Clustering auto-stabilisant à k sauts dans les réseaux Ad Hoc

Mandicou Ba, Olivier Flauzac, Haggar Bachar Salim, Florent Nolot, Ibrahima Niang

MajecSTIC 2012
Lille, France, du 29 au 31 octobre 2012

Clustering auto-stabilisant à k sauts dans les réseaux Ad Hoc

Mandicou BA[1] Olivier FLAUZAC[1], Bachar Salim HAGGAR[1], Florent NOLOT [1] et Ibrahima NIANG[2]

1 :Université de Reims Champagne-Ardenne, CReSTIC - SysCom EA 3804, Reims France
{ mandicou.ba, olivier.flauzac, bachar-salim.haggar, florent.nolot}@univ-reims.fr
2 : Université Cheikh Anta Diop, Laboratoire d'Informatique de Dakar (LID), Dakar Sénégal
iniang@ucad.sn

Résumé
Les réseaux *ad hoc* offrent de nombreux domaines d'application du fait de leur facilité de déploiement. La communication qui s'effectue classiquement par diffusion est couteuse et peut entrainer une saturation du réseau. Pour optimiser ces communications, une approche est de structurer le réseau en *clusters*. Dans cet article, nous présentons un algorithme de *clustering* asynchrone, distribué et auto-stabilisant qui construit des *clusters* à *k sauts*. Notre approche ne nécessite aucune initialisation. Elle se base uniquement sur l'information provenant des nœuds voisins à l'aide d'échange périodique de messages. Partant d'une configuration quelconque, le réseau converge à un état stable au bout d'un nombre fini d'étapes. Par un schéma de preuve, nous montrons que pour un réseau de n nœuds, la stabilisation est atteinte au plus en $n + 2$ transitions et nécessite une occupation mémoire d'au plus de $n * \log(2n + k + 3)$. Avec des simulations sous *Omnet++*, nous évaluons les performances moyennes de notre algorithme.

Abstract
Ad hoc networks offer many applications areas because of their easy of deployment. Communication that takes place by diffusion is typically expensive and may cause network saturation. To optimize these communications, one approach is to structure the network into clusters. In this paper, we present a self-stabilizing asynchronous distributed algorithm that builds k-hops clusters. Our approach does not require any initialization. It is based only on information from neighboring nodes with periodic exchange of messages. Starting from an arbitrary configuration, the network converges to a stable state after a finite number of steps. We prove that the stabilization is reached at most $n + 2$ transitions and uses at most $n * \log(2n + k + 3)$ space, where n is the number of network nodes. Using the *Omnet++* simulator, we performed an evaluation of our approach.

Mots-clés : réseaux *ad hoc*, *clustering*, algorithmes distribués, auto-stabilisation.

Keywords: ad hoc networks, clustering, distributed algorithms, self-stabilizing.

1 Introduction

Dans les réseaux *ad hoc*, la solution de communication la plus utilisée est la diffusion. C'est une technique simple qui nécessite peu de calcul. Mais cette méthode est coûteuse et peut entrainer une saturation du réseau. Pour optimiser cette communication qui est une importante source de consommation de ressources, une solution est de structurer le réseau en *arbres* [1] ou en *clusters* [3].
Le *clustering* consiste à découper le réseau en groupes de nœuds appelés *clusters* donnant ainsi au réseau une structure hiérarchique [6]. Chaque *cluster* est représenté par un nœud particulier appelé *clusterhead*. Un nœud est élu *clusterhead* selon une métrique telle que le degré, la mobilité, l'identité des nœuds, la densité, etc. ou une combinaison de ces paramètres. Plusieurs solutions de *clustering* ont été proposées. Elles sont classées en algorithmes à *1 saut* [5,7–9] et à *k sauts* [2,10]. Cependant, ces approches génèrent beaucoup de trafic et nécessitent d'importantes ressources.
Dans cet article, partant des travaux de Flauzac et *al.* [5], nous proposons un algorithme de *clustering* à *k sauts* qui est complètement distribué et auto-stabilisant. L'auto-stabilisation [4] a été introduite par Dijkstra en 1974 comme étant un système qui, quelque soit sa configuration de départ, est garanti d'arriver à une configuration légale en un nombre fini d'étapes. Notre approche

construit des *clusters* non-recouvrants à *k sauts* et ne nécessite aucune initialisation. Elle se base sur le critère de l'identité maximale des nœuds et s'appuie seulement sur l'échange périodique de messages avec le voisinage à *1 saut*. Le choix de l'identité comme métrique apporte plus de stabilité par rapport aux critères dynamiques comme la densité, la mobilité ou le poids des nœuds.
La suite de l'article est organisée comme suit. Dans la section 2, nous étudions les solutions de *clustering* existantes. La section 3 présente notre contribution. A la section 4, nous décrivons le modèle sur lequel se base notre approche. Puis à la section 5, nous donnons le principe d'exécution et les détails de notre algorithme. Dans la section 6, nous donnons le schéma de la preuve de convergence de notre approche et de l'occupation mémoire nécessaire. Par simulations, nous évaluons les performances de notre approche au niveau de la section 7. Une conclusion et des perspectives sont données dans la section 8.

