Rapport présenté pour l'obtention de
l'Habilitation à Diriger des Recherches

Quelques contributions à l'auto-stabilisation

Colette Johnen

Soutenu le 13 novembre 2007

Composition du jury :

Joffroy Beauquier, Université Paris-Sud 11, France

Marc Bui, Université Vincennes Saint denis, Paris 8, France (rapporteur)

Carole Delporte-Gallet, Université Paris-Diderot 7 France

Shlomi Dolev, Ben-Gurion, University of Negev, Israel (rapporteur)

Jean-Frédéric Myoupo, Université de Picardie Jules Vernes, France (président)

Masafumi Yamashita, Kyushu University, Japon (rapporteur)

Celui qui combat peut perdre, mais celui qui ne combat pas a déjà perdu. [Bertolt Brecht]

Préambule

Je me rappelle encore très nettement de mes premiers pas au LRI, en 1983, jeune, très jeune étudiante de maîtrise, j'ai passée le mois de juillet à faire un stage d'été sous la direction de Daniel Kaiser avec Catherine Recanati.

Ce premier stage a été le début d'une long cheminement personnel et professionnel (avec) et au sein du LRI.

Remerciement

Merci aux directrices et directeurs successifs du LRI (Gérard Roucairol, Marie-Claude Gaudel, Joffroy Beauquier, Laurence Puel, Dominique Gouyou-Beauchamps et Michel Beaudoin-Lafon) qui ont eu à coeur de construire et de maintenir un environnement propice au travail, tout en gardant une ambiance chaleureuse.

Les membres des services techniques et administratifs du LRI, toujours efficace et à l'écoute des besoins et des demandes des chercheurs, sont pour une très grande part dans l'ambiance très agréable, du LRI, ainsi que de son efficacité. Je profite donc de cette occasion pour les en remercier et leur témoigner ma reconnaissance.

Merci à Joffroy Beauquier, j'ai énormément appris à son contact sur le métier de chercheur et sur les qualités nécessaires à un bon directeur de recherche ; et si je me sens aujourd'hui capable d'encadrer des jeunes chercheurs, d'animer une équipe, c'est grâce à son exemple.

Merci à Maria Gradinariu et Le Huy Nguyen, mes ex-thésards, qui m'ont beaucoup appris.

Merci aux co-auteurs de mes différents papiers sans qui aucune publication n'aurait été possible. Je voudrais citer plus particulièrement Lisa Higham avec qui je travaille depuis plusieurs années, et j'espère pour encore de nombreuses années, tant pour la collaboration fructueuse que pour les liens d'amitiés tissés entre nous.

Merci à tous les collègues que j'ai rencontré au cours de séminaires, workshop, conférences avec qui j'ai échangé, discuté et bataillé ; ces échanges ont toujours été très stimulants.

Merci à Véronique Vèque, qui a partagé mon bureau, notre camaraderie s'est transformée en une solide amitié.

J'ai été touché par l'amabilité des membres de mon jury, qui ont fait preuve d'une grande disponibilité. Merci à Marc Bui, Shlomi Dolev et Masafumi Yamashita qui m'ont fait l'honneur de donner leur appréciation sur mon rapport « d'habilitation à diriger des recherches ». Merci à Jean-Frédéric Myoupo d'avoir présidé mon jury d'habilitation avec beaucoup de gentillesse. Merci à Joffroy Beauquier et Carole Delporte-Gallet de leur participation au jury.

Merci aux diverses équipes de direction de l'IUT d'Orsay et à tous les membres du département Informatique de l'IUT d'Orsay qui ont compris les difficultés intrinsèques à la réalisation d'une recherche de qualité tout en participant activement à la gestion (administrative et pédagogique) du département informatique. Leur compréhension a été d'une grande aide.

Merci à Marie-Claude Heydemann qui a été un modèle pour moi de professionnalisme, par son énergie, sa gentillesse avec les étudiants, les enseignants de l'IUT et avec moi.

Un grand merci à Dominique Gouyou-Beauchamps, Marianne Habrestrau, Annie Leredde, Lydia Nicaud, Nicole Polian et Alain Vauchelles, collègues de l'IUT d'Orsay, qui au fil des années, sont devenus des amis et qui ont été d'un grand soutien.

Merci aux « jeunes » collègues de l'IUT pour leur dynamisme et leur amour du métier d'enseignant/chercheur (Cédric Bastoul, Pascal Berthomé, Sylvie Delaët).

Merci aux intervenants dans les divers modules d'enseignement que j'animais et que j'anime, leur professionnalisme m'a permis de consacrer du temps à mes travaux de recherche. En particulier un grand merci à Emmanuel Motchane, qui, depuis son arrivé à l'IUT, me soulage de beaucoup de tâches système/réseau à l'IUT.

Merci à tous ceux qui ne sont pas nommément cités, mais qui ont participé de près ou de loin, à mon travail, les membres de l'équipe parallélisme du LRI, et du projet INRIA Future, grand large, ...

Merci enfin aux gens extérieurs au monde du travail qui font que je me sens bien dans ma vie : ma famille, mes proches et mes amis.

Chapitre 1

Introduction

Un système réparti est un ensemble d'entités autonomes et communicantes. Chaque entité (aussi appelé nœud, processus, processeur, machine, ...) a son propre code, ses propres objectifs, sa vitesse d'exécution, ... Ces entités sont communicantes, elles peuvent échanger entre elles des informations. Un élément important d'un système réparti est le graphe de communication (aussi appelé réseau d'interconnection, plus simplement réseau) qui définit quelle entité peut communiquer avec quelles autres entités. L'objectif d'un système réparti est de fournir un service qui ne pourrait pas être réalisé par une seule entité soit en terme de fonctionnalité, de disponibilité, de temps de réponse, de fiabilité, ...

Les systèmes répartis sont particulièrement sujet aux pannes. Car chaque entité est susceptible de tomber en panne. La taxonomie classique des pannes est la suivante :

- panne franche (en anglais, panne crash) : soit un élément fonctionne normalement (son comportement est standard), soit il ne fait rien (il est en panne).
- panne intermittente : un élément arrête de fonctionner pour une durée arbitraire, pour ensuite reprendre un fonctionnement correct. Ce type de panne correspond à la perte de messages, la rupture des liens de communication entre les entités du système réparti.
- panne byzantine ou attaque : un élément du système a un comportement non standard, qui peut être aléatoire ou pire correspondre à une attaque en règle du système. Ce type de pannes correspond au bogue existant dans la conception et réalisation des éléments du système ; mais aussi à l'insertion dans le système d'un « cheval de Troie »(d'un élément hostile, dont le but est de stopper la bonne marche du système réparti).

Les systèmes répartis sont aussi sujet aux modifications du graphe d'interconnexion. Les entités peuvent être itinérantes ou mobiles. Des liens de communication peuvent être ajoutés, des réseaux peuvent être connectés ou déconnectés.

Bref, de manière générale, un système réparti doit pouvoir gérer des événements de natures diverses qui perturbent son fonctionnement. Il est donc important d'avoir une algorithmique répartie qui s'adapte au mieux à un environnement changeant. Il existe deux classes d'algorithmes tolérant les défaillances/perturbations.

- Les algorithmes masquant les défaillances. Un algorithme masquant assure la continuité des fonctionnalités du système même en présence de perturbations. Ce type d'algorithmes

tolère des défaillances frappant continûment le système. Il s'agit d'une garantie très importante qui est coûteuse et complexe à mettre en œuvre. Une telle garantie est proposée et vérifiée uniquement pour les systèmes critiques dont le dysfonctionnement peut mettre en danger la vie d'êtres humains (par exemple, système de contrôle des avions, des centrales nucléaires, ...). De plus, ces algorithmes ne tolérent qu'une classe restreinte de défaillances

- Les algorithmes non masquant. Un algorithme non masquant ne garantit pas la continuité du service, mais il garantit la reprise du service après la fin des défaillances. Ce type d'algorithmes permet de tolérer toutes les perturbations sur un laps de temps fini. Après la fin de la rafale de perturbations, le système va s'auto-corriger pour reprendre un fonctionnement correct après une phase transitoire où le système ne fonctionne pas encore correctement, bien qu'il n'y ait plus de perturbations.

Edsger Dijkstra [Dij74] a défini la notion d'auto-stabilisation comme suit *Quelle que soit sa configuration initiale, un système auto-stabilisant est certain d'atteindre une configuration légitime en un nombre fini d'étapes.* Les algorithmes auto-stabilisants appartiennent à la catégorie des algorithmes non masquant tolérant toutes les pannes transitoires. Une panne transitoire advient uniquement durant une période de temps limité. George Varghese and Mahesh Jayaram, [VJ00], ont établi qu'une configuration quelconque du réseau peut être atteinte après une série quelconque de pannes franches, de remises en marche des éléments du réseau et de pertes et déséquencement des messages, ... Ainsi un algorithme auto-stabilisant tolère une série arbitraire de défaillances transitoires de durée finie. Après cette série, l'algorithme auto-stabilisant conduit le système dans une configuration légitime. Une fois une configuration légitime atteinte, le système fonctionne correctement, effectuant normalement ces tâches.

Dans ce document, je présente une partie de mes travaux de recherche concernant les algorithmes auto-stabilisants. Je me suis attachée à détailler l'imbrication de mes travaux avec les travaux des autres chercheurs. Je me suis volontairement limitée aux travaux concernant les trois axes suivants :

- étude théorique des modèles de communication et de calcul. Dans le chapitre 2, les différents modèles de communication par registre et de calcul utilisés dans l'algorithmique réparti sont présentés et comparés. Pour comparer la puissance de ces modèles, j'étudie la réalisation de compilateurs tolérants aux défaillances d'un modèle à l'autre.
- contribution à l'algorithmique répartie et classique. Deux problèmes « étalons » de l'informatique répartie : l'élection de leader et l'exclusion mutuelle sont intensivement étudiés dans le chapitre 3. Je me suis attaché à étudier les bornes à l'espace mémoire nécessaire à ces deux tâches dans le cadre d'anneaux anonymes.
- Algorithmique pour les réseaux Ad-Hoc. Dans le chapitre 4, des algorithmiques tolérants aux défaillances de maintenance de réseaux Ad-Hoc sont présentés.

Dans la continuité des trois axes de recherches présentés dans ce document sont proposés des perspectives de recherche, dans le chapitre 5.

Les références bibliographiques en gras sont celles signées ou co-signées par moi-même.

Chapitre 2

Compilateurs tolérants aux pannes

Pour construire et surtout pour prouver l'exactitude d'un algorithme réparti, il primordial de définir le modèle de communication mis en place entre les processus : communication par messages, communication par registres, ainsi que le modèle de calcul. Etablir que la communication est réalisée via des registres partagés n'est pas suffisant, il faut aussi définir la sémantique associée aux registres, le type des registres. Je présente dans ce chapitre aux différents modèles de communication par registres et modèle de calcul.

L'objectif de ce chapitre, est de comparer ces différents modèles en présentant des compilateurs tolérants aux pannes d'un modèle à l'autre ou en prouvant l'impossibilité de construire de tel compilateur.

2.1 Modèles de communication par registres

Le réseau d'interconnexion d'un système réparti est souvent modélisé par un graphe où les nœuds représentent les processeurs. Il existe un arc entre deux nœuds si et seulement si les deux processeurs peuvent communiquer directement (ces processeurs sont dit voisins).

Je me suis intéressée aux modèles de communication entre deux voisins par registres partagés. Un registre est une zone mémoire qui peut être lue par certains processus et dont le contenu est modifiable uniquement par le processus propriétaire de la zone mémoire.

2.1.1 Location d'un registre

Un processus peut avoir un seul registre de type multi-lecteurs. Ce registre est lisible par tous ses voisins. Ainsi en une seule étape, il communique à tous ses voisins son nouvel état. Ce type registre est localisé sur le nœud (processus).

Par contre, un processus peut avoir autant de registres que de voisins, un registre par lien de communication, chaque registre étant du type mono-lecteur. Dans ce cas, informer tous ses voisins d'un changement d'état est effectué en plusieurs pas de calcul : le processus doit écrire dans tous les registres sortants. On dit que les registres sont localisés sur les liens.

2.1.2 Sémantique d'un registre

Une opération sur un registre (lecture ou écriture) n'est pas instantanée, elle a une durée dépendant du système d'exploitation, du hardware, ... Donc à chaque opération sont associés deux temps : un temps de début de réalisation (nommé Td) et le temps de terminaison de l'opération (nommé Tf). Il est donc possible que deux opérations sur le même registre se chevauchent (ces opérations sont dit concurrentes). Par exemple, dans la figure 2.1, la lecture2 chevauche l'écriture de la lettre b dans le registre, car $Td_e < Tf_2 < Tf_e$. La lecture3 chevauche l'écriture de la lettre b dans le registre parce que nous avons $Td_e < Td_3 < Tf_e$. Une lecture dans un registre qui n'est pas concurrente à une écriture dans ce registre retourne le contenu du dit registre. Par exemple, la lecture1 de la figure 2.1 retourne toujours la valeur a alors que la lecture4 retourne toujours la valeur b. Par contre, la valeur retournée par une lecture qui est concurrente à une écriture varie d'un registre à l'autre en fonction de sa sémantique.

Leslie Lamport [Lam86] a défini 3 trois sémantiques : sûr (safe), régulier (regular) et atomique (atomic). La sémantique d'un registre est fonction du résultat retourné par la lecture du registre lorsque cette lecture chevauche une écriture sur ce registre.

Dans le cas de registres **sûrs** , la valeur retournée est quelconque. Par exemple, la valeur retournée de lecture2 et lecture3 dans la figure 2.1 peuvent être a, b ou n'importe quel caractère. Ce type de registre peut être facilement construit, mais il est complexe à utiliser. C'est-à-dire que la conception de programme réparti cohérent utilisant ce type de registre demande une grande expertise.

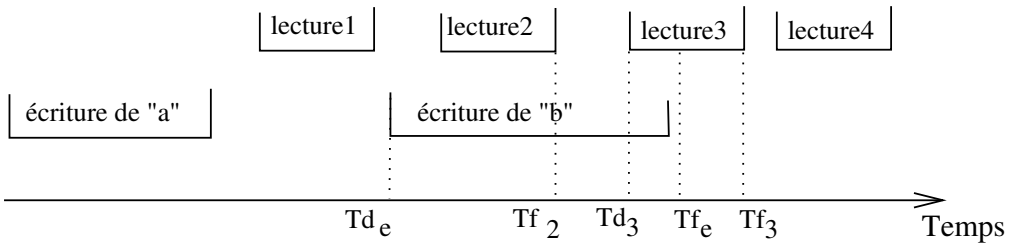

FIG. 2.1 – Opérations sur un même registre qui se chevauchent

Les registres **réguliers** assure une faible cohérence : la valeur retournée par une lecture chevauchant une écriture est la valeur écrite ou la valeur qui était stockée dans le registre. Par exemple, la lecture2 (ainsi que la lecture3) de la figure 2.1 peuvent retourner deux valeurs : a ou b. Ce type de registres reste relativement facile à implémenter, tout en étant plus facile à utiliser que les registres uniquement sûrs.

Les registres **atomiques** sont les plus facile à programmer, mais ils sont très difficiles à implémenter. Les registres atomiques sont des registres réguliers. Une fois que la valeur en cours d'écriture est lu, toutes les autres lectures du registre doivent retourner cette nouvelle valeur. Par exemple, la lecture2 de la figure 2.1 peuvent retourner deux valeurs : a ou b. Si la lecture2 de la figure 2.1 retourne b alors la lecture3 doit aussi retourner la valeur b. Dans le cas de registres atomiques, les opérations de lecture et d'écriture semblent être instantanées [HW90].

Il existe donc 6 modèles de communication basés sur les registres en fonction de la sémantique et de la location des registres (voir la figure 2.2).

	location sur les nœuds	*location sur les liens*
registre atomique	NOEUD-ATOMIQUE	LIEN-ATOMIQUE
registre régulier	NOEUD-RÉGULIER	LIEN-RÉGULIER
registre sûr	NOEUD-SÛR	LIEN-SÛR

FIG. 2.2 – Six modèles de communication par registre

Etant donné une graphe G, la notation LOCATION-SÉMANTIQUE(G) nomme le réseau de topologie G dont le modèle de communication est de type LOCATION-SÉMANTIQUE. Par exemple, noeud-atomique(G) est le réseau de topologie G tel que chaque processus a un seul registre atomique de type mono-écrivain et multi-lecteurs.

De nombreux algorithmes auto-stabilisants ont été conçus pour le cadre du modèle de communication LIEN-ATOMIQUE [Dol00]. D'autres papiers [MN98, AS99a, Hua00, NA02b], ont utilisé le modèle NOEUD-ATOMIQUE. Cet état de fait a été notre première motivation, pour étudier les compilateurs d'un modèle à l'autre. Le but est de transformer un algorithme conçu pour un modèle précis (appelé modèle d'*origine*) en un algorithme conçu pour un autre modèle (appelé modèle *cible*).

2.1.3 Transformations et compilateurs

La transformation d'un algorithme conçu pour un modèle d'origine en un autre modèle cible est dite **valide syntaxiquement** si elle est consiste à remplacer chaque opération du modèle d'origine par un programme contenant uniquement des opérations réalisables dans le modèle cible. Par exemple, pour transformer un algorithme conçu pour le réseau noeud-régulier(G) en un algorithme exécutable sur le réseau lien-régulier(G), nous devons remplacer chaque NOEUD-RÉGULIER-ÉCRITURE et chaque NOEUD-RÉGULIER-LECTURE opération du processus p par un programme utilisant uniquement les opérations LIEN-RÉGULIER-ÉCRITURE et LIEN-RÉGULIER-LECTURE réalisables par le processus p.
Ainsi, une transformation d'algorithmes de noeud-régulier(G) à lien-régulier(G) est juste l'application de la fonction τ ; en supposant que τ(NOEUD-RÉGULIER-ÉCRITURE($\mathcal{R}$, ν)) et τ(NOEUD-RÉGULIER-LECTURE($\mathcal{R}$)) sont des programmes réalisables dans lien-régulier(G). *-transformation naïve (Algorithme 1) est une transformation syntaxiquement valide des algorithmes conçus pour noeud-*(G) en algorithmes exécutables dans lien-*(G).

Je m'intéresse aux transformations syntaxiquement valides qui préservent **la validité sémantique** (correctness) de l'algorithme d'origine. La « validité sémantique » d'un algorithme est défini par un prédicat sur les exécutions de l'algorithme. Un système SO est défini par le triplet (G, LOCATION-SÉMANTIQUE, Alg) où G est un graphe, LOCATION-SÉMANTIQUE est un modèle de communication et Alg est un algorithme. Nous associons très naturellement un ensemble d'exécutions EO à SO : l'ensemble des exécutions de Alg sur le réseau LOCATION-SÉMANTIQUE(G). Soit τ une transformation de LOCATION-SÉMANTIQUE dans le modèle cible nommé LOCCIBLE-EFFCIBLE.

