Rubrique

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

Mandicou BA [*], Olivier FLAUZAC [*], Bachar Salim HAGGAR [*], Florent NOLOT [*] — Ibrahima NIANG [†]

[*] Université de Reims Champagne-Ardenne
UFR Sciences Exactes et Naturelles
CReSTIC - SysCom
1039-51687 Reims France
{ mandicou.ba, olivier.flauzac, bachar-salim.haggar, florent.nolot}@univ-reims.fr

[†] Université Cheikh Anta Diop
Département de Mathématiques et Informatique
Laboratoire d'Informatique de Dakar (LID)
5005 Dakar-Fann 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* complètement distribué et auto-stabilisant qui construit des *clusters* à *k sauts*. Notre approche ne nécessite pas d'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. Nous montrons que pour un réseau de n nœuds, la stabilisation est atteinte en au plus $n + 2$ transitions.

ABSTRACT. Ad hoc networks offer many application areas because of their easy of deployment. Communications that take 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 clustering algorithm fully distributed and self-stabilizing 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 stabilization is reached at most $n + 2$ transitions, where n is the number of network nodes.

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* [5].

Le *clustering* consiste à découper le réseau en groupes de nœuds appelés *clusters* donnant ainsi au réseau une structure hiérarchique [8]. 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* [2, 3, 7, 9, 10, 12] et à *k sauts* [4, 11]. Cependant, ces approches génèrent beaucoup de trafic et nécessitent d'importantes ressources.

Dans cet article, partant des travaux de Flauzac et *al.* [7], nous proposons un algorithme de *clustering* à *k sauts* qui est complètement distribué et auto-stabilisant. L'auto-stabilisation [6] a été introduite par Dijkstra en 1974 comme étant un système qui, quel que 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 pas d'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. Une conclusion et des perspectives sont données dans la section 7.

2. État de l'art

Plusieurs propositions de *clustering* ont été faites dans la littérature [2, 3, 4, 7, 9, 10, 11, 12].

Les approches auto-stabilisantes [7, 9, 10, 12] construisent des *clusters* à *1 saut*. Mitton et *al.* [12] 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.* [7], ont proposé un algorithme auto-stabilisant qui combine la découverte de topologie et de *clustering* en une seule phase. Elle ne nécessite qu'un unique message échangé entre voisins. Un nœud devient *clusterhead* s'il possède la plus grande identité parmi tous ses voisins. Johnen et *al.* [10] ont proposé un algorithme auto-stabilisant qui construit des *clusters* de taille fixe. Ils attribuent un poids à chaque nœud et fixe un paramètre $SizeBound$ 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 $SizeBound$ nœuds. Dans [9], Johnen et *al.* ont étendu leur proposition décrite dans [10] pour apporter la notion de *robustesse* qui est la propriété qui assure que partant d'une configuration quelconque, le réseau est parti-

tionné après un round asynchrone. Et durant de la phase de convergence, il reste toujours partitionner et vérifie un prédicat de sureté. Dans [2, 3], Bui et *al.* ont proposé un algorithme de *clustering* adaptatif aux changements de topologie mais non auto-stabilisant et non déterministe. Ils utilisent une marchent aléatoire pour construire d'abord un cœur de *cluster* composé de 2 à $MaxCoreSize$ nœuds. Ce cœur est ensuite étendu aux voisins immédiats, appelés nœuds ordinaires, pour former les *clusters*.

Les approches auto-stabilisantes [4, 11] construisent des *clusters* à *k sauts*.

Dans [11], Miton et *al.* étendent leurs travaux décrits dans [12] pour proposer un algorithme robuste de *clustering* auto-stabilisant à *k sauts*. Avec cet algorithme, si un nœud est trop mobile, il risque de ne rattacher à aucun *cluster*. Datta et *al.* [4], utilisant une métrique arbitraire, ont proposé un algorithme auto-stabilisant basé sur un modèle à états nommé *k-Clustering*. Cet algorithme est très lent, 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

Parmi les approches existantes, certaines exigent une connaissance à distance k du voisinage, génèrent beaucoup de messages et ne supportent pas la mobilité [11, 12]. Les propositions [2, 3] ne sont pas auto-stabilisantes, engendrent du trafic et sont lentes à cause de la marche aléatoire utilisée. La solution décrite dans [4] est très couteuse en termes d'espace mémoire et temps de stabilisation. Dans [10, 9] les auteurs n'ont pas spécifié la manière de déterminer les poids associés aux nœuds.