2 État de l'art

Plusieurs propositions de *clustering* auto-stabilisantes et déterministes ont été faites dans la littérature [2,5,7–10].
Les approches auto-stabilisantes [5,7–9] construisent des *clusters* à *1 saut*.
Mitton et *al.* [9] utilisent comme métrique la *densité* afin de minimiser la reconstruction de la structure en cas de faible changement de topologie. Chaque nœud calcule sa densité et la diffuse à ses voisins situés à distance *k*. Flauzac et *al.* [5], ont proposé un algorithme auto-stabilisant qui combine la découverte de topologie et de *clustering* en une seule phase. Elle ne nécessite qu'un seul type message échangé entre nœuds voisins. Un nœud devient *clusterhead* s'il possède la plus grande identité parmi tous ses voisins. Johnen et *al.* [7] ont proposé un algorithme auto-stabilisant basé sur un modèle à état qui construit des *clusters* de taille fixe. Ils attribuent un poids à chaque nœud et fixe un paramètre $\mathtt{SizeBound}$ qui représente le nombre maximal de nœuds dans un *cluster*. Un nœud ayant le poids le plus élevé devient *clusterhead* et collecte dans son *cluster* jusqu'à $\mathtt{SizeBound}$ nœuds. Dans [8], Johnen et *al.* ont étendu leur proposition décrite dans [7] pour apporter la notion de *robustesse* qui est la propriété qui assure que partant d'une configuration quelconque, le réseau est partitionné après un round asynchrone. Et durant la phase de convergence, il reste toujours partitionner et vérifie un prédicat de sureté.
Les approches auto-stabilisantes [2,10] construisent des *clusters* à *k sauts*.
Dans [10], Mitton et *al.* étendent leurs travaux décrits dans [9] pour proposer un algorithme robuste de *clustering* auto-stabilisant à *k sauts* basé sur un modèle à passage de messages. La robustesse leur permet de réduire le temps de stabilisation et d'améliorer la stabilité de *clusters*. Cependant, ils utilisent toujours comme métrique la densité que les nœuds calculent. Or en cas de forte mobilité, la densité change constamment. Datta et *al.* [2] ont proposé un algorithme auto-stabilisant basé sur un modèle à états nommé *k-Clustering*. Cet algorithme est basé sur une comparaison des identifiants des nœuds connectés par des arêtes pondérées. il s'exécute en $O(n * k)$ rounds et nécessite $O(\log(n) + \log(k))$ espace mémoire par processus, où n est le nombre de nœuds du réseau.

3 Contribution

Nous proposons une approche basée sur un modèle à passage de messages contrairement aux solutions proposées dans [2,7,8]. Notre solution est complètement distribuée et auto-stabilisante. Nous utilisons le critère d'identité maximale qui apporte plus de stabilité par rapport aux métriques variables utilisées dans [7–10]. Notre algorithme structure le réseau en *clusters* disjoints deux à deux et de diamètre au plus égal à 2k. Cette structuration ne nécessite pas de phase d'initialisation. Elle combine la découverte de voisinage et le *clustering* en une seule phase. Elle se base uniquement sur l'information provenant des nœuds voisins situés à distance 1 par le biais d'échange périodique de messages. Notre approche se stabilise au plus en $n + 2$ transitions et nécessite une occupation mémoire d'au plus de $n * \log(2n + k + 3)$. Avec des simulations sous *Omnet++*, nous évaluons les performances moyennes de notre approche.

4 Modèle

Le réseau peut être modélisé par un graphe connexe et non orienté $G = (V, E)$, où V est l'ensemble des nœuds du réseau et E représente l'ensemble des connexions existantes entre les nœuds. Une arête (u, v) existe si et seulement si u peut communiquer avec v et vice-versa. Ce qui implique que tous les liens sont bidirectionnels. Dans ce cas, les nœuds u et v sont voisins. L'ensemble des

nœuds $v \in V$ voisins du nœud u situés à distance 1 est noté N_u. Chaque nœud u du réseau possède un identifiant unique id_u et peut communiquer avec tout nœud $v \in N_u$. On définit la distance $d_{(u,v)}$ entre deux nœuds u et v quelconque dans le graphe G comme le nombre d'arêtes minimal le long du chemin entre u et v.
Dans notre approche, nous utilisons un modèle *asynchrone à passage de messages*. Pour cela, chaque nœud u envoie périodiquement à ses voisins $v \in N_u$ un message contenant les informations sur son état actuel. Un message envoyé est reçu au bout d'un temps fini mais non borné. Nous supposons qu'un message envoyé est correctement reçu. Chaque nœud u du réseau stocke aussi dans une table de voisinage l'état actuel de ses voisins. Dès réception d'un message d'un voisin, chaque nœud u met à jour sa table de voisinage et exécute l'algorithme de *clustering*.