Algorithme 1 *-Transformation Naïve

paramètre :

* - indique la sémantique des registres

code de p :

τ(NOEUD-*-ÉCRITURE($\mathcal{R}_p$, ν)) $\equiv$ for every q in neighbourhood of p, LIEN-*-ÉCRITURE($\mathcal{R}_{pq}$, ν)

τ(NOEUD-*-LECTURE($\mathcal{R}_q$)) $\equiv \nu \longleftarrow$ LIEN-*-LECTURE($\mathcal{R}_{qp}$) ; return ν

Nous notons $\tau(Alg)$ l'algorithme Alg après avoir été transformé. Les exécutions associées au système SC = (G, LOCCIBLE-EFFCIBLE, τ(Alg)) sont interprétables comme exécutions effectuées sur LOCATION-SÉMANTIQUE(G). Ces interprétations doivent être réalisable par le système SO. Précisément, EC - l'ensemble des exécutions associées à SC après avoir été interprétées - doit être un sous-ensemble de EO. Dans ce dernier cas, on dit que τ est une transformation sémantiquement valide SO dans le modèle LOCCIBLE-EFFCIBLE. Par exemple, l'interprétation de l'exécution présentée dans la figure 2.3 ne peut pas être réalisée dans le réseau noeud-atomique(G). Deux lectures de $\mathcal{R}_p$ chevauchent la période de temps où un processus exécute τ(NOEUD-ATOMIQUE-LECTURE($\mathcal{R}_p$)).
La première exécution de τ(NOEUD-ATOMIQUE-LECTURE($\mathcal{R}_p$)) retourne la valeur en cours d'écriture b, la deuxième exécution de τ(NOEUD-ATOMIQUE-LECTURE($\mathcal{R}_p$)) (qui a lieu plus tard) ne retourne pas cette valeur, aucune exécution dans le modèle NOEUD-ATOMIQUE ne peut produire un tel comportement.

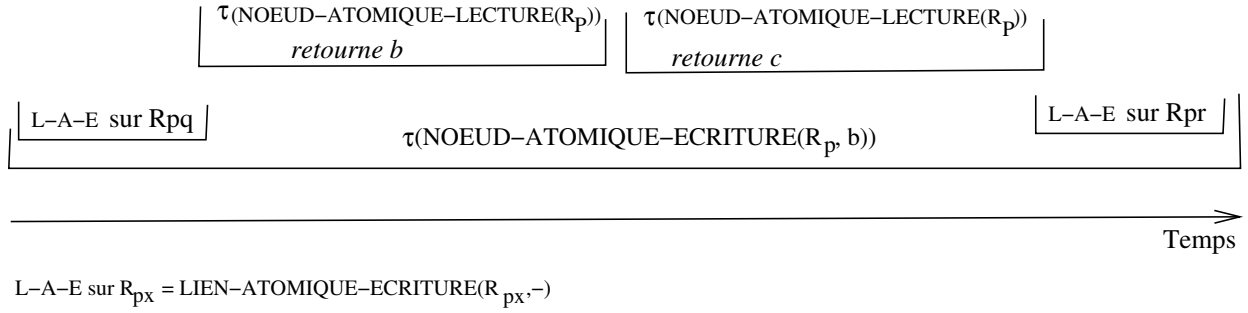

FIG. 2.3 – Exécution d'un algorithme conçu, pour le modèle NOEUD-ATOMIQUE transformé dans le modèle LIEN-ATOMIQUE, par « atomique-transformation naïve »

A représente une classe de programmes pour le modèle LOCATION-SÉMANTIQUE. Une transformation τ est un **compilateur** des algorithmes de A du modèle LOCATION-SÉMANTIQUE dans le modèle LOCCIBLE-EFFCIBLEsi τ est une transformation valide syntaxiquement et sémantiquement de tous les systèmes définis par S = (G, LOCATION-SÉMANTIQUE, Alg) sur le réseau LOCCIBLE-EFFCIBLE(G) et ceci quelle que soit la topologie du graphe G et pour tous les algorithmes Alg de A.

2.1.4 Compilateurs tolérant aux pannes

Un compilateur est dit **sans-attente** (Wait-Free, en anglais) s'il préserve la propriété « d'avancement ». Une exécution dans un système réparti est sans-attente si elle termine proprement après un nombre borné d'étapes quelles que soient les actions des autres processus. Une telle qualité implique que les pannes franches d'un sous-ensemble de processus ne peuvent pas empêcher les autres processus de terminer correctement leur exécution. Précisément, les interprétations des exécutions associées au système $SC = (G,$ locCible-effCible, $\tau(\text{Alg}))$ doivent être des exécutions réalisables dans le système d'origine. De plus, elles doivent être sans-attente si les exécutions du système d'origine sont sans-attente. Un tel compilateur assure la terminaison des exécutions de $\tau($location-sémantique-lecture$(\mathcal{R}))$ et de $\tau($location-sémantique-écriture$(\mathcal{R}, \nu))$ sans attente indéfinie, dans tous les contextes réalisables d'exécution.
Concrètement, il suffit que les exécutions de $\tau($location-sémantique-lecture$(\mathcal{R}))$ et de $\tau($location-sémantique-écriture$(\mathcal{R}, \nu))$ soient toujours effectuées en un nombre borné d'étapes. Une telle qualité implique que les pannes franches ne peuvent pas empêcher la terminaison des exécutions de $\tau($location-sémantique-lecture$(\mathcal{R}))$ et de $\tau($location-sémantique-écriture$(\mathcal{R}, \nu))$ par les processus qui ne sont pas en panne. Ainsi, ces processus peuvent compléter leur exécution. Par exemple, l'algorithme régulière-transformation naïve (Algorithme 1) est une transformation sans-attente.

Un compilateur est dit **auto-stabilisant** s'il préserve la propriété auto-stabilisation. Un système réparti est auto-stabilisant si toutes les exécutions atteigne une configuration légitime, quelle que soit leur configuration initiale. La légitimité est définie par un prédicat sur les configurations du système. Une configuration est l'état global du système, elle est constituée des éléments suivants : états des variables des processus, le contenu des registres de communication et la valeur du compteur ordinal de chaque processus. Une telle qualité implique que les pannes transitoires ne peuvent pas empêcher le système d'atteindre une configuration à partir de laquelle il fonctionnera correctement. Précisément, les interprétations des exécutions associées au système $SC = (G,$ locCible-effCible, $\tau(\text{Alg}))$ doivent être à terme des exécutions réalisables dans le système d'origine, quelle que soit la configuration initiale. Dans ce cas, si le système d'origine était auto-stabilisant, le système compilé l'est aussi.

2.1.5 Compilateurs existants

Dans [Lam86], Leslie Lamport a présenté un compilateur sans-attente d'un registre atomique mono-écrivain et mono-lecteur par des registres réguliers mono-écrivain et mono-lecteur, ce compilateur utilise deux registres, ainsi chaque processus peut écrire dans un registre et lire dans l'autre. Cette transformation est aussi un compilateur sans-attente de modèle lien-atomique dans le modèle lien-régulier. Il est trivial d'établir que ce compilateur est aussi auto-stabilisante.

Dans la même publication, Leslie Lamport a présenté un compilateur sans-attente d'un registre régulier mono-écrivain et mono-lecteur par des registres sûrs mono-écrivain et mono-lecteur. Ce compilateur est aussi un compilateur sans-attente de lien-régulier modèle dans le modèle lien-sûr. Dans [HPT02], il a été établi que ce compilateur n'est pas auto-stabilisant.

La régulière-transformation naïve (Algorithme 1) est un compilateur sans attente et auto-stabilisante du modèle noeud-régulier dans le modèle lien-régulier. La sûre-transformation

naïve est un compilateur sans attente et auto-stabilisante du modèle NOEUD-SÛR dans le modèle LIEN-SÛR.

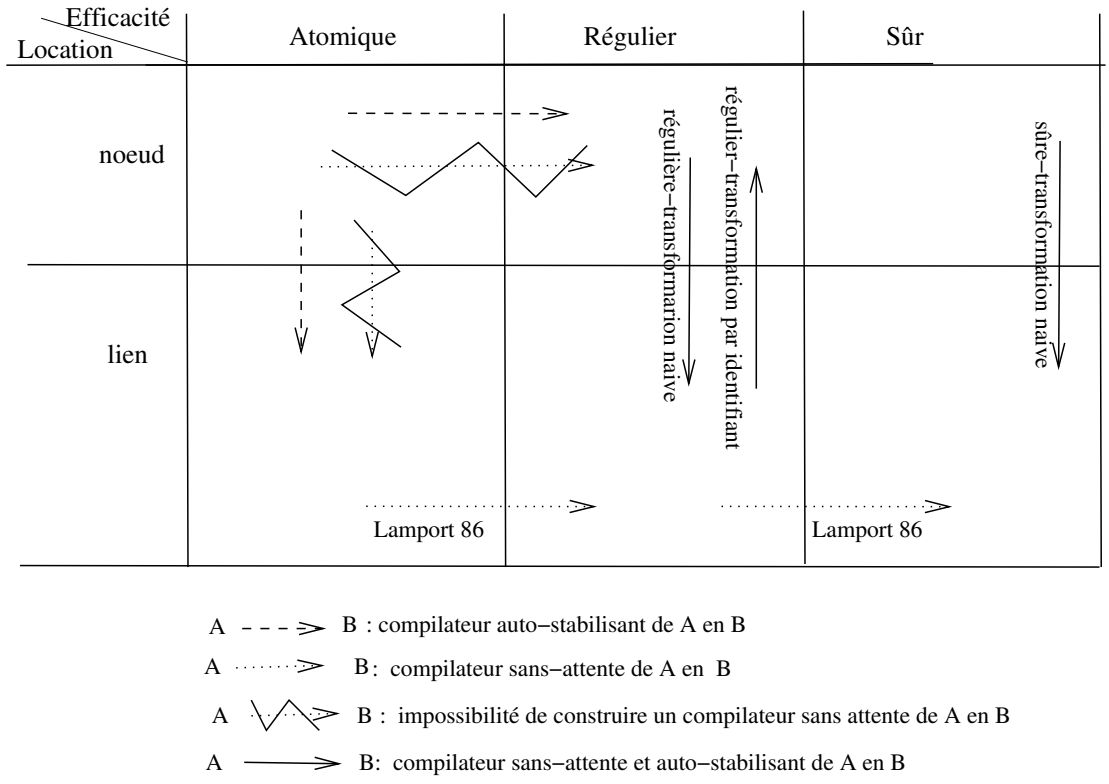

FIG. 2.4 – Compilateurs tolérant aux pannes

L'atomique-transformation par identifiant (Algorithme 2) est un compilateur sans attente et auto-stabilisante du modèle LIEN-ATOMIQUE dans le modèle NOEUD-ATOMIQUE. La régulière-transformation par identifiant est un compilateur sans attente et auto-stabilisant de LIEN-RÉGULIER modèle dans le modèle NOEUD-RÉGULIER.

Lisa Higham et moi avons conçu un compilateur auto-stabilisant du modèle NOEUD-ATOMIQUE dans le modèle LIEN-ATOMIQUE. Ce compilateur fut présenté à « l'IEEE International Parallel & Distributed Processing Symposium » en 2006 **[HJ06a]**. La procédure τ(NOEUD-ATOMIQUE-ÉCRITURE($\mathcal{R}$, ν)) et la fonction τ(NOEUD-ATOMIQUE-LECTURE($\mathcal{R}$)) ne sont pas sans attente. La procédure τ(NOEUD-ATOMIQUE-ÉCRITURE($\mathcal{R}$, ν)) nécessite au moins 2Δ opérations de communication. Une opération de communication est une lecture valide ou une écriture valide dans le modèle cible (dans ce cas, le modèle LIEN-ATOMIQUE).
La fonction τ(NOEUD-ATOMIQUE-LECTURE($\mathcal{R}$)) est à terme sans goulot d'étranglement (traduction libre du terme anglais « obstruction-free »). Une fonction est dite « sans goulot d'étranglement », [HLMI03], si elle se termine en un nombre fini d'étapes lorsque les autres processus sont gelés (ils n'ont aucune opération en cours). Une fois le système stabilisé, un processus p obtiendra le contenu du registre du processus q en réalisant moins de 2Δ opérations de communication, dans le cas où q n'exécute pas la procédure τ(NOEUD-ATOMIQUE-ÉCRITURE($\mathcal{R}$, ν)).

Algorithme 2 *-transformation par identifiant

paramètre :
* - indique la sémantique des registres.
Δ : degré maximum d'un nœud du réseau.
id_p : l'identifiant de p.

Structure de $\mathcal{R}_p$:
tableau de Δ éléments. $\mathcal{R}_p$[i] a deux champs : *nom* et *valeur*.

code de p :
τ(LIEN-*-ÉCRITURE($\mathcal{R}_{pq}$, ν))
$AR \longleftarrow$NOEUD-*-LECTURE($\mathcal{R}_p$) ;
i $\longleftarrow$ 0 ; **while** (AR[i].nom $\neq id_q$) **do** i++ ; **done**
AR[i].value $\longleftarrow$ ν ; NOEUD-*-ÉCRITURE($\mathcal{R}_p$, AR)

τ(LIEN-*-LECTURE($\mathcal{R}_{qp}$))
$AR \longleftarrow$NOEUD-*-LECTURE($\mathcal{R}_{q)}$;
i $\longleftarrow$ 0 ; **while** (AR[i].nom $\neq id_q$) **do** i++ ; **done**
return $AR[i].valeur$

Lisa Higham et moi avons conçu un compilateur auto-stabilisant du modèle NOEUD-ATOMIQUE dans le modèle NOEUD-RÉGULIER. La procédure τ(NOEUD-ATOMIQUE-ÉCRITURE($\mathcal{R}$, ν)) est sans attente, elle nécessite exactement $2 + \Delta$ opérations de communication.
La fonction τ(NOEUD-ATOMIQUE-LECTURE($\mathcal{R}$)) n'est pas sans attente ; par contre, elle est à terme sans goulot d'étranglement.
Une fois le système stabilisé, un processus p obtiendra le contenu du registre du processus q en réalisant une seule lecture valide d'un registre du modèle cibre, dans le cas où q n'exécute pas simultanément la procédure τ(NOEUD-ATOMIQUE-ÉCRITURE($\mathcal{R}$, ν)).

Notre compilateur du modèle NOEUD-ATOMIQUE dans le modèle NOEUD-RÉGULIER préserve la propriété de « quiétude ». Si l'algorithme initial est **silencieux** [DGS96] alors l'algorithme compilé l'est aussi. Un algorithme est dit silencieux, si aucune écriture dans les registres n'est effectuée une fois le système stabilisé. Supposons que l'algorithme d'origine soit « silencieux ». Une fois le système stabilisé, la procédure τ(NOEUD-ATOMIQUE-ÉCRITURE($\mathcal{R}$, ν)) n'est plus invoquée, ainsi les invocations de τ(NOEUD-ATOMIQUE-LECTURE($\mathcal{R}$)) sont réalisées sans que des LIEN-ATOMIQUE-ÉCRITURE opérations soient effectuées. Donc, l'algorithme compilé est silencieux. Ce compilateur est présenté en détail dans le rapport technique **[HJ06b]**.

2.1.6 Compilateurs impossible à construire

Un **graphe complet** est un graphe où tous les processus ont des liens de communication avec tous les autres processus. Sur un tel graphe de communication, un processus peut transmettre des informations (par exemple, sur la valeur retournée) à tous les autres processus pouvant lire ce registre. Donc, les lecteurs peuvent se coordonner relativement facilement. Concevoir un

compilateur pour les réseaux complets est similaire à concevoir des compilateurs dans un système à mémoire partagée. Les compilateurs sans-attente ont été largement étudié dans le cadre de système à mémoire partagée [Lam86, Abr95, HV95, LTV96, AW98]. Des constructions sans-attente d'un registre atomique multi-écrivains et multi-lecteurs avec uniquement des registres sûrs mono-écrivain et mono-lecteur ont été proposées. La conception de compilateurs sans-attente et auto-stabilisant a été étudiée par Jaap-Henk Hoepman, Marina Papatriantafilou et Phillipas Tsigas dans [HPT02] pour les systèmes à mémoire partagée. Des versions auto-stabilisantes de compilateurs sans-attente bien-connus sont proposées, ; ainsi ils construisent une chaîne de compilateurs auto-stabilisant et sans-attente qui permettent d'implémenter un registre atomique de taille bornée de type multi-écrivains et multi-lecteurs par des registres sûrs de type mono-écrivain et dual-lecteurs.

Une communication directe entre tous les processus (lecteurs) n'existent pas dans le cas de graphe non complet. Un processus peut ne pas avoir de lien de communication avec tous les voisins de ces voisins. Un processus ne peut donc pas connaître la dernière valeur « lue » sur un registre appartenant au processus p par un autre voisin de p. Synchroniser les valeurs « lues » est plus complexe dans les réseaux basés sur un graphe incomplet que dans les réseaux basés sur un graphe complet. Par exemple, il n'y a pas de compilateur sans-attente qui transforme un algorithme conçu dans le modèle NOEUD-ATOMIQUE par un algorithme s'exécutant dans le modèle NOEUD-RÉGULIER dans le cas de réseaux basés sur un graphe non complet. Cette preuve établit par Lisa Higham et moi fut présentée à IPDPS en 2006 **[HJ06a]**.

Supposons qu'il existe un compilateur sans-attente qui transforme un algorithme conçu dans le modèle NOEUD-ATOMIQUE par un algorithme s'exécutant dans le modèle NOEUD-RÉGULIER dans le cas de réseaux non complet. Nous nommons ce compilateur : ss_att_comp. Nous pourrions construire un compilateur sans-attente du modèle NOEUD-ATOMIQUE au modèle NOEUD-RÉGULIER via la composition séquentielle des trois compilateurs sans-attentes suivant : (1) le compilateur ss_att_comp, (2) l'implémentation sans-attente de Lamport d'un régulier registre mono-écrivain et mono-lecteur par des registres sûrs mono-écrivain et mono-lecteur, (3) puis la régulière-transformation par identifiant. Comme le compilateur ss_att_comp n'existe pas, nous concluons qu'il n'existe pas de compilateur sans-attente qui transforme un algorithme conçu dans le modèle LIEN-ATOMIQUE par un algorithme s'exécutant dans le modèle LIEN-RÉGULIER dans le cas de réseaux non complets.

2.2 Action composite

Edsger Dijkstra a défini l'auto-stabilisant dans un papier de 2 pages [Dij74]. Ce papier est très succinct : trois algorithmes auto-stabilisants de circulation de jeton sur un anneau sont présentés, sans aucune preuve. Le plus célèbre des trois algorithmes est l'algorithme « K-states » ; cet algorithme fonctionne sur les anneaux unidirectionnels **semi-uniforme**. Un système réparti est dit **semi-uniforme** lorsque tous les processus exécutent le même code sauf un processus (souvent appelé la *racine* du réseau). Typique, cette racine pourrait être obtenue après l'élection d'un leader sur le réseau.

Dans ce papier, Edsger Dijkstra fait une hypothèse forte sur le mode de fonctionnement du système réparti : l'hypothèse de **ACTION_COMPOSITE** . Instantanément, un processus (1) obtient

l'état de ces voisins, (2) calcule son nouvel état, **et** (3) change d'état.