Nous proposons une approche basée sur un modèle à passage de messages complètement distribuée et auto-stabilisante. Elle structure le réseau en *clusters* non-recouvrants de diamètre au plus égal à $2k$. Cette structuration ne nécessite pas de phase d'initialisation. Elle combine la découverte de voisinage et de *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 $hello$.

4. Modèle

Le réseau peut être modélisé par un graphe 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 à *passage de messages*. Pour cela, chaque nœud u envoie périodiquement à ses voisins $v \in N_u$ un message $hello$ contenant les informations sur son état actuel. 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. A la réception d'un message d'un voisin, chaque nœud u met à jour

sa table de voisinage. Des la réception d'un message de tous ses voisins, chaque nœud exécute l'algorithme de *clustering*.

5. Algorithme auto-stabilisant à k sauts

5.1. Préliminaires

Nous donnons quelques définitions utilisées dans la suite.

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.*

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). De plus, chaque nœud choisit un voisin $v \in N_u$, noté gn_u, par lequel il passe pour atteindre son CH.

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 :*

- $statut_u = CH \iff \forall v \in V_{cl_u}, (id_u > id_v) \wedge (dist_{(u,v)} \leq k)$.

*– Simple Node (**SN**) : un nœud u a le statut de SN si tous ses voisins appartiennent au même cluster que lui :*

- $statut_u = SN \iff (\forall v \in N_u, cl_v = cl_u) \wedge (\exists w \in V / (statut_w = CH) \wedge (dist_{(u,w)} \leq k))$.

*– Gateway Node (**GN**) : un nœud u a le statut de GN s'il existe un nœud v dans son voisinage appartenant un autre cluster :*

- $statut_u = GN \iff \exists v \in N_u, (cl_v \neq cl_u)$.

Définition 5.4 *(Nœud cohérent) : Un nœud u est cohérent si et seulement si, il est dans l'un des états suivants :*

– 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 $statut_u = CH$ alors $\{\forall v \in N_u, statut_v \neq CH\} \wedge \{\forall v \in N_u$ tel que $cl_v = cl_u$ alors $(id_v < id_u)\} \wedge \{\forall v \in N_u$ tel que $id_v > id_u$ alors $(cl_v \neq cl_u) \wedge (dist_{(v,CH_v)} = k)\}$.

– Si $statut_u = SN$ alors $\{\forall v \in N_u, (cl_v = cl_u) \wedge (dist_{(u,CH_u)} \leq k)\} \wedge \{\exists v \in N_u, (gn_v = id_u) \wedge (dist_{(v,CH_v)} = dist_{(u,CH_u)} + 1)\}$.

– Si $statut_u = GN$ alors $\exists v \in N_u, (cl_v \neq cl_u) \wedge \{(dist_{(u,CH_u)} = k) \vee (dist_{(v,CH_v)} = k)\}$.

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 du CH de son *cluster* (cl_u), son statut ($statut_u$), et la distance le séparant du CH de son *cluster* ($dist_{(u,CH_u)}$). 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 ($State_u$) contenant l'ensemble des états de ses nœuds voisins. $State_u[v]$ contient les états des nœuds v voisins de u.

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

A la réception de messages $hello$ de tous ses voisins, 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 *Cluster-1* puis le traitement de nouvelles informations est effectué à l'exécution de *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 CH, avec une plus grande identité, alors il devient membre (SN) du *cluster* de v.