5 Algorithme auto-stabilisant à k sauts

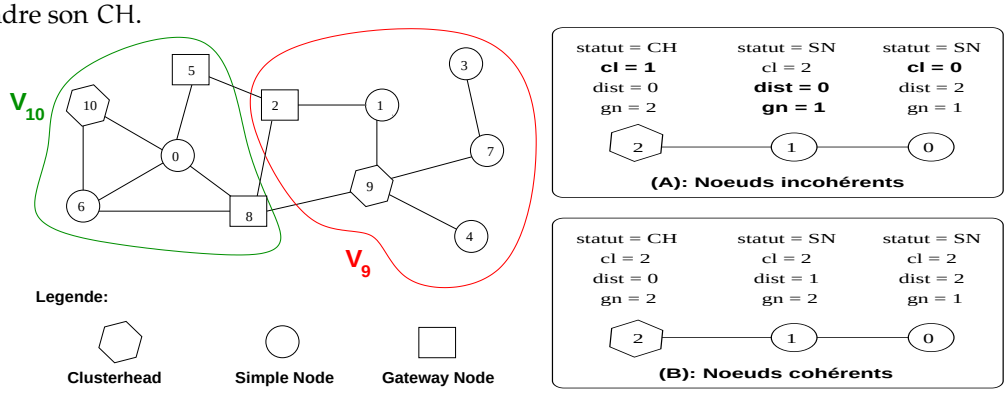
5.1 Préliminaires

Dans cette section, nous donnons quelques définitions utilisées dans la suite de ce papier.

Définition 5.1 *(Cluster) : Nous définissons un cluster à k sauts comme un sous graphe connexe du réseau, dont le diamètre est inférieur ou égal à* 2k*. L'ensemble des nœuds d'un cluster* i *est noté* V_i *(voir Figure 1(a)).*

Définition 5.2 *(Identifiant du cluster) : Chaque cluster possède un unique identifiant correspondant à la plus grande identité de tous les nœuds du cluster. L'identité d'un cluster auquel appartient le nœud* i *est notée* cl_i.

Dans nos *clusters*, chaque nœud u possède un statut noté $statut_u$. Ainsi, un nœud peut être soit *clusterhead* (CH), soit *Simple Node* (SN), ou soit *Gateway Node* (GN) comme illustré au niveau de la Figure 1(a). De plus, chaque nœud choisit un voisin $v \in N_u$, noté gn_u, par lequel il passe pour atteindre son CH.



(a) Statut des nœuds (b) Cohérences des nœuds

FIGURE 1 – Clustering à 2 sauts

Définition 5.3 *(Statut des nœuds) :*
- *Clusterhead (**CH**) : un nœud* u *a le statut de* CH *s'il possède le plus grand identifiant parmi tous les nœuds de son cluster (exemple du nœud 9 dans* V_9 *et du nœud 10 dans* V_{10} *illustré dans la Figure 1(a)) :*
  - $statut_u = CH \iff \forall v \in V_{cl_u}, (id_u \geq id_v) \wedge (dist_{(u,v)} \leq k)$.
- *Simple Node (**SN**) : un nœud* u *a le statut de* SN *si tous ses voisins appartiennent au même cluster que lui (exemple des nœuds 6 et 0 dans* V_{10} *illustré dans la Figure 1(a)) :*
  - $statut_u = SN \iff (\forall v \in N_u, cl_v = cl_u) \wedge (\exists w \in V / (statut_w = CH) \wedge (dist_{(u,w)} \leq k))$
- *Gateway Node (**GN**) : un nœud* u *a le statut de* GN *s'il existe un nœud* v *dans son voisinage appartenant à un autre cluster (exemple du nœud 2 dans* V_9 *et des nœuds 5 et 8 dans* V_{10} *illustré dans la Figure 1(a)) : :*
  - $statut_u = GN \iff \exists v \in N_u, (cl_v \neq cl_u)$

Définition 5.4 *(Nœud cohérent) : Un nœud* u *est cohérent si et seulement si, il est dans l'un des états suivants (voir Figure 1(b)) :*
- *Si* $statut_u = CH$ *alors* $(cl_u = id_u) \wedge (dist_{(u,CH_u)} = 0) \wedge (gn_u = id_u)$,
- *Si* $statut_u \in \{SN, GN\}$ *alors* $(cl_u \neq id_u) \wedge (dist_{(u,CH_u)} \neq 0) \wedge (gn_u \neq id_u)$.

Définition 5.5 *(Nœud stable) : Un nœud* u *est dans un état stable si et seulement si, il est cohérent et satisfait l'une des conditions suivantes :*

– *Si* $status_u = CH$ *alors* $\forall v \in N_u, (status_v \neq CH) \wedge \{((cl_v = cl_u) \wedge (id_v < id_u)) \vee ((cl_v \neq cl_u) \wedge (dist_{(v,CH_v)} = k))\}$
– *Si* $status_u = SN$ *alors* $\forall v \in N_u, (cl_v = cl_u) \wedge (dist_{(u,CH_u)} \leq k) \wedge (dist_{(v,CH_v)} \leq k)$
– *Si* $status_u = GN$ *alors* $\exists v \in N_u, (cl_v \neq cl_u) \wedge \{((dist_{(u,CH_u)} = k) \wedge (dist_{(v,CH_v)} \leq k)) \vee ((dist_{(v,CH_v)} = k) \wedge (dist_{(u,CH_u)} \leq k))\}$.