Le **code d'un algorithme dans le cadre d'actions composites** pour un processus consiste en un ensemble de variables et un ensemble fini de règles. Des exemples de tel protocole peuvent être trouvés dans la section 3.2.2.
Une règle est constituée de deux éléments : *Condition* $\longrightarrow$ *Mise à jour*.
La « Condition » est un prédicat sur l'état du processus et de ses voisins. La « Mise à jour » est une série d'instructions où le processus change uniquement les valeurs de ses propres variables. Si « Condition » est vérifiée le processus est dit **activable** (en anglais enabled). Un processus réalise de manière instantanée (1) l'évaluation des Conditions et (2) la Mise à jour associée à la règle activable (si une règle est activable).

Shlomi Dolev, Amos Israeli et Shlomo Moran [DIM93] ont été les premiers à concevoir des algorithmes auto-stabilisant pour un modèle de calcul plus réaliste (les actions sont simples). Dans le cas d'action simple, instantanément, (1) un processus écrit dans le ou dans un de ses registres de communication, (2) lit le contenu du ou d'un registre de communication de l'un de ses voisins **ou** (3) effectue des calculs. Dans le cas d'actions simples, les registres sont atomiques, car les lectures ou écritures dans les registres sont instantanées. Ils existent donc deux modèles de communication par registre où les opérations sont effectuées instantanément : le modèle NOEUD-ATOMIQUE et le modèle LIEN-ATOMIQUE.

Très vite, des compilateurs transformant un algorithme conçu dans le cadre d'actions composites en un algorithme fonctionnant avec des actions simples ont été proposés (voir la section 5.1.3).

2.2.1 Exécutions séquentielles versus exécutions concurrentes

Il est notoirement plus facile de prouver la convergence d'un algorithme auto-stabilisant dans le cas d'exécutions séquentielles. Une exécution est dite **séquentielle**, si elle est composée de pas de calcul simple (pas où un seul processus effectue une action composite à un instant donné). Par contre, supposez que toutes les exécutions sont séquentielles dans un système réparti où les processus sont autonomes, est une hypothèse peu réaliste. C'est pourquoi, il est préférable d'avoir un algorithme supportant des **pas de calcul concurrents**. Pas de calcul où plusieurs processus effectuent simultanément une action composite.

Une réponse à cette contradiction est la construction de compilateurs sémantiques. Un **compilateur sémantique** transforme un algorithme dans le cadre d'actions composites en un autre algorithme supportant une famille étendue d'ordonnancement. L'ordonnancement des processus est une abstraction du non-déterminisme de l'environnement. Les vitesses des processus, les délais pour recevoir ou émettre un message varient de manière substantielle d'une exécution à une autre. Pour représenter cet environnement fluctuant, nous utilisons la notion d'ordonnanceur. **L'ordonnanceur** (aussi appelé **démon**) choisit, de manière non déterministe, les processus activables qui effectuent leur Mise à jour et ceci à chaque pas de calcul. Ce choix (cet ordonnancement) détermine l'exécution qui sera réalisée dans le cadre d'algorithme déterministe. Dans le cadre d'algorithme probabiliste, ce choix détermine une stratégie (un ensemble d'exécutions formant un espace probabiliste). Citons quelques ordonnanceurs :

- [Dij74], un ordonnanceur **centralisé** choisit à chaque pas de calcul un et un seul processus activable (si un tel processus existe) ;

- Un ordonnanceur **synchronisant** choisit systématiquement à chaque pas de calcul tous les processus activables ;
- **[BGJ99]**, un ordonnanceur est k**-borné** garantit que tant qu'un processus est continûment activable, un autre processus exécute au plus k Mises à jour ;
- un ordonnanceur **libre** (en anglais, unfair distributed scheduler) choisit librement à chaque pas de calcul un ou plusieurs processus activables (s'ils existent).

Les propriétés des exécutions dépendent de l'ordonnancement, mais aussi de l'algorithme exécuté. Citons quelques types d'exécutions :

- une exécution est **équitable** si chaque processus exécute infiniment souvent une règle. ;
- **[BGJ07]**, une exécution est k**-équitable** si chaque processus exécute une règle tous les k pas de calcul. Une k-exécution équitable est infinie, alors qu'un ordonnanceur k-borné peut produire des exécutions finies (en fonction de l'algorithme exécuté) ;
- **[FJ01]**, une exécution **alternative** est une exécution séquentielle telle qu'entre deux Mises à jour par un processus ces voisins exécutent chacun une seule Mise à jour.

De nombreux algorithmes auto-stabilisant d'exclusion mutuelle locale ont été proposés dans le cadre d'actions composites [GH97, GH99, HC01, KY02a, HHH03, Kar05, HG05, HP05, LL05, LW06, LYT06]. Ces algorithmes permettent de transformer un algorithme Agl conçu pour un ordonnancement centralisé en un algorithme supportant un ordonnancement libre. Le code de Agl est la section critique dont l'accès est gére par l'algorithme auto-stabilisant d'exclusion mutuelle locale. Un processus sort de la section critique après avoir réalisé une action composite de Agl ou constaté qu'aucune action de Agl n'est activable. Toutes les exécutions ont un suffixe où deux processus voisins ne sont pas dans la section critique. C'est-à-dire qu'elles ont un suffixe où deux voisins n'exécutent pas simultanément une action composite de Agl. Un pas de calcul où seuls des processus non-voisins exécutent simultanément une action composite est sérialisable. Un pas de calcul **sérialisable** est un pas de calcul équivalent à une suite de pas de calcul séquentiel. Ainsi, toutes les exécutions de l'algorithme obtenu ont un suffixe séquentiel.

La cross-over composition présentée dans la section suivante est un outil que nous avons utilisé pour transformer des algorithmes qui convergent sous des ordonnancements k-bornés en algorithmes qui convergent sous l'ordonnanceur libre.

2.2.1.1 Cross-over composition

Joffroy Beauquier, Maria Gradinariu et moi avons présenté la cross-over composition en 2001 au « Symposium on Self-stabilizing Systems » **[BGJ01]**. Les preuves des propriétés de la cross-over sont aussi présentés dans **[BGJ07]**. La cross-over composition de Agl avec Agl', notée Agl $\diamond$ Agl', permet de corréler l'exécution des règles de Agl avec l'exécution des règles de Agl' de telle sorte qu'un processus effectue une règle de Agl uniquement lorsqu'il effectue une règle de Agl'.

Un exemple de code produit par une cross-over composition est donné dans Algorithme 6. Il s'agit de la cross-over composition de l'algorithme CTC avec l'algorithme DTC. Le code l'algorithme CTC est présenté dans Algorithme 5 ; le code l'algorithme DTC est présenté dans Algorithme 3.

Les propriétés de l'algorithme Agl $\diamond$ Agl' est fonction des propriétés de Agl et de Agl'. ces propriétés ont été présentées et prouvées dans le cas où (1) Agl et Agl' sont des algorithmes déterministes ; (2) un des deux algorithmes est probabilistes, ou (3) les deux algorithmes sont probabilistes.

1. Supposons que chaque exécution de Agl' atteint une configuration vérifiant le prédicat $\mathcal{L}$'. Chaque exécution de Agl $\diamond$ Agl' atteint une configuration vérifiant le prédicat $\mathcal{L}$'.
2. Supposons que chaque exécution de Agl atteint une configuration vérifiant le prédicat $\mathcal{L}$. Chaque exécution de Agl $\diamond$ Agl' atteint une configuration vérifiant le prédicat $\mathcal{L}$ si toutes les exécutions de Agl' sont équitables.
3. Toutes les exécutions de Agl $\diamond$ Agl' sont équitables si les exécutions de Agl' sont équitables.
4. Les exécutions de Agl $\diamond$ Agl' sont k-équitables si les exécutions de Agl' sont k-équitables.
5. Un processus ayant une règle de Agl activable attend au plus k pas de calcul avant d'exécuter cette règle, si toutes les exécutions de Agl' sont k-équitables.
6. Supposons que l'algorithme Agl est auto-stabilisant dans le cas d'un ordonnanceur k-borné. L'algorithme Agl $\diamond$ Agl' est un algorithme auto-stabilisant pour l'ordonnanceur libre, si toutes les exécutions de Agl' sont k-équitables.

2.2.1.2 Circulation de jetonS

Algorithme 3 *DTC* - Circulation déterministe de jetons sur un anneau

paramètres :
m_N : plus petit non diviseur de N.
m_N est égale à 2 pour les anneaux dont la taille est impaire.
l : le voisin de gauche de p

variables de p :
dt est un entier positif inférieur à m_N.

prédicat de p :
jeton_déterministe(p) $\equiv dt_p \neq (dt_l + 1) \mod m_N$

macro de p :
passer_jeton_déterministe(p) : $dt_p := (dt_l + 1) \mod m_N$

code de p :
$\mathcal{A}_1$: jeton_déterministe(p) $\longrightarrow$ passer_jeton_déterministe(p)

L'algorithme *DTC* fut présenté à PODC 1999 par Joffroy Beauquier, Maria Gradinariu et moi **[BGJ99]**, il est aussi présenté dans un papier accepté à « Distributed Computing ». **[BGJ07]**. Cet algorithme très simple a des propriétés très intéressantes :

- il y a toujours au moins un jeton dans l'anneau ;
- le nombre de jetons ne peut pas croître, quelle que soit l'exécution effectuée ;
- toutes les exécutions ont un suffixe où le nombre de jetons reste constant ;

- un processus attend au plus $N^2/2$ pas de calcul avant de passer le jeton à son voisin à sa droite. Donc les exécutions sont N^2-équitables.

L'algorithme $\mathcal{A}\text{gl} \diamond DTC$ est obtenu en corrélant l'exécution des règles de $\mathcal{A}\text{gl}$ avec l'exécution des règles de DTC de telle sorte qu'un processus effectue une règle de $\mathcal{A}\text{gl}$ uniquement lorsqu'il effectue une règle de DTC. Les propriétés de l'algorithme $\mathcal{A}\text{gl} \diamond DTC$ sont déterminées par les propriétés de DTC et celles de la cross-over composition :

- $\mathcal{A}\text{gl} \diamond DTC$ est un algorithme dont toutes les exécutions sont équitables ;
- les exécutions de $\mathcal{A}\text{gl} \diamond DTC$ sont N^2-équitables, un processus attend au plus N^2 pas de calcul avant d'exécuter un règle ;
- un processus ayant une règle de $\mathcal{A}\text{gl}$ activable attend au plus N^2 pas de calcul avant d'exécuter cette règle ;
- si l'algorithme $\mathcal{A}\text{gl}$ est auto-stabilisant sur un anneau unidirectionnel sous l'ordonnanceur N^2-borné alors l'algorithme $\mathcal{A}\text{gl} \diamond DTC$ est un algorithme auto-stabilisant sous l'ordonnanceur libre.

Chapitre 3

De l'espace mémoire nécessaire à des algorithmes auto-stabilisants

Edsger Dijkstra [Dij74] a introduit la notion d'auto-stabilisation via la présentation de protocoles de circulation d'un jeton sur un anneau semi-uniforme anonyme.

Ce premier papier a été le point de départ d'un nombre considérable de papiers sur la circulation de jetons ou/et l'élection auto-stabilisante d'un leader sur les anneaux anonymes dans le cadre d'actions composites. L'objectif de ce chapitre, est de présenter mes contributions à ce champ de recherche, en présentant leurs articulations avec les résultats obtenus dans ce domaine par la communauté.

3.1 Définition

La spécification du **problème d'élection d'un leader** est le prédicat suivant sur les exécutions : *un et un seul processus est un leader ; c'est toujours le même processus qui est leader au cours de l'exécution*. Un leader est un processus dont la configuration locale vérifie le prédicat *Leader*. La définition du prédicat *Leader* dépend de l'algorithme d'élection de leader.

La spécification du **problème de la circulation d'un jeton** est le prédicat suivant sur les exécutions : *un et un seul jeton circule dans le réseau ; tous les processus ont infiniment souvent le jeton au cours des exécutions*. Un processus a le jeton si sa configuration locale vérifie le prédicat *jeton*. La définition du prédicat *jeton* dépend de l'algorithme de circulation d'un jeton.

3.2 Algorithmes Déterministes

Nous avons circonscrit notre étude aux anneaux anonymes. Nous notons N la taille de l'anneau. L'élection et la circulation d'un jeton sur un anneau anonyme, à cause de la structure symétrique d'un anneau, synthétise presque toutes les difficultés rencontrées lors de la conception d'algorithmes auto-stabilisants.

Dans un réseau anonyme, les processus sont indiscernables les uns des autres : ils exécutent le même code, ils n'ont pas d'identifiant, la topologie du réseau à distance d est la même pour tous

les processus quelle que soit la valeur de d. Ils ont exactement le type de voisinage à distance 1, 2, ... A partir d'une configuration initiale où tous les processus sont exactement dans le même état il est quasi-impossible de rompre la symétrie de la situation. C'est pourquoi, il y a quelques résultats d'impossibilités.

3.2.1 Résultats d'impossibilité

Dana Angluin [Ang80] a établi qu'il n'existe pas de solution déterministe pour un réseau anonyme si les processus sont synchrones dans le cadre d'actions composites (en autres termes, dans le cas où l'ordonnanceur est synchronisant). La figure 3.1 illustre le résultat d'impossibilité. Initialement (configuration A) tous les processus sont dans le même **configuration locale**, la configuration est dite **symétrique**. Deux processus ont la même configuration locale s'ils ont le même état et si leur voisins sont aussi dans le même état. A partir d'une configuration symétrique, si un processus peut effectuer une action a alors tous les processus peuvent effectuer la même action a. Il s'agit d'une action déterministe, donc tous processus après avoir exécuté cette action obtiennent le même état. Bilan, après un pas de calcul synchrone où tous les processus activables ont effectué une action, la configuration obtenue est symétrique. Clairement, il n'y a pas moyen de briser la symétrie de la configuration initiale lorsque les processus sont synchrones. Un algorithme auto-stabilisant devant fonctionner quelle que soit la configuration initiale, en particulier, il doit converger à partir d'une configuration symétrique vers une configuration asymétrique où une seule configuration locale vérifie le prédicat *leader* ou *jeton*. Donc, il n'existe pas d'algorithme auto-stabilisant d'élection de leader et de circulation d'un jeton sur un anneau anonyme fonctionnant lorsque les processus sont synchrones.

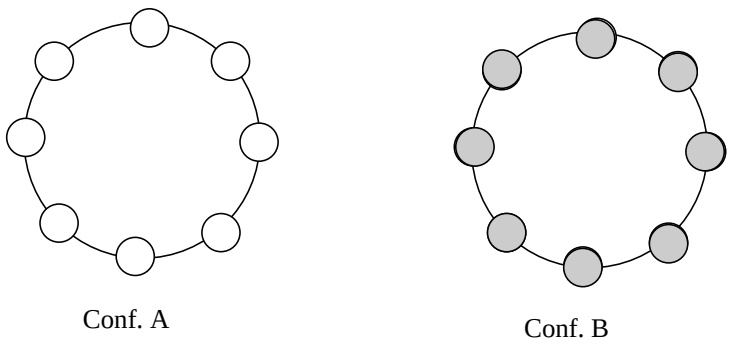

FIG. 3.1 – exécution synchrone à partir d'une configuration symétrique

C'est pourquoi, James E. Burns et Jan Pachl, [BP89], ont étudié la conception d'algorithmes opérationnels uniquement si les exécutions sont séquentielles et les actions composites. Ils ont établi un deuxième résultat d'impossibilité : il n'existe pas d'algorithme auto-stabilisant d'élection de leader sur un anneau anonyme dont la taille n'est pas un nombre premier. Nous illustrons ce résultat par une figure 3.2. La configuration initiale (configuration C) est **quasi-symétrique**. Dans une configuration quasi-symétrique l'ensemble des processus est partitionné de tel sorte que (1) tous les processus d'une classe ont la même configuration locale (2) deux processus

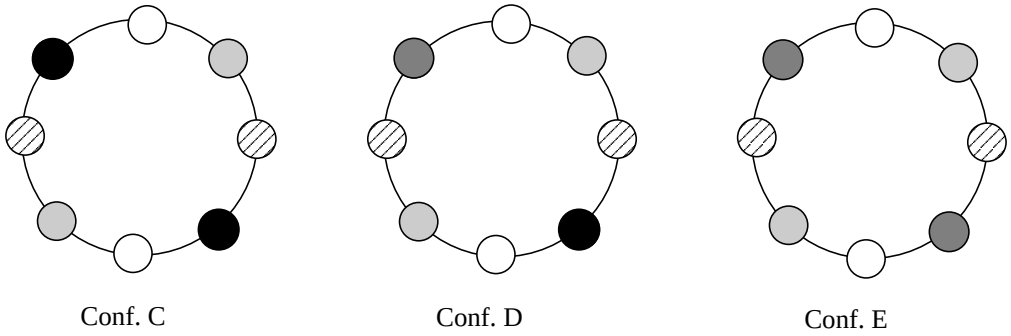

FIG. 3.2 – exécution séquentielle à partir d'une configuration quasi-symétrique

d'une même classe ne sont pas voisins et (3) chaque classe contient plusieurs processus. Si un processus peut effectuer une action donnée a, tous les autres processus de sa classe peuvent aussi effectuer la même action a. De plus, après qu'un processus est effectué l'action a, les autres processus de sa classe peuvent encore effectuer l'action a, car leur configuration locale n'a pas changé. Une action par un processus peut être suivie immédiatement par une séquence de pas de calcul séquentiels où chaque processus de sa classe vont effectuer la même action les uns après les autres. Une telle séquence est présentée dans la figure 3.2. Après cette séquence, une autre configuration quasi-symétrique est obtenue (configuration E). Il est donc possible de construire une exécution séquentielle infinie contenant périodiquement des configurations quasi-symétriques si la configuration initiale est quasi-symétrique. Dans une configuration quasi-symétrique, il y a zéro ou plusieurs processus ayant une configuration locale vérifiant le prédicat *leader* ou *jeton*. Il faudrait un et un seul processus dont la configuration locale vérifie ce prédicat ; nous avons donc une impossibilité.

Une configuration quasi-symétrique ne peut pas être construite sur un anneau dont la taille est un nombre premier. Nous nous intéressons donc à la conception d'algorithmes déterministes d'élection de leader et de circulation d'un jeton sur un anneau anonyme uniquement dans le cas où sa taille est un nombre premier et où les exécutions sont séquentielles. Dans la section 3.2.2, nous considérons les anneaux unidirectionnels, dans la section 3.2.3, nous étudions les anneaux bidirectionnels.