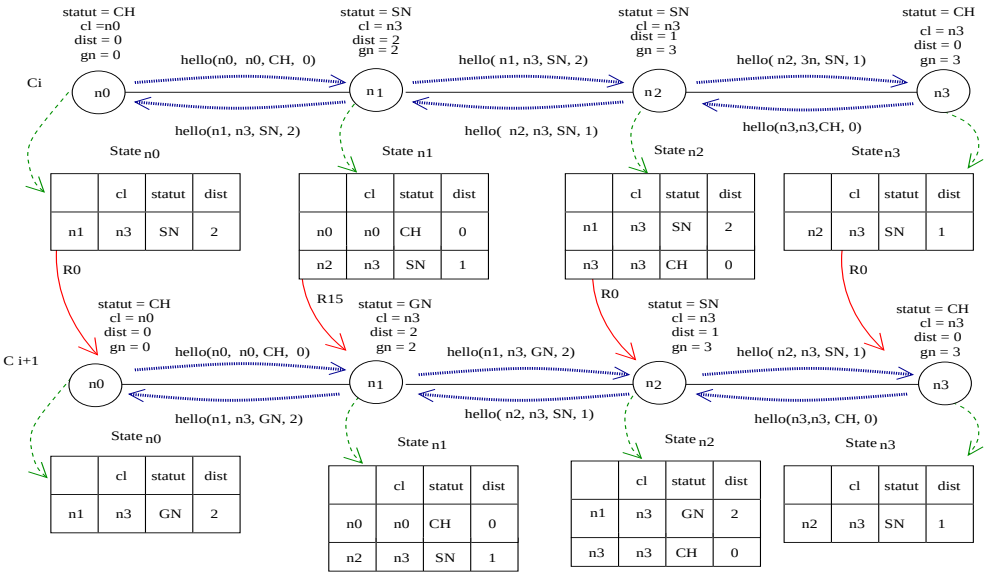

Figure 1. *Exemple de clustering à 2 sauts.*

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 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 1 illustre le passage d'une configuration C_i à C_{i+1}. A C_i, chaque nœud envoie à ses voisins un message *hello*. C_{i+1} est une configuration stable. Dès la réception

de 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'état. C_{i+i} correspond à un état stable.

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

/ Des la réception d'un message hello d'un voisin */*

Prédicats

$P_1(u) \equiv (statut_u = CH)$
$P_2(u) \equiv (statut_u = SN)$
$P_3(u) \equiv (statut_u = GN)$
$P_{10}(u) \equiv (cl_u \neq id_u) \vee (dist_{(u,CH_u)} \neq 0) \vee (gn_u \neq id_u)$
$P_{20}(u) \equiv (cl_u = id_u) \vee (dist_{(u,CH_u)} = 0) \vee (gn_u = id_u)$
$P_{40}(u) \equiv \forall v \in N_u, (id_u > id_v) \wedge (id_u \geq cl_v) \wedge (dist_{(u,v)} \leq k)$
$P_{41}(u) \equiv \exists v \in N_u, (statut_v = CH) \wedge (cl_v > cl_u)$
$P_{42}(u) \equiv \exists v \in N_u, (cl_v > cl_u) \wedge (dist_{(v,CH_v)} < k)$
$P_{43}(u) \equiv \forall v \in N_u / (cl_v > cl_u), (dist_{(v,CH_v)} = k)$
$P_{44}(u) \equiv \exists v \in N_u, (cl_v \neq cl_u) \wedge \{(dist_{(u,CH_u)} = k) \vee (dist_{(v,CH_v)} = k)\}$

Règles

/ Mise à jour du voisinage */*
$StateNeigh_u[v] := (id_v, cl_v, statut_v, dist_{(v,CH_v)})$;

/ Cluster-1 : Gestion de la cohérence */*
$R_{10}(u) : P_1(u) \wedge P_{10}(u) \longrightarrow cl_u := id_u; gn_u = id_u; dist_{(u,CH_u)} = 0;$
$R_{20}(u) : \{P_2(u) \vee P_3(u)\} \wedge P_{20}(u) \longrightarrow$
$statut_u := CH; cl_u := id_u; gn_u = id_u; dist_{(u,CH_u)} = 0;$