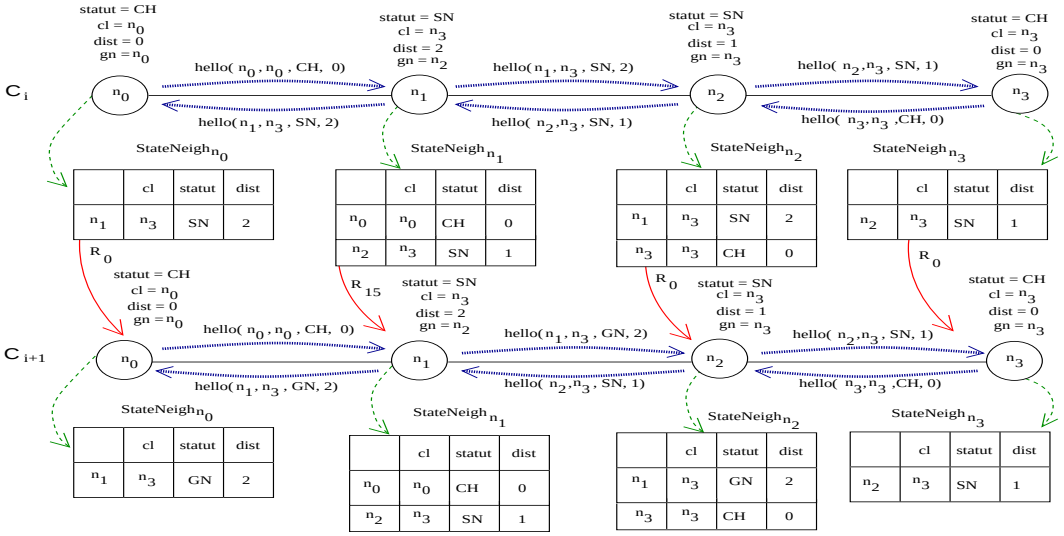


FIGURE 2 – Exemple de *clustering* à 2 sauts.

5.2 Principe d'exécution

Notre algorithme est auto-stabilisant, il ne nécessite aucune initialisation. Partant d'un état quelconque, avec seulement l'échange périodique de messages hello, les nœuds s'auto-organisent en *clusters* non-recouvrants au bout d'un nombre fini d'étapes. Ces messages *hello* contiennent l'identité du nœud (id_u), l'identité de son *clusterhead* (cl_u), son statut ($statut_u$), et la distance le séparant de son *clusterhead* ($dist_{(u,CH_u)}$). Ainsi, la structure des messages est la suivante : $hello(id_u, cl_u, statut_u, dist_{(u,CH_u)})$. De plus, chaque nœud maintient une table de voisinage ($StateNeigh_u$) contenant l'ensemble des états de ses nœuds voisins. $StateNeigh_u[v]$ contient les états du nœud v voisin de u.

La solution que nous proposons se déroule de la façon suivante :

A la réception de messages hello d'un voisin, chaque nœud met à jour sa table de voisinage puis exécute l'Algorithme 1 et informe tous ses voisins de son éventuel nouvel état. Au cours de l'exécution de l'Algorithme 1, une vérification de la cohérence des informations locales est effectuée à l'exécution de l'étape *Cluster-1* puis le traitement de nouvelles informations est effectué durant l'étape *Cluster-2*.

Un nœud u s'élit *clusterhead* s'il a la plus grande identité parmi tous les nœuds de son *cluster*. Si un nœud u a un voisin v de statut CH, avec une plus grande identité, alors il devient membre (SN) du *cluster* de v. Si un nœud u a un voisin appartenant à un *cluster* d'identité différente à celle de u et v. De plus, v est à distance inférieure à k de son CH. u en déduit qu'il existe un CH situé à une distance inférieure ou égale à k et dont l'identité est supérieure à celle de v. Si de plus l'identité du *cluster* de v est supérieure à celle de u, alors u devient SN et appartient au même *cluster* que v. Un nœud u devient nœud de passage, statut GN, s'il est voisin d'un nœud appartenant à un autre *cluster* ($cl_v \neq cl_u$).

La figure 2 illustre le passage d'une configuration C_i à C_{i+1}. Rappelons qu'une configuration C_i est une instance de l'état de tous les nœuds du réseau. A C_i, chaque nœud envoie à ses voisins un message *hello*. C_{i+1} est une configuration stabilisée. A la réception des messages venant des voisins, chaque nœud met à jour sa table de voisinage puis exécute l'Algorithme 1. Dans cet exemple, le nœud n_1 qui est un nœud simple appartenant au *cluster* de n_3 détecte le nœud n_0 comme un *cluster* voisin. Il devient nœud de passage avec un statut de GN et envoie un message hello à ses voisins pour une mise à jour. Les nœuds n_3, n_2 et n_0, en fonction de leurs états actuels ainsi que ceux de leurs voisins, ne changent pas d'états. C_{i+i} correspond à un état stable.

5.3 Algorithme auto-stabilisant de clustering à k sauts

Chaque nœud du réseau connait le paramètre k avec $k < n$, possède les *macros* suivants et exécute l'Algorithme 1.

- $NeighCH_u = \{id_v / v \in N_u \wedge statut_v = CH \wedge cl_u = cl_v\}$.

-$NeighMax_u = (Max\{id_v / v \in N_u \wedge statut_v \neq CH \wedge cl_u = cl_v\}) \wedge (dist_{(v,CH_u)} = Min\{dist_{(x,CH_u)}, x \in N_u \wedge cl_x = cl_v\})$.