3.2.2 Algorithmes sur les anneaux unidirectionnels

3.2.2.1 Election d'un leader

Algorithme de James E. Burns et Jan Pachl. James E. Burns et Jan Pachl, [BP89], ont proposé le premier algorithme de circulation d'un jeton sur un anneau unidirectionnel anonyme. Un processus décide de son état uniquement en fonction de l'état de son voisin de gauche et de son propre état. Cet algorithme peut être facilement transformé en un algorithme de leader élection. Algorithme 4 est une présentation originale du code de l'algorithme de Burns et Pachl. L'idée de Burns et Pachl est de numéroter successivement les processus de 0 à $N-1$, le processus d'indice

0 est l'unique leader. Un processus calcule son indice en incrémentant le numéro de son voisin de gauche (règle $\mathcal{R}_1$). Pour stabiliser la numérotation, un leader (un processus ayant l'indice 0) tente de garder son leadership aussi longtemps que possible (c'est-à-dire de conserver l'indice 0). Ainsi, des segments de processus ayant une numérotation séquentielle sont créés et sont quasi-stables. La longueur d'un segment est enregistrée dans la variable *puissance* de tous les processus du segment suivant (règles $\mathcal{R}_3$ et $\mathcal{R}_4$). La taille de l'anneau étant un nombre premier, les segments ne sont pas tous de même longueur. Si l'anneau a plusieurs segments alors la situation anormale suivante existe : « Un segment $S1$ de longueur $l1$ est suivi par un segment $S2$ de plus grande longueur $L2$ ». Le leader à la tête du segment suivant $S2$ constate cette anormalité (c'est-à-dire que la garde de la règle $\mathcal{R}_2$ est vérifiée par le leader du segment). Dans cette situation, il abandonne son leadership et joint le segment $S2$ (règle $\mathcal{R}_2$). Ce mécanisme assure que l'algorithme converge vers une situation où il y a un seul segment de longueur N.

Algorithme 4 Election de leader d'après l'algorithme de Burns et Pachl

paramètres :
N : est la taille de l'anneau
l : le voisin de gauche de p

variables de p :
indice et *puissance* sont des entiers positifs inférieurs à N.

fonction de p :
$funct(x, y) = x$;

prédicat :
$Ld_p \equiv indice_p == 0$

code de p :
$\mathcal{R}_1$: $indice_p \neq (indice_l + 1) \mod N$ et $indice_p \neq 0 \longrightarrow$
$indice_p := (indice_l + 1) \mod N$; $puissance_p := puissance_l$;
$\mathcal{R}_2$: $indice_p \neq (indice_l + 1) \mod N$ et $indice_p == 0$ et
$(indice_p == indice_l$ ou $puissance_l < indice_l) \longrightarrow$
$indice_p := (indice_l + 1) \mod N$; $puissance_p := puissance_l$;
$\mathcal{R}_3$: $indice_p == (indice_l + 1) \mod N$ et $puissance_p \neq puissance_l$ et $indice_p \neq 0 \longrightarrow$
$puissance_p := puissance_l$;
$\mathcal{R}_4$: $indice_p == 0$ et $indice_p \neq (indice_l + 1) \mod N$ et $puissance_l \geq indice_l$ et
$puissance_p \neq fonct(indice_l, puissance_l) \longrightarrow$
$puissance_p := fonct(indice_l, puissance_l)$;

L'algorithme de Burns et Pachl nécessite $O(N^2)$ états par processus (ou $2\lg(N)$ bits d'espace mémoire sur chaque processus). Chengdian Lin et Janos Simon dans [LS92] ont analysé en détail l'algorithme de l'algorithme de Burns et Pachl. Ils sont prouvés que le temps de convergence de l'algorithme est de $\Theta(N^3)$ dans le cas le pire. De plus ils ont proposé un variante de l'algorithme de Burns et Pachl nécessitant $O(N\sqrt{\frac{N}{\ln(N)\ln(\ln(N))}})$ états par processus. Pour cela, ils ont remplacé la fonction $func$ par une fonction de hachage H qui a la propriété suivante : *il n'existe pas un*

ensemble d'entiers $a_1, ..., a_k$ *ayant tous la même image par* H *et tel que* $\sum_{j=1}^{k} a_j = N$. Cette propriété permet de garantir que la multi-segmentation sera détectée et corrigée par la règle $\mathcal{R}_2$. Ainsi, l'algorithme converge vers une configuration où il y a un seul segment.

Algorithme de Faith Fich et Colette Johnen Faith Fich et moi avons conçu un algorithme d'élection de leader nécessitant $O(5N)$ états par processus. Cet algorithme fut présenté à DISC en 2001 **[FJ01]**. L'algorithme utilise le même mécanisme que l'algorithme de Burns et Pachl pour sélectionner un unique processus : un leader détecte la présence d'une pluralité de leaders en comparant les longueurs des segments adjacents (un segment étant la série de processus séparant deux leaders). L'originalité est qu'un processus ordinaire stocke en mémoire, à un instant donné, soit la valeur de la variable *puissance*, soit son indice. Par contre, un leader stocke en permanence l'indice de son voisin de gauche (qui est la valeur de sa « puissance ») Chaque processus a donc une seule variable *var* pouvant contenir les valeurs entières de 0 à N. Par ailleurs, chaque processus a une variable booléenne (Ld) indiquant s'il est un leader ou pas, et une autre variable (*status*) précisant la signification de la valeur de *var*. Clairement ($Ld \equiv ld_p$) un leader est un processus dont la variable Ld a la valeur « vrai ». Dans l'algorithme de Burns et Pachl, un leader décide d'abandonner son leadership en comparant les valeurs des deux variables de son voisin de gauche ($puissance_l < num_l$). Dans notre algorithme, le leader compare la valeur de *var* de son voisin de gauche avec sa valeur de *var* dans le cas où son voisin stocke la valeur de *puissance* : $(var_l < var_p) \wedge (status_l == puissance) \wedge (Ld_p ==$ vrai$)$.
Cette comparaison ne donnera le résultat escompté que si les processus relayent régulièrement la puissance de leur « leader » (la tête de leur segment) le long du segment et ensuite calcule la longueur de leur segment, en déterminant, les uns après les autres, leur indice. Il est donc nécessaire d'avoir un mécanisme très semblable à la circulation de messages sur l'anneau. Un leader indique dans un « message » la tâche à effectuer sur son segment (calcul des indices ou transmission de sa puissance au prochain leader). Les processus d'un segment à la réception d'un « message » exécutent la tâche à faire et transmettent le « message » au processus suivant dans l'anneau et ceci avant qu'ils reçoivent un autre « message ». Si l'exécution est alternative, ce mécanisme de prise en compte de manière fiable des messages/ordre est garanti. C'est pourquoi, notre algorithme, noté $Fich\_Johnen$ suppose que l'ordonnanceur produit uniquement des exécutions alternatives.
Utilisant la cross-over composition (présentée dans la section 2.2.1) et l'algorithme DTC (présenté dans la section 2.2.1), nous obtenons l'algorithme $Fich\_Johnen \diamond DTC$. Une fois le nombre de jetons fixe, les exécutions de DTC sont alternatives si l'ordonnanceur est centralisé. Entre deux actions d'un processus, ces deux voisins exécutent une seule action. Donc, toutes les exécutions de DTC ont un suffixe alternatif. Ainsi, l'algorithme $Fich\_Johnen \diamond DTC$ est un algorithme auto-stabilisant sous tous les ordonnancements centralisés. Notons que DTC est un algorithme **pseudo-stabilisant** [BGM93] : toutes les exécutions de DTC ont un suffixe vérifiant le prédicat d'exactitude (en anglais, correctness) suivant « *les exécutions sont alternatives* ».
Ainsi, l'algorithme $Fich\_Johnen \diamond DTC$ est un algorithme d'élection de leader sur anneau unidirectionnel anonyme sous un ordonnancement centralisé. Cet algorithme nécessite $10N$ états par processus.

J'ai réalisé une « applet » java simulant l'algorithme $Fich\_Johnen \diamond DTC$. Cette applet est accessible via l'URL « http ://www.lri.fr/∼colette/Java/F/deterministic.html ». A chaque accès,

un anneau de taille première est construit et une configuration initiale est proposée. La taille de l'anneau est déterminé aléatoirement parmi 11 nombres premiers tous inférieurs à 50. Cette applet permet de démarrer une nouvelle simulation à partir d'une configuration initiale calculer aléatoirement, à tout instant.

Bornes sur l'espace mémoire Joffroy Beauquier, Maria Gradinariu et moi avons déterminé une borne inférieure à l'espace mémoire nécessaire sur chaque processus pour l'élection d'un leader sur un anneau anonyme et unidirectionnel : $\lg(N)$. Ce résultat fut présentée à « l'ACM Symposium on Principles of Distributed Computing » en 1999 **[BGJ99]**. Nous avons fait une preuve par l'absurde en supposant que le nombre d'états par processeur est inférieur à la taille de l'anneau, par exemple $x = N-1$ états. Nous avons construit une exécution séquentielle infinie où tous les processus ont à la même fréquence les mêmes configurations locales. Il est impossible qu'un seul processus ait, en permanence, une configuration locale vérifiant un prédicat. Donc, il n'est pas possible de construire un algorithme d'élection de leader.

Ainsi, l'espace mémoire nécessaire par processus à un algorithme d'élection de leader sur un anneau unidirectionnel anonyme de taille première est asymptotiquement équivalent à $\lg(N)$ ($\Theta(\lg(N))$). Faith Fich et moi avons proposé une borne supérieure à l'espace mémoire nécessaire sur chaque processus pour l'élection d'un leader sur un anneau anonyme et unidirectionnel : $\lg(10N)$ en présentant un algorithme nécessitant $10N$ états par processus.

3.2.2.2 Circulation d'un jeton

Dans [GH96], Mohamed G. Gouda et Furman Haddix présente un algorithme de circulation d'un jeton sur un anneau semi-uniforme nécessitant 3 bits d'espace mémoire par processus. Un algorithme d'élection de leader sur un anneau anonyme permet de transformer l'anneau en un anneau semi-uniforme : tous les processus sauf le leader exécute le même code, le leader exécutant le code réservé à la racine.

En composant équitable l'algorithme d'élection de leader présenté dans **[FJ01]** et l'algorithme de circulation d'un jeton présenté dans [GH96], un algorithme de circulation d'un jeton dans un anneau anonyme unidirectionnel est obtenu. Cet algorithme nécessite $O(80N)$ états par processus (ou $7 + \lg(N)$ bits par processus).

Nous pouvons établir une borne inférieure à l'espace mémoire nécessaire sur chaque processus pour la circulation d'un jeton sur un anneau anonyme et unidirectionnel de la même manière que nous avons prouvé la borne pour l'élection de leader dans **[BGJ99]**. La preuve est par l'absurde : nous supposons que l'espace mémoire par processeur est inférieur à $\lg((N-1)/2)$. Nous avons construit une exécution séquentielle infinie où (1) tous les processus prennent à la même fréquence les mêmes états, (2) des configurations quasi-symétriques sont atteintes, à intervalles réguliers. . Comme nous l'avons déjà signalé dans la section 3.2.1, dans une configuration quasi-symétrique, il y a zéro ou plusieurs processus ayant une configuration locale vérifiant le prédicat *jeton*. Or, il faudrait qu'à tout instant, un et un seul processus ait une configuration locale vérifiant ce prédicat. Nous avons donc une impossibilité.

Ainsi, l'espace mémoire par processus nécessaire à un algorithme de circulation d'un jeton sur un anneau unidirectionnel anonyme de taille première est asymptotiquement équivalent à $\lg(N)$ ($\Theta(\lg(N))$).

3.2.3 Algorithmes sur les anneaux bidirectionnels

Dans cette section, nous nous intéressons aux anneaux anonymes et bidirectionnels. Les résultats d'impossibilité, présentés dans la section 3.2.1, sont valides pour les anneaux anonymes bidirectionnels. Donc, nous intéressons aux anneaux dont la taille est un nombre premier dans le cas d'exécutions séquentielles.
Gene Itklis, Chenddian et Janos Simon ont construit un algorithme d'élection de leader sur un anneau orienté [ILS95]. Cet algorithme nécessite 13 bits d'espace mémoire par processus. L'idée de l'algorithme est une variante de l'idée de de l'algorithme de Burns et Pachl. Chaque leader compare dd (sa distance avec le leader sur sa droite) avec dg (sa distance avec le leader sur sa gauche). Si $ds \neq dp$ alors un leader démissionne : il a plusieurs leaders sur l'anneau. Comme l'anneau a pour taille un nombre premier, une telle situation existe s'il a plusieurs leaders. La pluralité des leaders est ainsi éliminée de manière fiable. Cette comparaison est effectuée sans stocker la valeur de dd ou de dg. Un leader positionne une « marque » sur le segment à sa droite et une autre sur le segment à sa gauche. Ces « marques » doivent s'éloigner à la même vitesse du leader. Si une des marques atteint en premier un leader, ce dernier démissionne. Pour que les « marques » bouge à la même vitesse, le leader envoie sur sa droite un jeton qui pousse la marque de droite encore plus à droite ; puis ce jeton change de direction pour revenir vers le leader. Le leader envoie ce jeton sur sa gauche pour pousser la marque de gauche encore plus à gauche ; puis le jeton change de direction pour revenir vers le leader ; ainsi de suite.

Cet algorithme suppose que l'anneau soit orienté. Un processus a deux voisins, un qu'il considère à sa gauche et l'autre qu'il considère à sa droite. Un anneau est **orienté** si tous les processus sont d'accord sur le sens des directions « droite » et « gauche ». Dans le cas d'un anneau orienté, si un processus p a pour voisin de gauche q, alors le processus q considère que p est à sa droite. Jaap-Henk Hoepman [Hoe98] a proposé un algorithme déterministe d'orientation d'un anneau anonyme. Cet algorithme nécessite sur chaque processus 2 bits d'espace mémoire. Cet algorithme est opérationnel uniquement sur les anneaux de tailles impaires et dans le cas d'exécutions séquentielles.

En composant équitable l'algorithme d'orientation d'un anneau présenté dans [Hoe98] et l'algorithme d'élection de leader présenté dans [ILS95], un algorithme d'élection de leader dans un anneau anonyme bidirectionnel est obtenu. Cet algorithme nécessite 15 bits par processus.

En composant équitable l'algorithme d'orientation d'un anneau présenté dans [Hoe98], l'algorithme d'élection de leader présenté dans [ILS95] et l'algorithme de circulation d'un jeton présenté dans [GH96], un algorithme de circulation d'un jeton dans un anneau anonyme bidirectionnel est obtenu. Cet algorithme nécessite 18 bits d'espace mémoire par processus.

3.3 Algorithmes Probabilistes

Les algorithmes déterministes sont intéressants d'un point de vue théorique. Ils nous permettent de classifier les problèmes selon l'espace mémoire nécessaire à leur concrétisation, s'ils sont réalisables. Cependant, dans la majorité des cas, la conception d'un algorithme déterministe ne peut pas être envisagé. C'est pourquoi, nous sommes orientés vers la conception d'algorithmes probabilistes.

Dans les sections suivantes, nous nous intéressons à la conception d'algorithmes probabilistes d'élection de leader et de circulation d'un jeton sur un anneau anonyme unidirectionnel.

3.3.1 Circulation d'un jeton sur les anneaux unidirectionnels

Le premier algorithme probabiliste de circulation d'un jeton sur les anneaux anonymes unidirectionnels a été conçu par Ted Herman [Her90]. Il propose un algorithme fonctionnant sur les anneaux dont la taille est un nombre impair. Cet algorithme, très simple, nécessite 1 bit par processus. Ted Herman étudie le fonctionnement de cet algorithme uniquement sous l'ordonnanceur synchronisant.

3.3.1.1 Algorithme de Beauquier, Cordier et Delaët

L'algorithme CTC est la généralisation par Joffroy Beauquier, Alain Cordier et Sylvie Delaët [BCD95], de l'algorithme Ted Herman. Le code de l'algorithme CTC est présenté dans algorithme 5. L'algorithme CTC fonctionne sur des anneaux de toutes tailles. L'algorithme CTC est la version probabiliste de l'algorithme DTC, (le code est présenté dans algorithme 3). Donc l'algorithme CTC a les mêmes propriétés que DTC (propriétés présentées dans la section 2.2.1). L'anneau a toujours un jeton_probabiliste, le nombre de jeton_probabiliste, ne peut jamais croître quelle que soit l'exécution effectuée.

Algorithme 5 CTC - Circulation probabiliste d'un jeton

paramètres :
m_N : plus petit non diviseur de N.
m_N est égale à 2 pour les anneaux dont la taille est impaire.
l : le voisin de gauche de p

variable de p :
pt est un entier positif inférieur à 2.

prédicat de p :
jeton_probabiliste(p) $\equiv pt_p \neq (pt_l + 1) \mod m_N$

macro de p :
passer_jeton_probabiliste(p) : $pt_p := (pt_l + 1) \mod m_N$

code de p :
$\mathcal{R}_1$: jeton_probabiliste(p) $\longrightarrow$ **if** random(p(0)=1/2,p(1)=1/2) == 1 **then** passer_jeton_probabiliste(p)

Joffroy Beauquier, Alain Cordier et Sylvie Delaët ont prouvé que l'algorithme CTC stabilise lorsque l'ordonnanceur est équitable et à mémoire bornée [BCD95]. Un ordonnanceur est dit à **mémoire borné**, si le choix de l'ordonnanceur dépend uniquement des k précédents pas de calcul. L'ordonnanceur agit comme s'il avait qu'une mémoire à court terme, bornée aux k derniers pas de calcul - L'ordonnanceur ne semble pas se rappeler des pas de calcul antérieurs aux évènements immédiats -. Joffroy Beauquier, Maria Gradinariu et moi avons prouvé que l'algorithme CTC stabilise lorsque l'ordonnanceur est k-borné. Ce résultat fut présenté à « l'ACM Symposium on Principles of Distributed Computing » en 1999 **[BGJ99]** (la preuve complète est dans le papier **[BGJ07]** publié dans « Distributed Computing »). Un ordonnanceur équitable et à mémoire bornée est un ordonnanceur k-borné, mais la réciproque n'est pas vrai. Ce résultat fut présenté à « l'International Symposium on Stabilization, Safety, and Security of Distributed Systems » en 2006, par Joffroy Beauquier, Stéphane Messika et moi **[BJM06]**.

Joffroy Beauquier et moi avons analysé cet algorithme en détail dans un rapport technique du LRI **[BJ02]**. Cet algorithme est très intéressant ; il existe trois classes, non vides, d'ordonnancements équitables :

- ordonnancements sous lesquels l'algorithme CTC ne converge pas : la probabiliste que l'anneau contienne pour toujours plusieurs jetons est supérieure à 0 ;
- ordonnancements sous lesquels l'algorithme CTC converge, mais le temps de stabilisation est non borné ;
- ordonnancements équitables sous lesquels le temps de stabilisation est borné (par exemple, les ordonnanceurs k-bornés) ;

3.3.1.2 Algorithme de Beauquier, Gradinariu et Johnen

D'après les propriétés de l'algorithme DTC et de la cross-over composition (voir la section 2.2.1), l'algorithme $CTC \diamond DTC$ (nommé $SSCTC$) est un algorithme de circulation de jeton auto-stabilisant sous l'ordonnanceur libre (en anglais, unfair distributed scheduler). L'algorithme $SSCTC \diamond DTC$ fut présenté à « l'ACM Symposium on Principles of Distributed Computing » en 1999. Le code de l'algorithme $SSCTC$ est donné dans algorithme 6.