/ Cluster-2 : Clustering */*
$R_{11}(u) : : \neg P_1(u) \wedge P_{40}(u) \longrightarrow$
$statut_u := CH; cl_u := id_v; dist_{(u,CH_u)} := 0; gn_u := id_u;$
$R_{12}(u) : : \neg P_1(u) \wedge P_{41}(u) \longrightarrow$
$statut_u := SN; cl_u := id_v; dist_{(u,v)} := 1; gn_u := NeighCH_u;$
$R_{13}(u) : : \neg P_1(u) \wedge P_{42}(u) \longrightarrow$
$statut_u := SN; cl_u := cl_v; dist_{(u,CH_u)} := dist_{(v,CH_v)} + 1; gn_u := NeighMax_u;$
$R_{14}(u) : : \neg P_1(u) \wedge P_{43}(u) \longrightarrow$
$statut_u := CH; cl_u := id_v; dist_{(u,CH_u)} := 0; gn_u := id_u;$
$R_{15}(u) : : P_2(u) \wedge P_{44}(u) \longrightarrow statut_u := GN;$
$R_{16}(u) : : P_1(u) \wedge P_{41}(u) \longrightarrow$
$statut_u := SN; cl_v := id_v; dist_{(u,v)} := 1; gn_u := NeighCH_u;$
$R_{17}(u) : : P_1(u) \wedge P_{42}(u) \longrightarrow$
$statut_u := SN; cl_u := cl_v; dist_{(u,CH_u)} := dist_{(v,CH_v)} + 1; gn_u := NeighMax_u;$

/ Envoi d'un message hello */*
$R_0(u) : hello(id_u, cl_u, statut_u, dist_{(u,CH_u)});$

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.

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

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

6. Schéma intuitif de la preuve de la stabilisation

Dans cette section, nous énonçons les principaux théorèmes et propriétés vérifiés par notre algorithme. Par manque 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 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.

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. 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. Dans nos futurs travaux, nous évaluerons les performances moyennes de notre approche par le biais de simulations, car dans la plus part des cas, nous avons constaté un temps de stabilisation inférieur à $n + 2$.

Références

[1] Lalia Blin, Maria Gradinariu Potop-Butucaru, and Stephane Rovedakis. Self-stabilizing minimum degree spanning tree within one from the optimal degree. *Journal of Parallel and Distributed Computing*, pages 438 – 449, 2011.

[2] Alain Bui, Abdurusul Kudireti, and Devan Sohier. A fully distributed clustering algorithm based on random walks. *Parallel and Distributed Computing, International Symposium on*, 2009.

[3] Alain Bui, Devan Sohier, and Abdurusul Kudireti. A random walk based clustering with local recomputations for mobile ad hoc networks. *Parallel and Distributed Processing Workshops and PhD Forum, 2011 IEEE International Symposium on*, 2010.

[4] Eddy Caron, Ajoy K. Datta, Benjamin Depardon, and Lawrence L. Larmore. A self-stabilizing k-clustering algorithm for weighted graphs. *J. Parallel Distrib. Comput.*, pages 1159–1173, 2010.

[5] Ajoy Datta, Lawrence Larmore, and Priyanka Vemula. Self-stabilizing leader election in optimal space. In Sandeep Kulkarni and AndrÃ© Schiper, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 109–123. 2008.

[6] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, pages 643–644, 1974.

[7] Olivier Flauzac, Bachar Salim Haggar, and Florent Nolot. Self-stabilizing clustering algorithm for ad hoc networks. *Wireless and Mobile Communications, International Conference on*, pages 24–29, 2009.

[8] Colette Johnen and Le Nguyen. Self-stabilizing weight-based clustering algorithm for ad hoc sensor networks. In *Algorithmic Aspects of Wireless Sensor Networks*, pages 83–94. 2006.

[9] Colette Johnen and Le Huy Nguyen. Robust self-stabilizing weight-based clustering algorithm. *Theoretical Computer Science*, pages 581 – 594, 2009.

[10] Colette Johnen and LeHuy Nguyen. Self-stabilizing construction of bounded size clusters. *Parallel and Distributed Processing with Applications, International Symposium on*, pages 43–50, 2008.

[11] N. Mitton, E. Fleury, I. Guerin Lassous, and 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.

[12] Nathalie Mitton, Anthony Busson, and Eric Fleury. Self-organization in large scale ad hoc networks. 2004.