Algorithme 1 : Construction de *clusters* à *k sauts*

1 */* Des la réception d'un message hello d'un voisin */*
2 **Prédicats**
3 $P_1(u) \equiv (status_u = CH)$
4 $P_2(u) \equiv (status_u = SN)$
5 $P_3(u) \equiv (status_u = GN)$
6 $P_{10}(u) \equiv (cl_u \neq id_u) \vee (dist_{(u,CH_u)} \neq 0) \vee (gn_u \neq id_u)$
7 $P_{20}(u) \equiv (cl_u = id_u) \vee (dist_{(u,CH_u)} = 0) \vee (gn_u = id_u)$
8 $P_{40}(u) \equiv \forall v \in N_u, (id_u > id_v) \wedge (id_u \geq cl_v) \wedge (dist_{(u,v)} \leq k)$
9 $P_{41}(u) \equiv \exists v \in N_u, (status_v = CH) \wedge (cl_v > cl_u)$
10 $P_{42}(u) \equiv \exists v \in N_u, (cl_v > cl_u) \wedge (dist_{(v,CH_v)} < k)$
11 $P_{43}(u) \equiv \forall v \in N_u / (cl_v > cl_u), (dist_{(v,CH_v)} = k)$
12 $P_{44}(u) \equiv \exists v \in N_u, (cl_v \neq cl_u) \wedge \{(dist_{(u,CH_u)} = k) \vee (dist_{(v,CH_v)} = k)\}$
13
14 **Règles**
15 */* Mise à jour du voisinage */*
16 $StateNeigh_u[v] := (id_v, cl_v, statut_v, dist_{(v,CH_v)})$;
17 */* Cluster-1 : Gestion de la cohérence */*
18 $R_{10}(u) : P_1(u) \wedge P_{10}(u) \longrightarrow cl_u := id_u; gn_u = id_u; dist_{(u,CH_u)} = 0$;
19 $R_{20}(u) : \{P_2(u) \vee P_3(u)\} \wedge P_{20}(u) \longrightarrow statut_u := CH; cl_u := id_u; gn_u = id_u; dist_{(u,CH_u)} = 0$;
20 */* Cluster-2 : Clustering */*
21 $R_{11}(u) : : \neg P_1(u) \wedge P_{40}(u) \longrightarrow status_u := CH; cl_u := id_v; dist_{(u,CH_u)} := 0; gn_u := id_u$;
22 $R_{12}(u) : : \neg P_1(u) \wedge P_{41}(u) \longrightarrow status_u := SN; cl_u := id_v; dist_{(u,v)} := 1; gn_u := NeighCH_u$;
23 $R_{13}(u) : : \neg P_1(u) \wedge P_{42}(u) \longrightarrow$
$status_u := SN; cl_u := cl_v; dist_{(u,CH_u)} := dist_{(v,CH_v)} + 1; gn_u := NeighMax_u$;
24 $R_{14}(u) : : \neg P_1(u) \wedge P_{43}(u) \longrightarrow status_u := CH; cl_u := id_v; dist_{(u,CH_u)} := 0; gn_u := id_u$;
25 $R_{15}(u) : : P_2(u) \wedge P_{44}(u) \longrightarrow status_u := GN$;
26 $R_{16}(u) : : P_1(u) \wedge P_{41}(u) \longrightarrow status_u := SN; cl_v := id_v; dist_{(u,v)} := 1; gn_u := NeighCH_u$;
27 $R_{17}(u) : : P_1(u) \wedge P_{42}(u) \longrightarrow$
$status_u := SN; cl_u := cl_v; dist_{(u,CH_u)} := dist_{(v,CH_v)} + 1; gn_u := NeighMax_u$;
28
29 */* Envoi d'un message hello */*
30 $R_0(u) : hello(id_u, cl_u, statut_u, dist_{(u,CH_u)})$;

6 Schéma intuitif de la preuve de la stabilisation

Dans cette section, nous énonçons les principaux théorèmes et propriétés vérifiés par notre algorithme et une analyse de complexité. Faute de place, nous ne donnons pas les détails des preuves. L'état d'un nœud est définit par la valeur de ses variables locales. Une *configuration* C_i du réseau est une instance de l'état de tous les nœuds. Avec notre approche, les nœuds les plus grands se fixent en premier. Un nœud u est dit *fixé* à partir de la configuration C_i si le contenu de sa variable cl_u ne change plus. L'ensemble des nœuds fixés à C_i est noté $\mathcal{F}_i$. Une *transition* τ_i est le passage d'une configuration C_i à C_{i+1}. Au cours d'une transition, chaque nœud a reçu un message de tous ses voisins et a exécuté l'Algorithme 1. Ainsi, avec notre approche le nombre de nœuds fixés croit strictement à chaque configuration et tend vers n, n étant le nombre de nœuds du réseau.

Lemme 6.1 *Soit C_0 une configuration quelconque. A C_1, $\forall u \in V$, u est cohérent.*

Preuve : à C_0 quelque soit son état, chaque nœud vérifie et corrige sa cohérence par exécution de la règle R_{10} ou R_{20} durant la transitions τ_0. Ainsi, à C_1 tout nœud est cohérent.

Corollaire 6.1 $|\mathcal{F}_1| \geq 1$.

Idée de la preuve : comme tous les nœuds sont cohérents d'après le lemme 6.1. Et $\exists$ u tel que $\forall$ v $\in V$, $id_u > id_v$. Au moins u applique R_{11} durant τ_0 et est donc fixé à C_1. D'où $u \in F_1$ et $|\mathcal{F}_1| \geq 1$ $\Rightarrow |\mathcal{F}_1| > |\mathcal{F}_0|$. Ce nœud u est noté CH_{Max}.