Le temps de stabilisation de l'algorithme $SSCTC$ a été étudié par Laurent Rosaz [Ros00]. Il a établi que le temps de convergence est en moyenne $\Theta(N^3)$ pour n'importe quel ordonnancement. Il a proposé une amélioration de l'algorithme $SSCTC$ ayant en moyenne un temps de convergence de $\Theta(N^2 \lg(N))$. Les jetons probabilistes sont étiquetés par une *vitesse*. Trois vitesses de progression existent : *lente*, *normale*, *rapide*. La probabilité de déplacement d'un jeton probabiliste varie en fonction de sa vitesse. Si le jeton est lent, il a une probabilité de $1/2$ d'avancer ; si le jeton a une vitesse normale, il a une probabilité de $3/4$ d'avancer ; et si le jeton est rapide, il a une probabilité de $7/8$ d'avancer. Les jetons déterministes transportent la vitesse du dernier jeton probabiliste rencontré. Si la vitesse transportée est identique à celle du jeton probabiliste atteint alors ce jeton probabiliste change de vitesse. Ainsi, deux jetons consécutifs et probabilistes n'ont pas la même vitesse de progression dans l'anneau. Par contre, deux jetons consécutifs et probabilistes ont presque les mêmes occasions d'avancer : le nombre de fois où un jeton essaye d'avancer (d'effectuer la règle $\mathcal{R}_2$) est identique au nombre de fois où un autre jeton essaye d'avancer à une constante près. Un jeton rapide avance, donc, en moyenne, plus rapidement que son prédécesseur.

Algorithme 6 *SSCTC* - Circulation auto-stabilisante d'un jeton

paramètres :

m_N : plus petit non diviseur de N.
l : le voisin de gauche de p

variables de p :

dt est un entier positif inférieur à m_N.
pt est un entier positif inférieur à m_N.

prédicats de p :

jeton_déterministe(p) $\equiv dt_p \neq (dt_l + 1) \mod m_N$
jeton_probabiliste(p) $\equiv pt_p \neq (pt_l + 1) \mod m_N$

macros de p :

passer_jeton_déterministe(p) : $dt_p := (dt_l + 1) \mod m_N$
passer_jeton_probabiliste(p) : $pt_p := (pt_l + 1) \mod m_N$

code de p :

$\mathcal{R}_1$: jeton_probabiliste(p) $\wedge$ jeton_déterministe(p) $\longrightarrow$
passer_jeton_déterministe(p);
if random(p(0)=1/2,p(1)=1/2) == 1 **then** passer_jeton_probabiliste(p)
$\mathcal{R}_2$: $\neg$jeton_probabiliste(p) $\wedge$ jeton_déterministe(p) $\longrightarrow$ passer_jeton_déterministe(p)

Il le rattrape donc plus rapidement que s'il avait la même vitesse de progression. C'est pourquoi, cet algorithme a un meilleur temps de convergence.

Bornes sur l'espace mémoire Joffroy Beauquier, Maria Gradinariu et moi avons établi une borne inférieure du nombre d'états par processus nécessaire à un algorithme probabiliste et auto-stabilisant de circulation d'un jeton sur un anneau unidirectionnel supportant l'ordonnanceur libre. Ce résultat fut présenté à « l'ACM Symposium on Principles of Distributed Computing » en 1999 **[BGJ99]**. Chaque processus doit avoir $(m_N - 2)/2$ états, sachant que m_N est le plus petit entier non diviseur de N, N étant la taille de l'anneau. La preuve est par l'absurde : nous supposons que l'espace mémoire par processeur est inférieur à $\lg(m_N - 2/2)$ bits. Nous établissons que soit, (1) il existe une configuration terminale soit il existe une exécution infinie, nommée ex, décrite plus loin. Une configuration est **terminale** si aucun processus n'est activable. Un algorithme auto-stabilisant de circulation d'un jeton ne saurait avoir une ou des configurations terminales. L'exécution ex a les propriétés suivantes : (1) l'ordonnanceur sélectionne toujours le même processus qui est toujours activable et (2) toutes les configurations atteintes pas cette exécution sont quasi-symétriques. L'exécution ex n'a pas de suffixe où un seul jeton circule (voir la preuve d'impossibilité de la section 3.2.1).

Ainsi, nous avons établi que l'espace mémoire nécessaire à un algorithme auto-stabilisant de circulation d'un jeton sur un anneau unidirectionnel supportant l'ordonnanceur libre est asymptotiquement équivalent à $\Theta(\lg(m_N))$ bits par processus. L'algorithme *SSCTC* établit une borne supérieure au nombre d'états nécessaire par processus m_N^2, donc l'espace mémoire nécessaire est $2.\ln(m_N)$ bits par processus.

3.3.1.3 Algorithme de Datta, Gradinariu et Tixeuil

Ajoy K. Datta, Maria Gradinariu et Sébastien Tixeuil ont mis en évidence une faille de l'algorithme *SSCTC*. Il n'y a pas de borne au temps d'attente par un processus du jeton. En moyenne, une fois le système stabilisé, le temps d'attente est inférieur à $2N^3$ pas de calcul, sous tous les ordonnancements. Mais quelle que soit la valeur de B, il existe toujours une probabilité non nulle qu'un processus attende plus de B pas de calcul avant d'obtenir le jeton. Dans [DGT00, DGT04], Ajoy K. Datta, Maria Gradinariu et Sébastien Tixeuil ont proposé un algorithme auto-stabilisant de circulation d'un jeton ayant une borne au temps de service. Si après $k+1$ essais, un processus n'a toujours pas réussi à transmettre le jeton probabiliste au voisin à sa droite alors ce processus transmettra le jeton de manière déterministe lors de sa prochaine action. Donc chaque processus a un compteur numérique borné à la valeur $k+1$. Dans un premier temps, ils ont construit l'algorithme *Alg1* fonctionnant sous les ordonnanceurs k-bornés. Un processus est assure d'obtenir le jeton en moins de $(k+1)N$ pas de calcul quelle que soit l'exécution, sous un ordonnanceur k-borné. Ensuite, ils proposent un algorithme *Alg2* fonctionnant sous tous les ordonnanceurs, L'algorithme *Alg2* est obtenu via la cross-over composition de *Alg1* avec *DTC*. L'algorithme *Alg2* fonctionne sous tous ordonnancements. Mais, l'espace mémoire nécessaire sur chaque processus est $2\lg(N.m_N)$ bits, son temps de service est au plus $(N+1)N^2$ pas de calcul sous l'ordonnanceur libre.

3.3.1.4 Algorithme de Kakugawa et Yamashita

Hirotsugu Kakugawa et Masafumi Yamashita dans [KY02b] ont proposé un algorithme ayant un temps de service borné à $2N$ sous l'ordonnanceur libre, l'espace mémoire est $\lg(N-1)+2$ bits, sur chaque processus, le temps de convergence est $O(N^3)$.

3.3.1.5 Algorithme de Johnen, SRDS 2002

Les algorithmes de circulation d'un jeton étaient, peu ou prou, tous basés sur le même mécanisme [Her90, BCD95, Ros00, DGT00, KY02b, DGT04], **[BGJ99]** et **[BGJ07]** « retarder de manière aléatoire la circulation des jetons ». Un processus ayant le jeton aléatoirement décide de transmettre ou non le jeton au voisin à sa droite. Ce choix probabiliste permet de garantir que les jetons ne progresseront pas dans l'anneau à la même vitesse. C'est pourquoi, il est inévitable qu'un jeton rattrape le jeton qui le précède ; une fois les deux jetons sur le même processus, ils fusionnent en un seul jeton ou les deux jetons disparaissent. Le problème de ce mécanisme est qu'une fois l'anneau stabilisé, l'unique jeton est aussi retardé. Donc, en moyenne, il mettra plus de N pas de calcul à faire le tour de l'anneau.

Hirotsugu Kakugawa et Masafumi Yamashita, [KY97], ont présenté le premier algorithme de circulation d'un jeton, basé sur un autre mécanisme : « la détection par un processus ayant un jeton, de la pluralité des jetons ». Une fois cette situation détectée, le processus bloque son jeton. Ce dernier sera rattrapé et absorbé par un autre jeton circulant. La difficulté de cette technique est d'éviter les situations d'inter-blocage où tous les jetons sont bloqués. Hirotsugu Kakugawa et Masafumi Yamashita ont analysé leur algorithme uniquement sous l'ordonnanceur centralisé. Chaque processus doit connaître la taille exacte de l'anneau. De plus, chaque processus a $O(N^2)$ états.

Algorithme 7 O_SSCTC- algorithme auto-stabilisant de circulation d'un jeton ayant un temps de service optimal

variables de p :
v_p est un entier borné par $BN > N$
r_p la signature booléenne de p

macros de p : - l est le voisin de gauche de p -
passer_jeton(p) = $v_p := (v_l + 1) \mod BN$

prédicats de p :
jeton(p) $\equiv v_p \neq (v_l + 1) \mod BN$
jeton_bloqué(p) $\equiv (v_p == 0) \wedge (r_l > r_p)$
mauvais_jeton(p) $\equiv \neg jeton(p) \wedge (r_p \neq r_l) \wedge (v_p \neq 0)$
tête_segment(p) $\equiv (v_l == B - 1)$

code de p :
$\mathcal{R}_1$: jeton(p) $\wedge$ ¬jeton_bloqué(p) $\wedge$ ¬tête_segment(p) $\rightarrow$ passer_jeton(p); $r_p := r_l$
$\mathcal{R}_2$: jeton(p) $\wedge$ ¬jeton_bloqué(p) $\wedge$ tête_segment(p) $\rightarrow$
passer_jeton(p); $r_p := random(p(0) = 1/2, p(1) = 1/2)$
$\mathcal{R}_3$: mauvais_jeton(p) $\rightarrow r_p := r_l$

J'ai présenté à « l'IEEE Symposium on Reliable Distributed Systems » en 2002 **[Joh02]**, une amélioration de leur algorithme de telle sorte que les processus n'ont plus besoin de connaître la taille exacte de l'anneau. Par contre, ils doivent connaître une borne supérieure à cette taille, nommé BN. Le code de cet algorithme (nommé O_SSCTC) est présenté dans Algorithme 7. J'ai fait une analyse exhaustive de l'algorithme O_SSCTC : preuve de convergence, calcul du temps de service et du temps de convergence. J'ai établi que l'algorithme O_SSCTC converge sous l'ordonnanceur libre.

Le temps de convergence de l'algorithme O_SSCTC est du même ordre que le temps de convergence de l'algorithme CTC et de l'algorithme de Datta, Gradinariu et Tixeuil, à savoir $0(N^3)$ pas de calcul.

Une fois l'anneau stabilisé, un processus a le jeton tous les N pas de calcul, quel que soit l'ordonnacement. Le temps de service fourni par l'algorithme O_SSCTC est optimal sous l'ordonnanceur libre, à ma connaissance c'est le seul algorithme ayant cette propriété.

L'algorithme O_SSCTC demande $\lg(N) + 1$ bits d'espace mémoire par processus.

3.3.1.6 Algorithme de Johnen, IPDPS 2004

J'ai présenté à « l'IEEE International Parallel & Distributed Processing Symposium » en 2004 **[Joh02]**, l'algorithme BS_SSCTC. Il s'agit d'un algorithme de circulation d'un jeton ayant une borne au temps de service. Le mécanisme d'élimination des jetons de l'algorithme BS_SSCTC n'est pas basé sur le retardement aléatoire de la circulation des jetons. Le mécanisme utilisé est celui proposé par Ted Herman dans [Her92]. Ted Herman a réalisé une version probabiliste de

Algorithme 8 BS_SSCTC - Circulation auto-stabilisante d'un jeton ayant un temps de service borné

paramètres :

m_N : plus petit non diviseur de N.
l : le voisin de gauche de p

variables de p :

dt est un entier positif inférieur à m_N.
pt est un entier positif inférieure à m_N.
c est un entier positif ou nul inférieur à 3.

prédicats de p :

jeton_déterministe(p) $\equiv dt_p \neq (dt_l + 1) \mod m_N$
jeton_probabiliste(p) $\equiv pt_p \neq (pt_l + 1) \mod m_N$
jeton(p) $\equiv$ (¬jeton_probabiliste(p) $\wedge$ $c_l \neq c_p$) $\vee$ (jeton_probabiliste(p) $\wedge$ $c_l == c_p$)

macros de p :

passer_jeton_déterministe(p) : $dt_p := (dt_l + 1) \mod m_N$
passer_jeton_probabiliste(p) : $pt_p := (pt_l + 1) \mod m_N$

code de p :

$\mathcal{R}_1$: jeton_probabiliste(p) $\wedge$ jeton_déterministe(p) $\wedge$ jeton $\longrightarrow$
passer_jeton_déterministe(p);
if random(p(0)=1/2,p(1)=1/2) == 1 **then**
passer_jeton_probabiliste(p) ; $c_p := c_l$;
else $c_p := (c_l + random(p(1) = 1/2, p(2) = 1/2)) \mod 3$;
$\mathcal{R}_2$: jeton_probabiliste(p) $\wedge$ jeton_déterministe(p) $\wedge$ ¬jeton $\longrightarrow$
passer_jeton_déterministe(p);
if random(p(0)=1/2,p(1)=1/2) = 1 **then**
passer_jeton_probabiliste(p) ; $c_p := c_l$;
$\mathcal{R}_3$: ¬jeton_probabiliste(p) $\wedge$ jeton_déterministe(p) $\longrightarrow$
passer_jeton_déterministe(p) ; $c_p := c_l$;

l'algorithme « K-state » de Dijkstra dans [Her92]. Cette version probabiliste nécessite 3 états par processus alors que l'algorithme de Dijkstra demande $N+1$ états par processus. Il s'agit d'un algorithme semi-uniforme sur un anneau, tous les processus ordinaires exécutent le même code, alors que l'unique leader exécute un code distinct. Le leader lorsqu'il a le jeton, change la couleur du jeton aléatoirement avant de le transmettre à son voisin de droite. Les processus ordinaires transmettent le jeton sans changer sa couleur. Le leader bloque tous les jetons qu'il reçoit et qui n'ont pas de sa couleur. Ainsi que le nombre de jetons est réduit jusqu'à n'obtenir un seul jeton. Cet unique jeton n'est jamais bloqué donc le temps de service est N. Ted Herman a prouvé la convergence de son algorithme uniquement dans le cas d'un ordonnanceur centralisé.

Pour utiliser le mécanisme de Ted Herman, l'anneau doit avoir un processus distingué (le leader). Pour cela, j'ai combiné l'algorithme $SSCTC$ avec l'algorithme de Ted Herman. Le rôle de l'algorithme $SSCTC$ est de sélectionner le processus distingué (il s'agit du processus ayant l'unique jeton probabiliste). Il exécute le code du leader de l'algorithme de Ted Herman ; les autres processus exécutent le code des processus ordinaires de l'algorithme de Ted Herman. Le code de l'algorithme BS_SSCTC est présenté dans Algorithme 8. Dans l'algorithme de BS_SSCTC, le processus distingué est circulant ce qui n'est pas le cas pour l'algorithme de Ted Herman. J'ai fait une analyse exhaustive de l'algorithme de BS_SSCTC : preuve de convergence, calcul du temps de service et du temps de convergence. L'algorithme de BS_SSCTC fonctionne correctement sous l'ordonnanceur libre. Une fois l'anneau stabilisé, un processus a le jeton tous les N^2 pas de calcul, quel que soit l'ordonnancement. Dans le cas d'un ordonnanceur synchronisant, un processus a le jeton tous les N pas de calcul. L'algorithme BS_SSCTC demande $2\lg(m_N))+2$ bits par processus. L'algorithme BS_SSCTC est donc asymptotiquement optimal en espace mémoire demandé ; de plus le temps de service est optimal sous un ordonnanceur synchronisant.

Le temps de convergence de l'algorithme $SSCTC$ et de l'algorithme de Datta, Gradinariu et Tixeuil sont du même ordre que le temps de convergence de l'algorithme CTC, à savoir $0(N^3)$ pas de calcul.

Modèle de communication par passage de message Dans [DHT04], Duchon, Hanusse et Tixeuil ont proposé un algorithme de circulation de jeton sur les anneaux unidirectionnels de toutes tailles. Le temps de service de cet algorithme est optimal. De plus, l'espace mémoire demandé par cet algorithme est constant quelle que soit le taille de l'anneau. Cet algorithme converge uniquement sous un ordonnanceur synchronisant. Le modèle de communication utilisé est le modèle par passage de messages. La problématique est de garantir qu'un seul message (jeton) circule dans l'anneau. Un mécanisme extérieur (par exemple, une libéralisation d'un jeton sur expiration d'un délai d'attente) garantit que l'anneau a au moins un jeton. Sous les mêmes hypothèses, dans [MOOY92, MOOY02], un algorithme de circulation d'un jeton sur les anneaux bidirectionnels de toutes tailles est proposé. L'espace mémoire demandé par cet algorithme est constant quelle que soit la taille de l'anneau. L'algorithme présenté dans [MOOY92, MOOY02], converge sous l'ordonnanceur libre.

3.3.2 Circulation d'un jeton sur les anneaux bidirectionnels

Amos Israeli et Marc Jalfon dans [IJ90b] ont présenté un mécanisme de circulation auto-stabilisant de jeton basé sur les marches aléatoires sur les anneaux unidirectionnels. Les jetons vont effectuer une marche aléatoire. Ils ne sont pas retardés dans leur mouvement, par contre leur direction change aléatoirement, ils décident, à chaque pas de calcul, équiprobablement d'aller sur leur droite ou sur leur gauche. Le temps de stabilisation de ce mécanisme à été analysé par Prasad Tetali et Peter Winkler dans [TW91], il est de $8N^2/27$. Le temps de service moyen est le temps moyen que l'unique jeton traverse tous les processus de l'anneau, il s'agit du temps de couverture de l'anneau par une marche aléatoire. Don Coppersmith, Uriel Feige et James B. Shearer dans [CFS96] ont établi que ce temps est $\Theta(N^2)$ pour les graphes réguliers (un anneau est un graphe 2-régulier). Une étude de l'ensemble des travaux sur les marches aléatoires par Lászlò Lovász est accessible via l'URL « http ://research.microsoft.com/users/lovasz/ », cette étude fut publiée dans [Lov96]. Dans [DL00], Jérome Durand-Lose a proposé un algorithme de circulation d'un jeton sur les réseaux bidirectionnels. Sur les anneaux bidirectionnels, cet algorithme est une implémentation du mécanisme proposé par Israeli et Jalfon. L'algorithme de Durand-lose nécessite un registre par voisin, ce registre peut prendre m_N values distinctes. Cet algorithme est facilement adaptable au modèle ACTION_COMPOSITE après avoir coloré les processus de tel sorte que deux processus à distance 2 ou 1 ont des couleurs distinctes. L'algorithme adapté au modèle ACTION_COMPOSITE nécessite $5{m_N}^2$ états sur chaque processus.

Une deuxième approche est de combiner de manière séquentielle un algorithme auto-stabilisant d'orientation d'un anneau bidirectionnel conçu pour le modèle ACTION_COMPOSITE avec un algorithme de circulation d'un jeton sur un anneau unidirectionnel. Amos Israeli et Marc Jalfon ont étudié l'orientation des anneaux bidirectionnels dans [IJ90a, IJ93]. Ils ont établi qu'il est impossible de concevoir un algorithme auto-stabilisant déterministe d'orientation d'un anneau fonctionnant sous l'ordonnanceur libre pour les anneaux de taille paire. Ils ont proposé un algorithme probabiliste nécessitant moins de 10 bits d'espace mémoire sur chaque processus. Cet algorithme converge sous l'ordonnanceur libre et pour toute taille d'anneau. Le temps de convergence de l'algorithme de Israeli et Jalfon est $O(N^2)$.

