dans le cadre du présent travail, mais avec un *r*-opérateur induisant un ordre qui n'est que partiel.

Pour construire la liste des antécédents, nous considérons l'ensemble $\mathbb{S}$ des listes d'ensemble de sommets. Les listes de l'exemple présenté au § 2, telle que (u,v,w) ou (u,vw), sont bien des éléments de $\mathbb{S}$. On définit sur $\mathbb{S}$ l'opérateur $\oplus$, qui fusionne les listes en supprimant les informations superflues ou répétitives (un nœud n'apparaît qu'une seule fois dans une liste d'antécédents). Par exemple $(d,b,ac) \oplus (c,ae,b) = (dc,bae,acb) = (dc,bae)$. Enfin, on définit l'endomorphisme r de $\mathbb{S}$ qui consiste à insérer la liste vide en début de liste. Par exemple $r(d,b,a,c) = (\emptyset,d,b,a,c)$. On définit alors l'opérateur $\triangleleft$ par : $l_1 \triangleleft l_2 = l_1 \oplus r(l_2)$, où l_1 et l_2 sont des listes de $\mathbb{S}$. Il s'agit d'un *r*-opérateur strictement idempotent, induisant une relation d'ordre partielle, appelé `ant` [DT03].

4 Algorithme de gestion de voisinage dynamique

Le but de l'algorithme de Gestion de Voisinage Dynamique (GVD) est de maintenir l'ensemble des voisins (*neighbors*) et un tableau local des derniers messages reçus pour chacun des voisins du nœud v ($lastMsg[]$). Ces deux variables sont écrites par GVD et lues par l'algorithme de gestion de groupe dynamique (GGD), décrit dans la prochaine section. La communication entre les deux algorithmes est donc réalisée par variables partagées en lecture ; les actions gardées des algorithmes sont atomiques. Nous supposons que l'algorithme GGD émet régulièrement des messages dans le voisinage.

Un nœud u est considéré comme voisin de v si ce dernier reçoit au moins un message du nœud u dans un délai inférieur à un certain seuil, noté *neighDelay*.

Algorithme de Gestion de Voisinage Dynamique (GVD) du nœud v

1 $\mathcal{R}_1$ **Réception d'un message** `msg` **émis par** `sender`
2 $lastMsg[sender] \leftarrow msg$
3 $neighbors \leftarrow neighbors \cup \{sender\}$
4 $date[sender] \leftarrow localTime()$
5 $\mathcal{R}_2$ **Expiration du chien de garde de voisinage**
6 **pour tout** $u \in neighbors$ **faire**
7 **si** $localTime()$ - $date[u] > neighDelay$ **alors**
8 supprimer les entrées u dans $lastMsg, neighbors$ et $date$
9 **fin si**
10 **fin pour**
11 réarmer le chien de garde de voisinage avec le délai *neighDelay*

5 Algorithme de gestion de groupe dynamique

L'algorithme de gestion de groupe est composé de deux *actions gardées*, $\mathcal{R}_3$ et $\mathcal{R}_4$. La première règle est active à chaque modification de la variable partagée *lastMsg*, afin de recalculer la liste des antécédents (*listAnt*) via le r-opérateur `ant`. Pour gérer le mécanisme de marquage des nœuds, lorsque le nœud local v n'est pas dans *lastMsg* de l'un de ses antécédents u, alors il ne prend en compte dans le calcul de *listAnt* que la liste $(\underline{u})$. Dans le cas contraire, le nœud v prend en compte le contenu de $lastMsg[u]$ moins les nœuds marqués. Si le résultat du calcul local dépasse la taille autorisée par l'application (D_{max}), alors cette liste est recalculée sans la ou les entrées de $lastMsg[]$ responsables du dépassement. La règle $\mathcal{R}_3$ fournit à la fois la liste des antécédents pour la règle $\mathcal{R}_4$ et la composition du groupe pour l'application utilisatrice.

Nous montrons que la composition de l'algorithme *GVD* avec l'algorithme *GGD* respecte les propriétés de *vivacité* et de *sûreté*. La première garantit que si un nœud v envoie infiniment souvent un message

Algorithme de Gestion de Groupe Dynamique (GGD) du nœud v

```
1   R3 Modification de la variable partagée lastMsg[]
2   listAnt ← (v)
3   pour tout u ∈ neighbors faire
4     si v ∉ lastMsg[u] alors
5           list ← (u)
6     sinon
7           list ← lastMsg[u] moins les nœuds marqués
8     fin si
9     listAnt ← ant(listAnt,list)
10  fin pour
11  si |listAnt| > Dmax + 1 alors
12    refaire le calcul précédent sans les listes trop longues
13  fin si
14  group ← ensemble des identités présentes dans listAnt
15  R4 Expiration du chien de garde du groupe
16  diffuser (listAnt)
17  réarmer le chien de garde du groupe avec le délai groupDelay
```

contenant son identité à un voisin u membre d'un groupe et dont la liste $listAnt(u)$ n'a pas atteint D_{max}, alors le nœud v finira par être admis comme membre du groupe considéré par u ($v \in group(u)$). La propriété de sûreté garantit que tout nœud u dans le groupe de v qui envoie infiniment souvent des messages se trouve à une distance inférieure à D_{max}. On montre aussi que l'algorithme construit des groupes disjoints. Par ailleurs, outre la stabilisation malgré les défaillances transitoires, on montre que l'algorithme s'adapte rapidement à l'arrivée ou au départ d'un nœud : ces modifications sont prises en compte en au plus $O(D_{max})$ étapes, et n'entraînent pas une restabilisation complète de l'ensemble du système.

6 Conclusion

En utilisant des listes d'antécédents tronquées, nous obtenons un service de gestion de groupe. En basant cet algorithme sur le *r*-opérateur `ant`, notre algorithme est auto-stabilisant et supporte les pannes transitoires dans le réseau. Il est capable de s'adapter à la dynamique du réseau puisqu'une anomalie est vite corrigée. Enfin, puisqu'il n'utilise pas de consensus, il est adapté aux réseaux asynchrones non fiables. Ce service est à la base des applications pour réseaux de véhicules que nous étudions.

Références

[DDT06] S. Delaët, B. Ducourthial, and S. Tixeuil. Self-stabilization with r-operators revisited. In *Journal of Aerospace Computing, Information, and Communication*, 2006.

[DSW06] S. Dolev, E Schiller, and J.L. Welch. Random walk for self-stabilizing group communication in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(7) :893–905, 2006.

[DT03] B. Ducourthial and S. Tixeuil. Self-stabilization with path algebra. *Theor. Comput. Sci.*, 293(1) :219–236, 2003.

[Duc07] B. Ducourthial. r-semi-groups : A generic approach for designing stabilizing silent tasks. In *9th Stabilization, Safety, and Security of Distributed Systems (SSS'2007)*, pages 281–295, Paris, novembre 2007.

[KPS99] S. Kutten and B. Patt-Shamir. Stabilizing time adaptive protocols. *Theoretical Computer Science*, 220(1) :93–111, 1999.

[SBB02] H. Seba, .N Badache, and A. Bouabdallah. Solving the consensus problem in a dynamic group : An approach suitable for a mobile environment. *ISCC*, 00 :327, 2002.

[Sch06] André Schiper. Dynamic group communication. *Distributed Computing*, 18(5) :359–374, 2006.