Théorème 6.1 $\forall\, i < k+1, |\mathcal{F}_{i+1}| > |\mathcal{F}_i|$ *et* $\mathcal{F}_i \subset \mathcal{F}_{i+1}$.
Idée de la preuve : d'après le corollaire 6.1, $|\mathcal{F}_1| > |\mathcal{F}_0|$. Pour $i = 0$, le résultat est vrai. A C_2, nous pouvons constater que les nœuds situés à distance 1 du CH_{Max} sont fixés soit par la règle R_{12} soit par la règle R_{16} selon que leur statut est SN ou CH. Donc, $|\mathcal{F}_2| > |\mathcal{F}_1|$. Nous prouvons ensuite par récurrence qu'à C_i, les nœuds situés à distance $(i-1)$ de CH_{Max} se fixent par la règle R_{13} ou par la règle R_{17}. Pour $i = k$, nous obtenons par récurrence $|\mathcal{F}_{k+1}| > |\mathcal{F}_k|$.

Théorème 6.2 ***(Convergence)***
Partant d'une configuration quelconque, une configuration stable est atteinte au plus en $n+2$ *transitions.*

Idée de la preuve : comme $|\mathcal{F}_1| \geq 1$, nous avons $|\mathcal{F}_{k+1}| > k$. En réitérant le processus à partir d'un nouveau CH_{Max} qui est le nœud d'identité maximale $\notin \mathcal{F}_{k+1}$, nous prouvons que $\forall\, i < n$, $|\mathcal{F}_{i+1}| > |\mathcal{F}_i|$ et $\mathcal{F}_i \subset \mathcal{F}_{i+1}$. Pour $i = n$, nous avons $|\mathcal{F}_{n+1}| > |\mathcal{F}_n|$ d'où $|\mathcal{F}_{n+1}| = n$. Il faut ensuite une transition de plus pour que l'état des nœuds ne change plus. Nous obtenons un temps de stabilisation d'au plus $n+2$ transitions.

Théorème 6.3 ***(Clôture)***
A partir d'une configuration légale C_i*, sans occurrence de fautes, chaque nœud restera dans une configuration légale.*

Idée de la preuve : Soit C_i une configuration légale. $\forall\, u \in V$, u est fixé et seule la règle R_0 s'exécutera. Nous aurons donc $\forall j > i$, à C_j une configuration légale.

Lemme 6.2 ***(Occupation mémoire)***
L'occupation mémoire est d'au plus $n * \log(2n + k + 3)$.

Idée de la preuve : Chaque nœud, stocke pour chaque voisin l'identité, l'identifiant de *cluster*, le statut et la distance. L'espace mémoire pour chaque voisin est de $\log(2n+k+3)$. De plus, Il stocke aussi les mêmes informations pour lui. Chaque nœud a donc besoin d'un espace mémoire d'au plus $n * \log(2n + k + 3)$.

Remarque 6.1 *(Pire des cas)*
Avec notre approche, nous observons le pire des cas dans une topologie où les nœuds forment une chaine ordonnée. Dans une telle topologie, les nœuds se fixent du plus grand au plus petit et le temps de stabilisation est de $n+2$ *transitions.*

7 Simulations

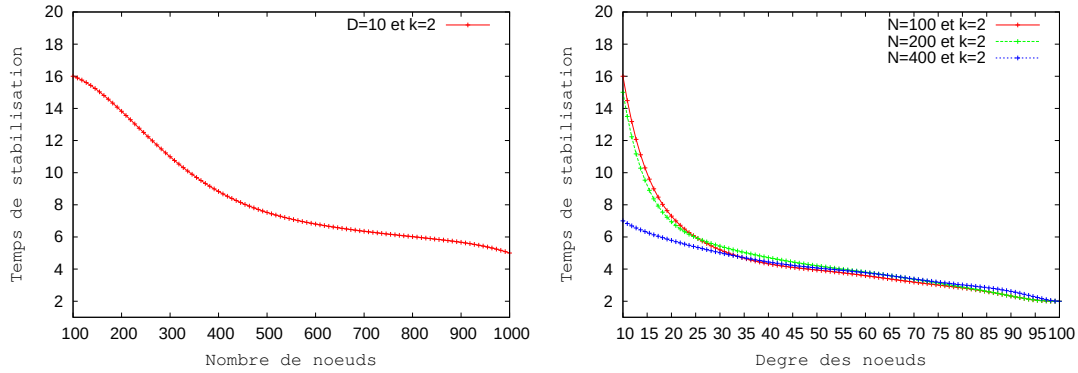
Comme nous l'avons montré au niveau de la section 6, notre réseau converge au plus en $n+2$ transitions. Ceci correspond au cas le plus défavorable d'une topologie où les nœuds forment une chaine ordonnée. Or un réseau *ad hoc* est caractérisé par une topologie dense et aléatoire. C'est ainsi que nous faisons recours à une campagne de simulations pour évaluer les performances moyennes de notre algorithme.
Nous utilisons le simulateur *Omnet++*[1] et la librairie *SNAP*[2] comme générateur de graphes. Toutes nos simulations sont effectuées sous *Grid'5000*[3].