La composition séquentielle de l'algorithme de Israeli et Jalfon avec l'algorithme BS_SSCTC produit un algorithme auto-stabilisant de circulation d'un jeton sur un anneau bidirectionnel convergeant en $O(N^3)$ et nécessitant sur chaque processus $2\lg(m_N)) + 12$ bits d'espace mémoire. Cette composition a un temps de service de $2N^2$ pas de calcul, quel que soit l'ordonnancement. Dans le cas d'un ordonnanceur synchronisant, le temps de service est de N pas de calcul.

La composition séquentielle de l'algorithme de Israeli et Jalfon avec l'algorithme O_SSCTC produit un algorithme auto-stabilisant de circulation d'un jeton sur un anneau bidirectionnel convergeant en $O(N^3)$ et nécessitant sur chaque processus $\lg(N)) + 11$ bits d'espace mémoire. Cette composition a un temps de service de N pas de calcul, quel que soit l'ordonnancement.

3.3.3 Election d'un leader sur les anneaux unidirectionnels

Joffroy Beauquier, Maria Gradinariu et moi avons présenté un algorithme d'élection de leader sur un anneau unidirectionnel à « l'ACM Symposium on Principles of Distributed Computing »

Algorithme 9 LE - Election d'un leader sur un anneau unidirectionnel

paramètres :

m_N : plus petit non diviseur de N.
l : le voisin de gauche de p

variables de p :

ld est un entier positif inférieur à m_N.
c est un booléen.

prédicats de p :

$\text{Leader}(p) \equiv ld_p \neq (ld_l + 1) \mod m_N$

macros de p :

$\text{passer\_leader}(p) : ld_p := (ld_l + 1) \mod m_N$

code de p :

$\mathcal{L}_1 : \text{Leader}(p) \wedge c_p \neq c_l \longrightarrow$ **if** $(random(p(0) = 1/2, p(1) = 1/2) == 0)$ **then**
$c_p := c_l$; passer_leader(p) ;
$\mathcal{L}_2 : \text{Leader}(p) \wedge c_p == c_l \longrightarrow c_p := random(p(0) = 1/2, p(1) = 1/2)$;
$\mathcal{L}_3 : \neg\text{Leader}(p) \longrightarrow c_p := c_l$;

---

en 1999 **[BGJ99]**.
Cet algorithme, nommé LE, est publié dans le journal « Distributed Computing » .

Dans cet algorithme, l'encodage du leadership est identique à l'encodage du jeton_déterministe dans l'algorithme DTC, et à l'encodage du jeton _probabiliste dans CTC. Idem aux propriétés de l'algorithme DTC (présentées dans la section 2.2.1), l'anneau a toujours un leader, le nombre de leaders ne peut jamais croître quelle que soit l'exécution effectuée. Le code de l'algorithme LE est présenté dans Algorithme 9. Dans cette présentation les règles de l'algorithme LE doivent être corrélées avec la circulation d'un jeton. A savoir qu'un processus effectue une action de l'algorithme LE uniquement s'il a un jeton, et à la fin de l'action, il transmet le jeton à son voisin de droite. L'algorithme LE peut être combiné avec n'importe quel algorithme auto-stabilisant de circulation d'un jeton sur un anneau unidirectionnel (par exemple avec les algorithmes CTC $SSCTC$, O_SSCTC ou BS_SSCTC [KY97, DGT00, Ros00, KY02b, DGT04]).

Un leader choisit aléatoirement sa couleur $c1$ (règle $\mathcal{L}_2$), le jeton va convoyer la couleur $c1$ jusqu'au prochain leader (règle $\mathcal{L}_3$). Si ce leader n'a pas la couleur $c1$ alors il soupçonne de ne pas être l'unique leader sur l'anneau : la règle $\mathcal{L}_1$ est activable. Le leader choisit de manière équiprobable de garder la situation de leader ou de transmettre ce rôle à son voisin de droite. Si ce leader a la couleur $c1$ alors il soupçonne d'être l'unique leader sur l'anneau : la règle $\mathcal{L}_2$ est activable. Le leader choisit aléatoirement sa nouvelle couleur sans bouger.

Dans le cas où il y a une pluralité de leaders, (1) ces derniers vont tous essayer d'avancer, (2) leur vitesse de circulation est aléatoire ; (3) de plus, ils effectuent au cours d'une exécution quelconque le même nombre de pas de calcul à une constante près. Donc, avec une probabilité 1, deux leaders vont se rejoindre (pour fusionner en 1 seul leader, ou pour disparaître). Dans **[BGJ99]** et **[BGJ07]**, nous avons analysé en détail la combinaison de LE avec CTC. Nous

obtenons un algorithme,, nommé LE_k_borne, nécessitant $2\lg(m_N)+1$ bits d'espace mémoire par processus, l'algorithme LE_k_borne converge sous l'ordonnanceur k-borné. La cross-over composition de LE_k_borne avec DTC, $LE\_k\_borne \diamond DTC$, est un algorithme d'élection de leader sous l'ordonnanceur libre, nécessitant $3\lg(m_N)+1$ bits d'espace mémoire par processus.

Bornes sur l'espace mémoire Nous pouvons établir une borne inférieure à l'espace mémoire nécessaire sur chaque processus pour la circulation d'un jeton sur un anneau anonyme et unidirectionnel de la même manière que nous avons prouvé la borne pour l'élection de leader dans **[BGJ99]**.

Ainsi, nous avons établi que l'espace mémoire nécessaire à un algorithme auto-stabilisant d'élection de leader sur un anneau unidirectionnel supportant l'ordonnanceur libre est asymptotiquement équivalent à $\Theta(\lg(m_N))$ bits par processus. L'algorithme $LE\_k\_borne \diamond DTC$ établit une borne supérieure au nombre d'états nécessaire par processus $2m_N^3$, donc l'espace mémoire nécessaire est de $3.\ln(m_N)+2$ bits par processus.

3.3.4 Election d'un leader sur les anneaux bidirectionnels

La composition séquentielle de l'algorithme de Israeli et Jalfon avec l'algorithme $LE\_k\_borne \diamond DTC$ produit un algorithme auto-stabilisant d'élection d'un leader sur un anneau bidirectionnel nécessitant sur chaque processus $3\lg(m_N)+11$ bits d'espace mémoire.

3.4 Bilan

3.4.1 Tableaux récapitulatifs pour les algorithmes déterministes

type d'anneaux	**problème**	**borne inférieure à espace mémoire**	**borne supérieure à espace mémoire**
unidirectionnel	Circulation	$\lg(N)-1$ **[BGJ99]**	$\lg(N)+7$ **[FJ01]** avec [GH96]
unidirectionnel	Election	$\lg(N)$ **[BGJ99]**	$\lg(N)+4$ **[FJ01]**
bidirectionnel	Circulation	-	18 [ILS95, GH96, Hoe98]
bidirectionnel	Election	-	15 [ILS95, Hoe98]

3.4.2 Tableaux récapitulatifs pour les algorithmes probabilistes

type d'anneaux	**problème**	**temps de service**	**borne inférieure à espace mémoire**	**borne supérieure à espace mémoire**
unidirectionnel	Circulation	non optimal	$\lg(m_N - 2) - 1$ **[BGJ99]**	$2.\lg(m_N) + 2$ **[Joh04]**
unidirectionnel	Circulation	optimal	$\lg(m_N - 2) - 1$ **[BGJ99]**	$\lg(N + 1) + 2$ **[Joh02]**
unidirectionnel	Election	-	$\lg(m_N - 2) - 1$ **[BGJ99]**	$3.\lg(m_N) + 1$ **[BGJ07]**
bidirectionnel	Circulation	non optimal	?	$2.\lg(m_N) + 12$ **[Joh04]** avec [IJ93]
bidirectionnel	Circulation	optimal	?	$\lg(N + 1) + 11$ **[Joh02]** avec [IJ93]
bidirectionnel	Election	-	?	$3.\lg(m_N) + 10$ **[BGJ07]** avec [IJ93]

Chapitre 4

Algorithmiques pour les réseaux Ad Hoc

L'objectif de ce chapitre est de présenter mes travaux relatifs à la construction des protocoles auto-stabilisants adaptés aux réseaux dont la topologie varie très fréquemment.

Un protocole auto-stabilisant converge automatiquement d'une configuration quelconque à une configuration légitime à partir de laquelle le système aura un comportement correct. Aucune propriété n'est garantie durant la phase de convergence. Donc le système peut avoir un comportement complètement arbitraire durant toute la phase de convergence. Dans le pire cas, la durée de cette phase de convergence est souvent proportionnelle à la longueur du réseau, pour les tâches suivantes l'élection de leader, la circulation d'un jeton, les protocoles de construction de table de routage, certains protocoles d'agrégation, ...

C'est pourquoi, les protocoles auto-stabilisants sont relativement mal adaptés aux réseaux dont la topologie varie très rapidement. Les réseaux Ad-Hoc, les réseaux mobiles, les réseaux de très grandes tailles font partie de cette catégorie. Néanmoins, les protocoles auto-stabilisants ont des qualités intéressantes puisqu'ils garantissent la convergence automatique dans un état légitime quel que soit l'état initial. Donc, les protocoles auto-stabilisants refonctionneront correctement après une rafale de perturbations ou changements de topologie. Une fois cette rafale passée, un protocole auto-stabilisant convergera, vers un comportement correct.

Je me suis donc intéressée à construire des protocoles auto-stabilisants et robustes.

4.1 Les protocoles robustes

La notion de protocole robuste fut proposé par Mohamed Gouda et Jorge Cobb dans [CG01, CG02]. Un protocole auto-stabilisant est **robuste** si et seulement s'il existe un prédicat $\mathcal{SP}$, nommé le **prédicat de sureté**, sur les configurations tel que

- Une fois le prédicat $\mathcal{SP}$ vérifié, il reste vérifié après toute action du protocole.
- Une fois le prédicat $\mathcal{SP}$ vérifié, le système peut fonctionner en mode dégradé.
- Le temps de convergence du protocole à une configuration vérifiant le prédicat $\mathcal{SP}$ doit être très rapide.

Une configuration est dite **sûre** si elle vérifie le prédicat $\mathcal{SP}$. Dans le cas d'un protocole auto-stabilisant et robuste, le système va fonctionner très rapidement, en mode dégradé, même après

une rafale de perturbations. Un protocole auto-stabilisant et robuste est donc très intéressant pour les réseaux dont la topologie change très fréquemment tel que les réseaux Ad-Hoc, les réseaux mobiles et les réseaux de très grande taille.

Un protocole robuste est super-stabilisant. La définition des protocoles super-stabilisants fut proposée par Shlomi Dolev et Ted Herman dans [DH95a, DH95b, DH97]. Un protocole est super-stabilisant par rapport aux changements de la classe Λ si (1) il est auto-stabilisant et (2) le prédicat $\mathcal{SP}$ est vérifié après UN changement de la classe Λ. Un protocole robuste est toujours un protocole super-stabilisant par rapport aux changements préservant le prédicat $\mathcal{SP}$. De plus, un protocole robuste garantit, dans tous les cas, une convergence rapide vers une configuration sûre.

La notion de sûreté est aussi liée à la notion auto-organisation [DT06]. Un protocole est auto-organisé si (1) il est auto-stabilisant et il converge très « rapidement » après un seul changement de topologie. La notion de convergence rapide n'est pas formellement définie, mais le temps de convergence après un changement doit être très inférieur au pire temps de convergence.

4.2 L'agrégation

Le problème de l'agrégation (en anglais, *clustering*) consiste à partitionner les nœuds d'un réseau en grappes. Dans le cas de grappes de diamètre 2 (en anglais 1-hop cluster) les membres de la grappe sont à distance 1 du leader de la grappe (appelé en anglais clusterhead).

Dans le cadre de grappes de diamètre 2, les protocoles proposés sont presque tous basés sur le même principe : Un nœud décide de devenir chef de groupe s'il n'existe pas dans son voisinage à distance 1 un ou des nœuds plus apte(s) à être leader(s). Ensuite, les voisins des leaders se rallient à eux. De manière générale, un nœud devient leader après avoir été informé que les nœuds de son voisinage plus aptes que lui-même à jouer le rôle de leader ont rejoint une grappe existante et ne seront donc pas leaders de grappe. Ensuite, ses voisins qui n'ont pas intégré une grappe se joignent à sa grappe, ainsi de suite. Le processus se termine lorsque tous les nœuds ont intégré une grappe. Avec ce type de mécanisme, l'ensemble des leaders de grappe forment un ensemble dominant. Un ensemble de nœuds d'un graphe est **dominant** si chaque nœud n'appartenant pas à cet ensemble est à distance 1 d'un nœud dominant.

Les protocoles classiques de construction de grappes se différencient principalement en fonction du critère de sélection des leaders de grappe :

- dans [BE81, EWB87, MGM03, XHGS03, TV05, DFG06], le critère de sélection est la valeur de l'identifiant du nœud - Un nœud ayant le plus petit identifiant dans son voisinage s'auto-sélectionne comme leader. Les protocoles de la deuxième génération [MGM03, XHGS03, TV05, DFG06] sont auto-stabilisants.
- dans [GT95], le critère de sélection est le degré des nœuds - plus le degré d'un nœud est grand plus il est apte à être chef d'une grappe.
- dans [BKL01], une métrique représentant la stabilité du nœud est utilisé. Un nœud gardant le même voisinage est considéré comme peu mobile ; il est donc un bon candidat au rôle de chef de grappe.

- dans [CDT02], une métrique composée de quatre paramètres est le critère de sélection. Cette métrique prend en compte le degré du nœud, la qualité de la transmission avec ses voisins, la mobilité du nœud et l'énergie restante dans sa batterie.

Quel que soit le critère de choix des leaders de grappe, le mouvement d'un seul leader de grappe, où l'arrivée dans le réseau d'un nœud apte à devenir chef de grappe peut entraîner une réaction en chaîne qui de proche en proche conduit à la refonte de tous les grappes. Durant la réaction en chaîne la couverture du réseau par des grappes n'est plus garantie. Donc ce type de protocoles n'est pas bien adapté aux réseaux mobiles ou aux réseaux de grande taille. La topologie de ces réseaux change très fréquemment ; il est donc important de garantir la stabilité de la structure formée par les grappes. Des changements de moindre « importance » ne doivent pas enclencher une réaction en chaîne disproportionnée. Isabelle Mitton, Anthony Busson et Eric Fleury, dans [MBF04], ont choisi un critère de sélection (la densité du voisinage du nœud) qui permet de construire une structure relativement stable qui n'est pas remise en cause par des modifications de topologie de moindre importance.

Stefano Basagni, dans [Bas99], propose une technique générale de construction de grappes de diamètre 2. Cette technique est appelée **GDMAC**, generalized Distributed Mobility-Adaptative Clustering. La sélection des responsables de grappe est effectuée en fonction de leur poids. Le poids correspondant à un critère (le degré du nœud, son identifiant) ou à une somme pondérée de critères. Ainsi, cette technique ne dépend pas du critère de sélection des leaders. Ce mécanisme peut donc être adapté aux critères de choix proposés dans [BE81, EWB87, GT95, BKL01, CDT02, MBF04]. Cette technique permet de limiter les réactions en chaîne de reconstruction de grappes, sans les éviter totalement. Une évaluation empirique de cette technique a été effectuée par Christian Bettstetter et Bastian Friedrich, dans [BF03]. Deux mécanismes sont proposés dans GDMAC pour limiter le nombre des reconstructions de grappes tout en permettant d'avoir une structure utile. Une structure est utile si le nombre de grappes est largement inférieur au nombre de nœuds et si les leaders de grappe soient à même d'assurer leur fonction (c'est-à-dire qu'ils aient des poids en accord avec leur rôle).

- Autoriser dans un voisinage plusieurs chefs de grappe. L'objectif est d'éviter la destruction de grappes. Néanmoins, il est souhaitable de limiter le nombre de chefs de grappes pour avoir une structuration hiérarchique utile. C'est pourquoi, cette technique limite le nombre de leaders dans le voisinage d'un autre leader à k. La valeur de k est un paramètre qui peut varier au cours de temps et varier selon les nœuds. Plus la valeur de k est importante, plus sera grand le nombre de grappes. Ainsi, si la valeur de k est égale au degré des nœuds, la technique de Basagni peut conduire à une structure ayant autant de grappes que de nœuds. En contre partie, plus la valeur de k est importante, plus la réaction en chaîne de reconstruction des grappes sera rare. Si la valeur de k est égale à 1 alors l'ensemble des chefs de grappe forment un ensemble indépendant. Un ensemble de nœuds d'un graphe est dit **indépendant** si les nœuds deux à deux de cet ensemble sont à une distance supérieure à 1.
- Autoriser qu'un nœud ne soit pas affilié avec le leader optimal. Le leader optimal pour un nœud est le chef de grappe dans son voisinage qui a le plus grand poids. L'objectif de ce mécanisme est d'éviter les restructurations des grappes. Néanmoins, il est souhaitable que les grappes soient gérées par des nœuds de capacité suffisante. C'est pourquoi, la

différence de capacité entre le leader optimal et le leader d'un nœud est limité par une valeur prédéfinie, nommée h. La valeur de h est un paramètre qui peut varier au cours de temps et varier selon les nœuds. Si la valeur de h est 0, alors les nœuds sélectionnent la grappe de leur voisinage gérée par le nœud de plus grand poids. Si la valeur de h est infinie, alors les nœuds peuvent être dans n'importe quelle grappe de leur voisinage dont le leader a un poids plus grand que le leur.

La technique de GDMAC partitionne les nœuds en grappes bien formées. Les **grappes bien formées** vérifient les deux propriétés suivantes :

- les grappes forment une partition de l'ensemble des nœuds : chaque nœud appartient à une et une seule grappe.
- chaque nœud est à distance 1 de son chef de grappe. Donc les grappes construites sont de diamètre 2 et les chefs de grappe forment un ensemble dominant.

Le Huy Nguyen et moi avons présenté une version auto-stabilisante de GDMAC à AlgoSensor en 2006 **[JN06a]**.

Le Huy Nguyen et moi avons conçu un protocole de construction de clusters ayant des objectifs orthogonaux à ceux de l'algorithme proposé dans **[JN06a]**. Il s'agit de construire des grappes bien formées de taille proportionnée :

- le chef d'une grappe a un poids plus fort que tous les autres membres de sa grappe.
- la taille des grappes est limitée à une valeur prédéfinie - nommée s. La charge de travail d'un responsable de grappe est proportionnel à la taille de sa grappe, ce choix permet donc de ne pas 'surcharger' les leaders.
- Pour limiter le nombre de grappes, plusieurs responsables de grappe sont voisins uniquement en cas de nécessité : ces responsables ne peuvent intégrer une grappe pré-existante car elles sont « complétées » (elles ont s membres).

Les protocoles d'agrégation présentés dans [BK01, VET07] limitent la taille des grappes. Dans [BK01], les grappes ont une taille minimale et maximale, par contre un nœud peut appartenir a plusieurs grappes. Le protocole présenté dans [VET07] utilise un arbre de couverture pour construire des grappes de taille limitée. Ce deuxième protocole est présenté à ISPA, en 2007 **[JN07]**.