7.1 Impacte de la densité de connexité du réseau sur le temps de stabilisation

Ici nous étudions l'impacte de la densité et le nombre de nœuds sur le temps de stabilisation du réseau. Nous fixons $k = 2$. Pour chaque pas, nous effectuons plusieurs séries de simulations et nous fournissons des valeurs moyennes. Au niveau de la Figure 3(a), nous fixons une densité de connexité de 10% et nous faisons varier le nombre de nœuds de 100 à 1000 par pas de 100. Avec *SNAP*, nous générons des graphes de *Erdos-Reny*. Nous remarquons que le temps de stabilisation diminue avec l'augmentation du nombre de noeuds dans le réseau. En effet, à densité égale, le nombre de voisins augmente quand le nombre de nœuds augmente. Ainsi, durant chaque transition, nous avons plus de nœuds qui se fixent en même temps. De plus, nous remarquons que pour des topologies quelconques, le temps de stabilisation moyen est très en dessous de $n+2$, valeur formelle trouvée dans le pire des cas.
Pour mieux observer l'impacte de la densité du réseau comme illustré avec la Figure 3(b), nous fixons le nombre de nœuds et nous faisons varier le degré (nombre de voisins) des nœuds en générant des graphes *k-réguliers* à l'aide de *SNAP*. Nous observons aussi que pour un nombre de

1. http ://www.omnetpp.org/
2. http ://snap.stanford.edu/index.html.
3. https ://www.grid5000.fr

nœuds fixè à 100, 200 et 400 plus le degré des nœuds augmente, plus le temps de stabilisation diminue. Avec notre approche, plus le réseau est dense, plus nous avons de meilleurs temps de stabilisation. Or les réseau *ad hoc* sont souvent caractérisés être des réseaux à forte densité.



(a) Temps de stabilisation en fonction du nombre de noeuds

(b) Temps de stabilisation en fonction du degré des noeuds

FIGURE 3 – Impacte de la densité et du nombre de noeuds sur le temps de stabilisation

7.2 Étude du passage à l'échelle

Pour étudier le passage à l'échelle de notre approche, nous faisons varier le nombre de nœuds dans le réseau en même temps que la densité de connexité. Pour $k = 2$, nous faisons évoluer le nombre de nœuds de 100 à 1000 par pas de 100. Et pour chaque valeur de nombre de nœuds fixée, nous faisons varier la densité du réseau de 10% à 100% par pas de 10. Nous obtenons la courbe *3D* de la Figure 4. Nous remarquons que sauf pour de faibles densités (10% et 20%), le temps de stabilisation varie légèrement avec l'augmentation du nombre de nœuds. Et cas de faibles densités, nous observons un pic. Mais avec l'augmentation du nombre de nœuds, le temps stabilisation diminue et nous observons le même phénomène que la Figure 3(b). Avec cette série de simulations, nous pouvons soulever deux remarques. *(i)* Seule la densité de connexité est le facteur déterminant avec notre approche et notre solution supporte le passage à l'échelle. *(ii)* En moyenne, pour des réseaux avec une topologie quelconque, le temps de stabilisation est très inférieur à celui du pire des cas ($n + 2$ transitions). Rappelons que ce pire des cas est une topologie où les nœuds forment une chaine ordonnée.

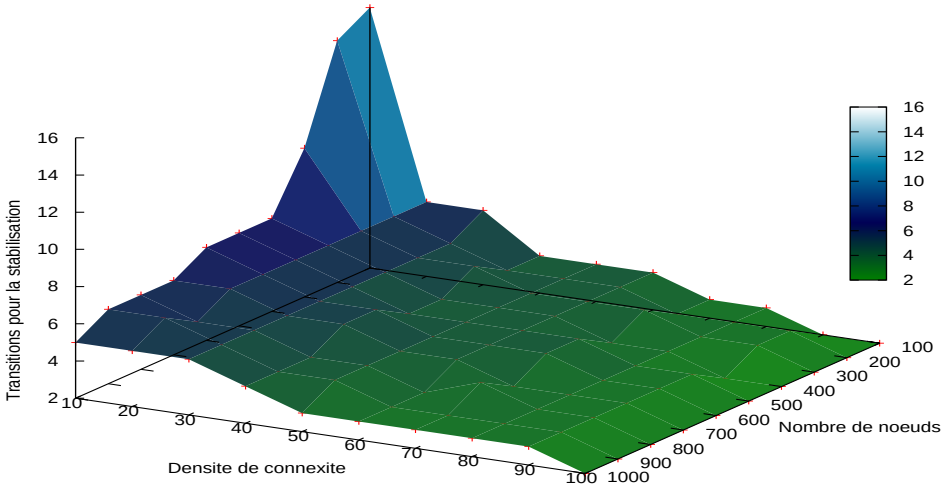
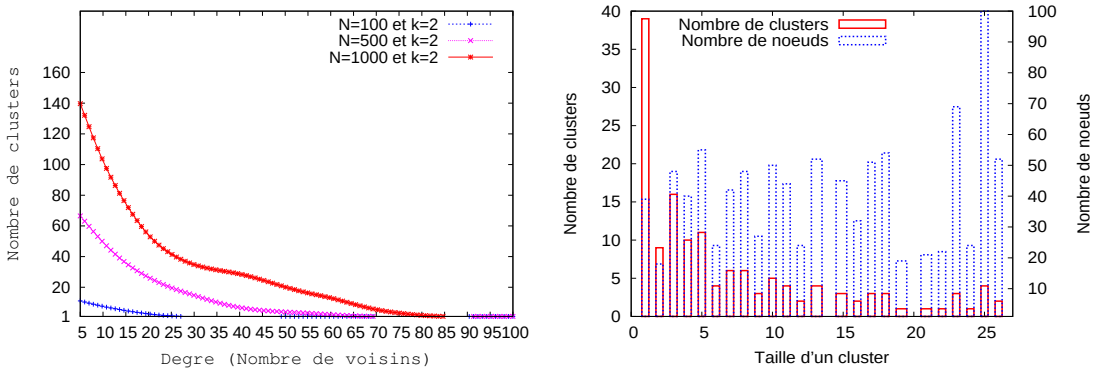


FIGURE 4 – Temps de stabilisation en fonction de la variation de la densité et du nombre de nœuds

7.3 Étude de la taille et le nombre de clusters

Comme la densité est le facteur déterminant pour notre algorithme, nous évaluons le nombre de *clusters* obtenus selon la densité du réseau. Pour $k = 2$, nous fixons le nombre de nœuds à 100, 500 et à 1000. Nous faisons varier le nombre de voisins de 5 à 100 nœuds par pas de 5 et nous

évaluons le nombre *clusters* obtenus. La Figure 5(a) illustre que quelque soit le nombre de nœuds, plus le degré augmente moins on a de *clusters*. En effet, plus le réseau est dense, plus les nœuds les plus grands absorbent au seins de leurs *clusters* beaucoup plus de nœuds.
Comme nous obtenons plus de *clusters* avec de faibles densités, nous fixons un degré de 5 voisins par nœuds pour un réseau de 1000 nœuds. Nous évaluons la répartition des nœuds entre les *clusters*. La première remarque avec la Figure 5(b), est que nous obtenons des *clusters* de taille variable. Nous remarquons toujours avec Figure 5(b) que nous obtenons quelques 39 *clusters* singletons environs 4% du nombre total de nœuds. Nous remarquons aussi que les *clusters* de plus grande identité regroupent plus de nœuds. Ceci est du fait qu'avec notre approche, les nœuds choisissent comme CH le nœud avec la plus grande identité.



(a) Nombre de cluters en fonction du degré

(b) Taille et nombre de clutsers

FIGURE 5 – Taille et nombre de cluters

8 Conclusion

Dans cet article, nous avons présenté un algorithme complètement distribué et auto-stabilisant pour structurer le réseau en *clusters* non-recouvrants à *k sauts*. Sans initialisation, notre solution n'utilise que des informations provenant des nœuds voisins. Elle combine la découverte du voisinage et la construction des *clusters*. Nous avons montré que l'état stable est atteint en au plus $n + 2$ transitions et nécessite au plus une occupation mémoire de $n * \log(2n + k + 3)$. Par simulations, nous avons évalué les performances moyennes de notre approche. En moyenne, nous avons constaté un temps de stabilisation très inférieur à $n + 2$. Dans nos futurs travaux, nous comptons trouver des mécanismes pour équilibrer les *clusters*, les maintenir formés en cas de modifications topologiques et par simulations, comparer notre approche avec les autres solutions existantes.

Bibliographie

1. L. Blin, M. G. Potop-Butucaru, et S. Rovedakis. Self-stabilizing minimum degree spanning tree within one from the optimal degree. *Journal of Parallel and Distributed Computing*, pages 438 – 449, 2011.
2. E. Caron, A. Datta, B. Depardon, et L. Larmore. A self-stabilizing k-clustering algorithm for weighted graphs. *J. Parallel Distrib. Comput.*, pages 1159–1173, 2010.
3. A. Datta, L. Larmore, et P. Vemula. Self-stabilizing leader election in optimal space. In *Stabilization, Safety, and Security of Distributed Systems*, pages 109–123. 2008.
4. E. W Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, pages 643–644, 1974.
5. O. Flauzac, B. S. Haggar, et F. Nolot. Self-stabilizing clustering algorithm for ad hoc networks. *International Conference on Wireless and Mobile Communications*, pages 24–29, 2009.
6. C. Johnen et L. H. Nguyen. Self-stabilizing weight-based clustering algorithm for ad hoc sensor networks. In *Algorithmic Aspects of Wireless Sensor Networks*, pages 83–94. 2006.
7. C. Johnen et L. H. Nguyen. Self-stabilizing construction of bounded size clusters. *International Symposium on Parallel and Distributed Processing with Applications*, pages 43–50, 2008.
8. C. Johnen et L. H. Nguyen. Robust self-stabilizing weight-based clustering algorithm. *Theoretical Computer Science*, pages 581 – 594, 2009.
9. N. Mitton, A. Busson, et E. Fleury. Self-organization in large scale ad hoc networks. 2004.
10. N. Mitton, E. Fleury, I. Guerin L., et S. Tixeuil. Self-stabilization in self-organized multihop wireless networks. In *Proceedings of the Second International Workshop on Wireless Ad Hoc Networking - Volume 09*, ICDCSW '05, pages 909–915, 2005.