Ce deuxième protocole **[JN07]** construit, très rapidement, des grappes ayant une taille inférieure ou égale à s. Cette propriété est close : une fois qu'une grappe a moins de s membres, sa taille ne sera jamais supérieure à la valeur s. Le protocole **[JN07]** converge en moins de $3D + 8$ étapes asynchrones (D étant le diamètre du réseau). Après seulement 3 étapes asynchrones, les grappes contiennent moins de s membres. Informellement, une étape **asynchrone** (en anglais, asynchrone round) est une série de pas de calcul consécutifs où tous les nœuds/processus qui pouvaient effectuer une action l'ont fait ou ont arrêté de pouvoir effectuer d'action. Ainsi, dans le cas où tous les processus ou nœuds sont synchrones, une configuration sûre est atteinte en un seul pas de calcul.

Il est souhaitable qu'à tout moment, les grappes soient bien formées. Après un changement de topologie du réseau, par exemple le mouvement d'un nœud, la panne franche d'un chef de

grappe, une coupure de communication entre un nœud et son chef de grappe, etc la structure n'est plus maintenue. Il faut donc rétablir cette structure très rapidement. Des protocoles probabilistes construisant une partition très rapidement sont proposés dans [YF04a, YF04b, DT06]. La sélection est effectuée de manière probabiliste uniforme pour [DT06] et non-uniforme pour [YF04a, YF04b]. Dans [YF04a, YF04b], la probabilité de devenir leader pour un nœud varie en fonction de son énergie résiduelle. Le temps de construction de la structure de grappes est rapide, en moyenne. Cependant, dans le pire des cas, ces protocoles ne sont pas plus rapides que les protocoles déterministes. De plus, aucune propriété sur les chefs de grappe ne peut être garantie.

4.2.1 Protocoles robustes d'agrégation

A OPODIS en 2006, Le Huy Nguyen et moi avons présenté une version auto-stabilisante et robuste de GDMAC **[JN06b]**. La propriété de sûreté intéressante dans le cas de protocole de construction de grappes est la conservation de la structure de grappes. Si les grappes sont bien formées alors la structure en grappe est pérenne. Nous avons prouvé qu'une configuration sûre est atteinte en 1 étape asynchrone.

Le temps de convergence du protocole **[JN06b]** à une configuration où la structure est stabilisée nécessite au plus $2D + 4$ étapes asynchrones (D étant le diamètre du réseau) quelles que soient la configuration initiale et la topologie du réseau.

4.3 Construction des tables de Routage

Le routage est une tâche de la couche réseau (couche 3 du modèle OSI) qui permet de déterminer la prochaine passerelle sur le chemin que devra suivre le paquet pour rejoindre sa destination. Dans le cas de réseau Ad-Hoc, la couche réseau fonctionnant en mode non connecté, cette tâche doit être répétée pour chaque paquet entrant, d'où la nécessité d'une prise de décision rapide. Traditionnellement, dans le cas de protocole de routage pro-actif, une table maintenue par chaque routeur d'un réseau, **la table de routage**, permet d'établir une correspondance entre la destination finale du paquet, et l'adresse du prochain routeur sur le chemin.

Dans le cas de réseau Ad-Hoc, cette table doit être maintenue dynamiquement. De plus, elle ne doit nécessiter aucune configuration initiale. Typiquement, un protocole auto-stabilisant de construction de table répondrait à ces besoins. Il existe plusieurs protocoles auto-stabilisants de construction de tables sur le principe du vecteur de distance (en anglais, distance vector) [AGH90, Dol97, CG02, DT01, GS99]. Dans [AGH90, Dol97], les protocoles construisent le chemin de plus petit coût. Les protocoles de [DT01, GS99] sont génériques, le critère de sélection de la route peut être divers : maximiser la bande passante, minimiser le coût du chemin, ... Seul le protocole présenté dans [CG02] est sans boucle (en anglais loop-free). Le concept de protocole de construction de **tables de routage sans boucle** a été proposé par Jose Joaquin Gracia-luna-Aceves dans [GLA93]. Une destination est dite avec boucle à un instant donné, si le chemin spécifié dans les tables de routage pour cette destination précise passe deux fois par le même routeur. Un protocole construit des tables de routage sans boucle s'il garantit qu'une destination sans boucle reste sans boucle, même durant la phase de reconstruction des tables pour optimiser

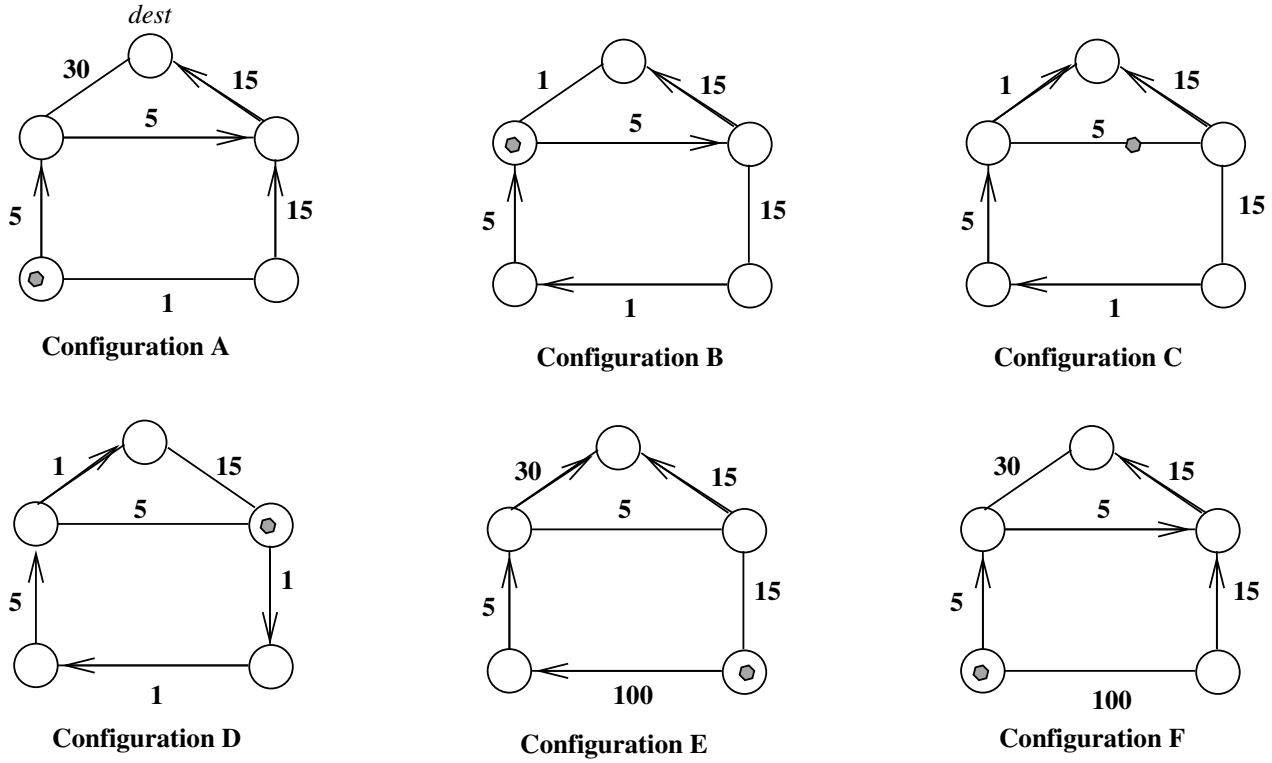

FIG. 4.1 – protocole de routage sans boucle, mais sans garanti d'acheminement

le chemin pris par les paquets. Les inconvénients du protocole de Cobb et Gouda sont que (1) chaque nœud doit connaître la taille du réseau et que (2) l'acheminement des paquets à la destination n'est pas garanti. Ce dernier point est illustré dans la figure 4.1. A tout instant, le nœud *dest* est une destination sans boucle, néanmoins un paquet à destination du nœud *dest* peut boucler dans le réseau sans jamais atteindre *dest*.

Sébastien Tixeuil et moi avons conçu un protocole auto-stabilisant de type vecteur de distance qui construit des tables de routage sans boucle **[JT03]**. Les deux inconvénients du protocole de Cobb et Gouda sont corrigés. Les routeurs n'ont pas besoin de connaître la taille du réseau. De plus, notre protocole garantit la propriété de sûreté suivante : « le poids d'un paquet est le coût du chemin de l'émetteur au destinataire *dest* et le poids d'un paquet diminue à chaque saut (en anglais, à chaque hop) ». Ainsi tout paquet arrive à destination en un nombre fini de sauts, ce nombre étant borné par le coût initial du chemin de l'émetteur au destinataire. Cette propriété est conservée même durant la phase de convergence où les tables de routage sont modifiées pour construire les routes de plus petits coûts.

Le principe de notre protocole est de limiter les circonstances où un routeur peut augmenter le coût de ses routes de telle sorte que la propriété de sûreté est toujours vérifiée. Un routeur r peut augmenter le coût de sa route à destination de *dest* uniquement si (1) les routes passant par r pour *dest* auront encore après cette augmentation un coût plus important que la route à partir

de r et si (2) il n'a pas dans ces files d'attente des paquets à destination de *dest*. Pour éviter les situations de blocage, un routeur r devant augmenter le coût d'un chemin informe les routeurs voisins de son besoin. A charge pour ces derniers de procéder à l'augmentation du coût de leur route, si elle passe par r. Une fois que les routes passant par r ont un coût acceptable, le nœud r peut augmenter le coût de sa route après avoir transité les paquets en attente.

Le fonctionnement de notre protocole est illustré dans la figure 4.2. Dans la configuration F, le routeur v ne peut pas augmenter le coût de sa route car il a un paquet en transit. Une fois ce message transmis au routeur w, v peut augmenter le coût de sa route. Le nœud w met à jour sa route à destination de *dest* uniquement après que le nœud v ait mis à jour sa propre route (car elle passe par w) et après avoir routé le paquet en attente.

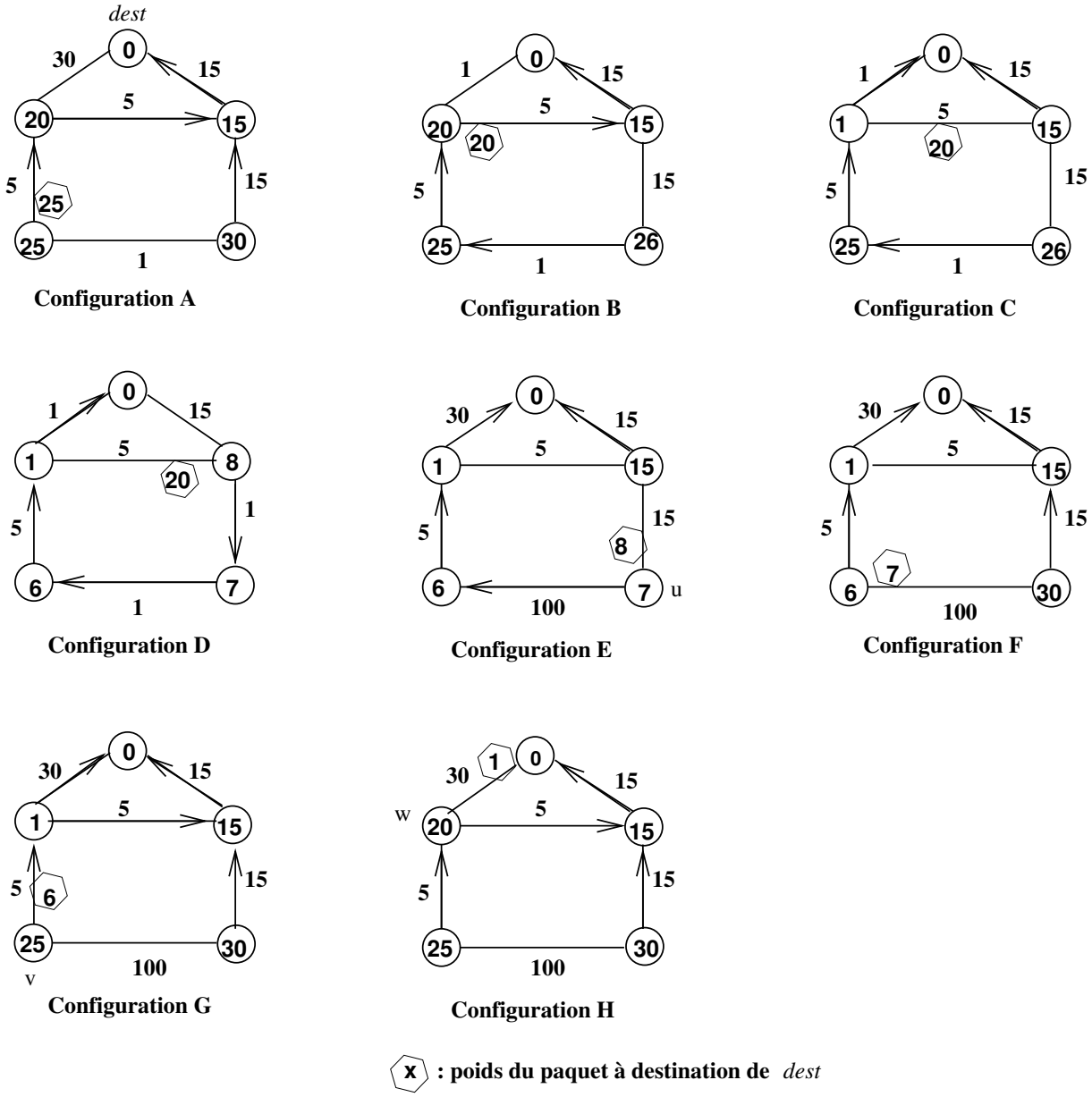

FIG. 4.2 – routage à l'aide du protocole **[JT03]**, avec garanti d'acheminement

Chapitre 5

Perspectives

L'objectif de ce chapitre est de présenter mes projets de recherche dans les trois axes développés dans ce document à savoir :

- compilateurs tolérants aux pannes
- algorithmique classique et auto-stabilisante
- algorithmes pour les réseaux Ad-Hoc

5.1 Compilateurs tolérant aux pannes

5.1.1 Modèle de communication par registres

Dans le cas de communication par registres partagés, l'unique moyen de communiquer entre deux processus est l'écriture/lecture dans des registres partagés. Il existe 6 modèles de communication basés sur les registres en fonction de la sémantique et de la location des registres. Les modèles sont présentés dans la figure 5.1 et définis dans la section 2.1.

	location sur les nœuds	*location sur les liens*
registre atomique	NOEUD-ATOMIQUE	LIEN-ATOMIQUE
registre régulier	NOEUD-RÉGULIER	LIEN-RÉGULIER
registre sûr	NOEUD-SÛR	LIEN-SÛR

FIG. 5.1 – Six modèles de communication par registre

Pour compléter notre étude, nous devons nous intéresser aux registres sûrs. En particulier, il faudrait proposer un compilateur qui transforme un algorithme auto-stabilisant conçu dans le modèle LIEN-RÉGULIER, par un algorithme conçu dans le modèle LIEN-SÛR. Un algorithme auto-stabilisant d'exclusion mutuelle conçu dans le modèle sûr-SR fournit ce service [Lam74, Lam85] (l'accès au registre est une section critique). Mais, est-il absolument nécessaire de ralentir les processus en leur interdisant d'effectuer simultanés des opérations sur le même registre ? Je conjecture que non dans la mesure où l'on autorise que l'implémentation de la lecture d'un registre régulier contienne des écritures dans des registres sûres. Ainsi, il serait possible d'avoir un compilateur sans attente et auto-stabilisant.

Je conjecture qu'il n'existe pas d'implémentation sans-attente et auto-stabilisant d'un registre régulier mono-écrivain et mono-lecteur par des registres sûrs mono-écrivain et mono-lecteur, si l'implémentation de la lecture d'un registre régulier ne peut contenir que des lectures. Cette conjecture rejoint la conjecture de Jaap-Henk Hoepman, Marina Papatriantafilou et Philippas Tsigas, dans [HPT02]. Ainsi que le résultat de Gregory Chockler, Rachid Guerraoui and Idit Keirar présenté dans [GCK07] qui établit qu'il n'existe pas d'implémentation d'un registre régulier ayant un domaine de valeur non borné via un compilateur amnésique utilisant uniquement des registres sûres.

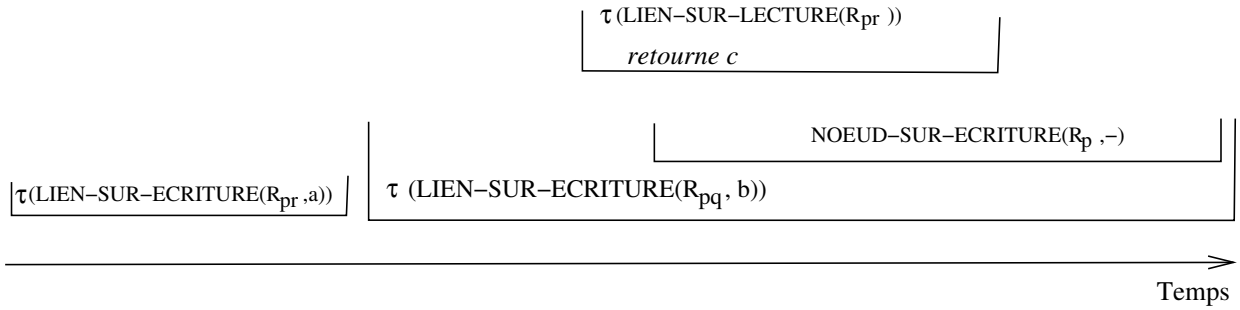

FIG. 5.2 – Exécution d'un algorithme conçu pour le modèle LIEN-SÛR transformé dans le modèle NOEUD-SÛR, par la « sûre-transformation par identifiant »

La sûre-transformation par identifiant (présentée dans la section 2.1.5) n'est pas un compilateur du modèle LIEN-SÛR dans le modèle NOEUD-SÛR. Par exemple, l'interprétation de l'exécution présentée dans la figure 5.2 ne peut pas être réalisée dans le réseau lien-sûr(G). L'exécution de τ(LIEN-SÛR-LECTURE($\mathcal{R}_{pr}$)) chevauchent la période de temps où le processus p exécute NOEUD-SÛR-ÉCRITURE($\mathcal{R}_p$, $\dot{}$) Ainsi, l'exécution de τ(LIEN-SÛR-LECTURE($\mathcal{R}_{pr}$)) peut retourner n'importe quelle valeur, dans notre cas, elle retourne la valeur c (qui n'est pas la dernière valeur écrite dans le registre $\mathcal{R}_{pr}$). Aucune exécution dans le modèle LIEN-SÛR ne peut produire un tel comportement. Il serait intéressant de déterminer si il est possible et sous quelle condition de construire un compilateur du modèle NOEUD-SÛR dans le modèle LIEN-SÛR.

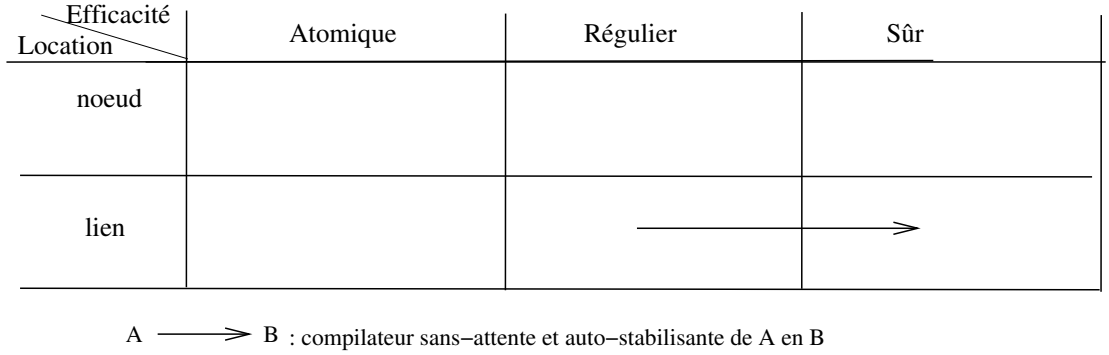

FIG. 5.3 – Conjectures sur les compilateurs tolérant aux pannes

5.1.2 Lien bidirectionnel versus lien unidirectionnel

Dans la section 2.1, nous avons considéré que les liens de communication sont bidirectionnels. Un processus p est voisin du processus q si le processus p peut lire dans un registre appartenant à q et si le processus q peut lire dans un registre appartenant à p. Il existe une autre famille de graphe : les graphes unidirectionnels. Dans ce type de graphe, les liens de communication sont orientés. Dans le cas d'un lien unidirectionnel du processus p au processus q, le processus p peut communiquer des informations au processus q, sans que ce dernier puisse répondre (c'est-à-dire que p possède un registre lisible par q, mais q possède aucun registre lisible par p).
Dans [DH01], Shlomi Dolev et Ted Herman ont proposé une version de l'algorithme de Dijkstra « K-states » dans le modèle LIEN-RÉGULIER. L'algorithme **« K-state » de Dijkstra** [Dij74] est un algorithme auto-stabilisant de circulation d'un jeton sur les anneaux unidirectionnels (nous avons parlé de cet algorithme dans le pragraphe 2.2). Dans le même papier, il est établi qu'il n'existe pas de version de l'algorithme de Dijkstra « K-states » utilisant un registre sûr par lien de communication. Il serait intéressant de proposer une version de l'algorithme de Dijkstra utilisant plusieurs registres sûrs par lien de communication, ou de prouver qu'un tel algorithme ne peut être conçu.

5.1.3 Compilateur d'action composite en actions simples

Le premier type de compilateurs auto-stabilisants du modèle ACTION_COMPOSITE dans le modèle LIEN-ATOMIQUE est basé sur un algorithme auto-stabilisant de circulation d'un jeton conçu dans le modèle LIEN-ATOMIQUE. L'idée est qu'un processus réalise une action composite de l'algorithme d'origine uniquement lorsqu'il a un jeton. L'algorithme de circulation étant auto-stabilisant, il atteint un état légitime à partir duquel un seul jeton circule dans le réseau. À partir de cet état, les exécutions de l'algorithme cible sont interprétables dans le modèle ACTION_COMPOSITE. Plus précisément, elles correspondent aux exécutions où à chaque instant, un seul processus effectue une action composite. En autre terme, il s'agit des exécutions obtenues sous un ordonnanceur centralisé (ce terme est défini dans la section 2.2.1). Un premier compilateur de ce type fut proposé dans [Dol00], il est obtenu par la composition équitable des algorithmes auto-stabilisants suivant conçus dans le modèle LIEN-ATOMIQUE : élection d'un leader [DIM97], construction d'un arbre de couverture dont l'origine est le leader [DIM93], construction d'un anneau eulérien, et une adaptation de l'algorithme de circulation d'un jeton de Dijkstra « K-state ».

Ce type de solution n'est pas bien adapté au problème initial car il interdit que plusieurs processus effectuent des actions composites simultanément. Or le modèle ACTION_COMPOSITE autorise plusieurs processus à effectuer simultanément une action. C'est pourquoi, ce type de compilateurs n'est pas très performant : un processus doit attendre d'avoir le jeton pour exécuter une action composite - il attend donc en moyenne $O(N)$ pas de calcul, dans le cas d'un réseau ayant N processus -.

Il a été proposé un deuxième type de compilateurs auto-stabilisants basé sur un algorithme auto-stabilisant exclusion mutuelle locale [AS99a, AS99b, MN98, Hua00, SCP03]. Un algorithme **d'exécution mutuelle locale** garantit qu'un seul processus a accès à la section critique dans

un voisinage. Plus précisément, le prédicat *EX_ locale* sur les exécutions est vérifié : *si un processus est dans la section critique alors aucun de ses voisins n'y est et chaque processus accède infiniment souvent à la section critique*. L'idée est similaire au premier type de compilateurs, un processus effectue une action composite de l'algorithme d'origine uniquement dans la section critique. L'algorithme d'exclusion mutuelle étant auto-stabilisant, il atteint un état légitime. A partir d'un état légitime, les exécutions de l'algorithme cible correspondent aux exécutions de l'algorithme d'origine où deux voisins n'exécutent pas simultanément une action composite. Ce type de compilateurs est plus performant que le premier type ; parce qu'il autorise plusieurs processus à effectuer simultanément une action composite. Les compilateurs de Gheorghe Antonoiu et Pradip K. Srimani [AS99a, AS99b] adaptent un algorithme conçu dans le modèle ACTION_COMPOSITE au modèle NOEUD-ATOMIQUE. Le compilateur présenté par Shing-Tsaan Huang [Hua00] adapte un algorithme conçu dans le modèle ACTION_COMPOSITE au modèle LIEN-ATOMIQUE mais il ne fonctionne que sur des réseaux bidirectionnels où les liens de communication ont une orientation formant un graphe acyclique.

Les compilateurs auto-stabilisants présentés dans [MN98, SCP03] sont basés sur l'exclusion mutuelle locale à la demande. Un processus doit demander l'accès à la section critique, dans le cas contraire il ne participe pas à la compétition. Plus précisément, le prédicat *EX_ locale_ aLaDemande* sur les exécutions est vérifié : *si un processus est dans la section critique alors aucun de ses voisins n'y est et un processus demandant l'accès à la section critique, y accédera*. Donc, un processus n'est pas obligé d'entrer régulièrement dans la section critique. Ce type de compilateurs peut préserver la propriété de « quiétude » à la différence des types précédents. Si l'algorithme initial est « silencieux » [DGS96] alors l'algorithme compilé l'est aussi. Les compilateurs présentés dans [MN98, SCP03] ont une caractéristique qui les rend difficilement réalisables : ils utilisent une estampille chronologique non bornée dont la valeur ne peut que croître. Dans le cas d'algorithme auto-stabilisant, la valeur d'une estampille non bornée ne peut pas avoir une implémentation non bornée : initialement, elle peut avoir n'importe quelle valeur 2^{32}, 2^{64}, 2^{128}, ... et ces valeurs ne peuvent que croître. Donc, l'espace mémoire nécessaire à ces compilateurs est effectivement non borné. Dans [BPV04], Christian Boulinier, Franck Petit et Vincent Villain ont établi qu'il était possible de borner la valeur des estampilles, en effectuant les incrémentations sur les estampilles modulo une valeur prédéfinie, appelée B. La valeur de B dépend de propriétés structurelles du graphe tel que la taille maximale « d'un trou » (cycle sans raccord ou raccourci - le plus petit chemin entre deux processus sur le cycle est un segment du cycle). Un compilateur utilisant une estampille bornée n'est pas très adaptatif : en cas de changement de topologie du graphe, la valeur de B doit être recalculée et communiquée à tous les processus du réseau.

Le compilateur auto-stabilisant présenté par Mikhail Nesterenko et Anish Arora dans [NA02b] est d'un troisième type. Il est basé sur un mécanisme d'autorisation ponctuelle. Un processus désirant exécuter une action composite demande une autorisation à tous ses voisins. Une fois les autorisations obtenus, il est à même d'exécuter cette action (consommant, par ce fait, les autorisations obtenues). Sans la participation de ses voisins (ils doivent écrire dans l'un de leur registre pour donner leur autorisation), un processus ne peut pas exécuter une action composite. Ce compilateur demande donc la participation de tous les processus : un processus doit répondre aux demandes de ses voisins, même s'il n'aura jamais plus besoin d'exécuter d'action composite. Néanmoins, le compilateur de [NA02b] préserve la propriété de terminaison. Si l'algorithme d'origine termine alors l'algorithme cible termine aussi. On dit qu'un algorithme a terminé une

fois qu'aucune écriture dans des registres de communication n'est effectuée. Par ailleurs, c'est le seul compilateur à notre connaissance qui n'utilise pas d'estampilles chronologiques. Le compilateur de Mikhail Nesterenko et Anish Arora [NA02b] adapte un algorithme conçu dans le modèle ACTION_COMPOSITE au modèle NOEUD-ATOMIQUE.

5.1.4 Projet

Il serait intéressant d'avoir un compilateur auto-stabilisant ne demandant pas la participation récurrente de tous les processus. Une fois le système stabilisé, lorsqu'un processus n'a plus d'action composite à effectuer alors ce processus devrait être silencieux dans le système compilé. Ce compilateur pourrait être basé sur une solution du problème du dîner des philosophes. A chaque lien de communication est associé une ressource (appelée *fourchette*) qui est la copropriété des deux processus connectés par le lien. A instant donné, une *fourchette* est à la disposition d'un seul de ses deux copropriétaires. Lorsqu'un processus a, à sa disposition, toutes les fourchettes dont il est copropriétaire, il peut « dîner » (c'est-à-dire qu'il peut effectuer une action composite).

Un algorithme réparti est tolérant aux pannes franches à distance d ou plus si l'impact de la panne franche d'un processeur p est circonscrit uniquement aux processus à une distance inférieure ou égale à d de p. Des solutions au problème du dîner des philosophes tolérant aux pannes franches à distance 2 ont été proposés dans [TB94, CS96, SPS00]. De plus, il a été prouvé l'optimalité de la valeur 2. Une solution auto-stabilisante dans le cadre d'actions composites au problème du dîner des philosophes tolérant aux pannes franches à distance 2 a été présentée par Mikhail Nesterenko et Anish Arora dans [NA02a]. Il serait intéressant d'avoir un compilateur auto-stabilisant qui soit tolérant aux pannes franches à distance 2 de modèle ACTION_COMPOSITE dans le modèle LIEN-ATOMIQUE.

5.2 Algorithmes probabilistes et auto-stabilisants

Les résultats obtenus sur l'espace mémoire nécessaire aux algorithmes probabilistes d'élection de leader et de circulation d'un jeton sont synthétisés dans la figure 5.4. Ils ouvrent de nombreuses pistes de réflexions.

La bidirection est-elle un avantage pour établir la circulation d'un jeton ? Dans le cas d'algorithme déterministe la réponse est clairement oui, au regard de l'espace mémoire par processus : dans un cas il faut $O(N)$ états ; dans l'autre cas, un nombre d'états constant, pour toutes les tailles d'anneau, est suffisant.

Qu'en est-il au regard de l'espace mémoire nécessaire à un algorithme probabiliste de circulation d'un jeton ? Dans [IJ90b], Amos Israli et Marc Jalfon ont établi une borne inférieure au nombre d'états par processus nécessaire à un tel algorithme : $\sqrt{m_N}$. Cette borne est calculée pour les algorithmes tel qu'une fois stabilisé, seul le processus ayant le jeton est activable.

Je conjecture qu'il est possible d'établir une borne inférieure de l'ordre de $min((m_N-3)/3, \sqrt{m_N})$. Il serait intéressant de concevoir un algorithme de circulation de jeton sur les anneaux bidirectionnels nécessitant $O(\sqrt{m_N})$ états par processus, ou du moins, nécessitant significativement moins de $O({m_N}^2)$ états par processus.

type d'anneaux	problème	temps de service	borne inférieure à espace mémoire	borne supérieure à espace mémoire
unidirectionnel	Circulation	non optimal	$\lg(m_N - 2) - 1$ **[BGJ99]**	$2.\lg(m_N) + 2$ **[Joh04]**
unidirectionnel	Circulation	optimal	$\lg(m_N - 2) - 1$ **[BGJ99]**	$\lg(N + 1) + 2$ **[Joh02]**
unidirectionnel	Election	-	$\lg(m_N - 2) - 1$ **[BGJ99]**	$3.\lg(m_N) + 1$ **[BGJ07]**
bidirectionnel	Circulation	non optimal	?	$2.\lg(m_N) + 12$ **[Joh04]** avec [IJ93]
bidirectionnel	Circulation	optimal	?	$\lg(N + 1) + 11$ **[Joh02]** avec [IJ93]
bidirectionnel	Election	-	?	$3.\lg(m_N) + 10$ **[BGJ07]** avec [IJ93]

FIG. 5.4 – Espace mémoire nécessaire aux algorithmes probabilistes « référants »

Une autre question ouverte est le coût du temps de service optimal. Sur un anneau unidirectionnel, il semble que la contrepartie pour obtenir un temps de service optimale est la taille de l'espace mémoire dans le cas d'anneaux unidirectionnels. Que se passe-t-il dans le cas d'anneaux bidirectionnels ? Est-il possible de construire un algorithme de circulation d'un jeton optimal en temps de service, nécessitant moins de $\lg(N)$ bits d'espace mémoire sur chaque processus ?

Une approche complètement différente est d'utiliser un algorithme déterministe de circulation d'un jeton sur un anneau bidirectionnel semi-uniforme. De tels algorithmes sont proposés dans [Dij74, BD95]. Ces algorithmes nécessite 2 bits d'espace mémoire et le temps de service est $2N$. Pour cela, il nous faut concevoir un algorithme efficace d'élection de leader sur les anneaux bidirectionnels. Par exemple, un algorithme dont le temps de convergence est $O(N^2)$; ou un algorithme nécessitant moins de $3\lg(m_N)$ bits d'espace mémoire.

Qu'en est-il de l'espace mémoire nécessaire à un algorithme probabiliste d'élection d'un leader, sur un anneau bidirectionnel ? Nous avons établi, une borne inférieure proche pour les anneaux unidirectionnels de $(m_N - 2)/2$ états par processus. Je conjecture qu'il est possible d'établir une borne inférieure de l'ordre de $min((m_N - 3)/3, \sqrt{m_N}) - 2$ à l'élection d'un leader sur un anneau. A noter qu'un tel algorithme ne pourra pas être silencieux, car dans [DGS96], Mohamed Gouda, Shlomi Dolev et Marco Schneider ont établi qu'un algorithme silencieux auto-stabilisant d'élection d'un leader a besoin d'au moins $\lg(N)$ d'espace mémoire par processus.

5.3 Algorithmique pour les réseaux Ad-Hoc

La gestion des réseaux Ad-Hoc à grande échelle. C'est à dire des réseaux comportant des milliers voir des millions de nœuds ne peuvent pas être gérés comme des systèmes de petite taille. Lorsqu'il y a peu d'éléments, les perturbations (panne franche, remise en marche, ajout d'éléments, mouvement des nœuds, ...) sont relativement rares. Ainsi, le délai entre deux perturbations est

suffisamment important pour que le système puisse prendre le temps de récupérer toutes ses fonctionnalités avant de fournir un service optimal. Dans ce contexte, les algorithmes auto-stabilisants sont adaptés au besoin.

Dans le cas de système de grande taille, ce paradigme n'est plus acceptable. Le délai moyen entre deux perturbations ne permet pas de retrouver un fonctionnement optimal. La construction de protocole robuste et auto-stabilisant prend tout son intérêt dans ce cadre.

5.3.1 Version robuste du protocole de Johnen et Nguyen, ISPA 2007

Il serait intéressant de proposer une version robuste du protocole **[JN06b]**. Une version robuste qui garantirait les propriétés suivantes :

- Une fois le réseau partitionné en grappes bien formées, il reste partitionné, même durant la reconstruction des grappes. Toutes les applications utilisant la structure en grappes peuvent fonctionner correctement, une fois que les grappes sont bien formées.
- Le temps de convergence du protocole à une configuration où les grappes sont bien formées doit être constant pour toutes les tailles de réseaux et toutes les configurations initiales.

5.3.2 Compilateur auto-stabilisant et robuste

Plus généralement, il serait intéressant de proposer un compilateur auto-stabilisant et robuste qui transformerait tout protocole auto-stabilisant de construction de grappes bien formées en un protocole auto-stabilisant et robuste.

5.3.3 Management auto-stabilisant et robuste des réseaux Ad-Hoc

La tâche principale du management d'un réseau est d'assurer le routage des paquets de l'émetteur au destinataire. Pour assurer un routage, pro-actif, deux éléments sont nécessaires (1) un protocole de construction de table de routages et (2) un mécanisme de transfert des paquets.

L'agrégation est souvent utilisée afin de réduire la quantité d'information échangée en vu de construire les tables de routage [CYC02, KVCP97, KG02, SG02, BR03, KRKV03]. Le routage d'un paquet est effectué en deux temps. Dans un premier temps, le paquet est acheminé à la grappe du destinataire. Finalement, le leader de cette grappe, se charge du routage inter-grappes. Chaque nœud a besoin de garder dans sa table de routage uniquement les routes à destination des leaders de grappe. Ainsi, la taille des tables de routage est réduite et la maintenance des tables est facilitée.

Il serait intéressant de concevoir une structure protocolaire complète qui (1) sera sans perte de service et (2) qui aura les avantages d'un protocole de routage basé sur une agrégation des nœuds. L'objectif est qu'une fois que le service est établi, la continuité du service soit assurée, tout en permettant au système d'évoluer vers un fonctionnement optimal. Concrètement, dès qu'une route existe entre deux nœuds, l'acheminement des paquets entre ces deux nœuds devrait être assuré pour toujours. L'acheminement des paquets devra être garanti même durant la phase où les routes évoluent pour devenir optimales ou bien la phase de reconstruction des grappes pour

obtenir des grappes ayant des propriétés intéressantes (par exemple, des grappes bien formérs de taille bornée).

Dans le cas de grappes bien formées, tous les nœuds doivent maintenir une table de routage et router les paquets. Les tables sont de taille réduite parce qu'elles ne contiennent que les routes aux leaders de grappe. Néanmoins, ce type d'approche nécessite la participation de tous les nœuds du réseau au routage.

Une autre approche est de déterminer des nœuds passerelles (en anglais, bridge) entre les chefs de grappe. L'objectif est que l'ensemble des chefs et des passerelles forment un graphe connecté. Ces nœuds (qualifiés de nœuds actifs) sont donc les seuls à maintenir une table de routage et à participer à l'acheminement des paquets.

Des constructions auto-stabilisantes de nœuds actifs ont été proposées [DLGP05, DFG06]. Améliorer les protocoles existants dans deux axes distincts suivant semble possible :

- Proposer un protocole auto-stabilisant minimisant le nombre de passerelles.
- Proposer un protocole auto-stabilisant et robuste, donc sans perte de service.

Bibliographie

[Abr95] U Abraham. On interprocess communication and the implementation of multi-writer atomic registers. *Theoretical Computer Science*, 149(2) :257–298, 1995.

[AGH90] A Arora, MG Gouda, and T Herman. Composite routing protocols. In *SPDP'90, the 2nd IEEE Symposium on Parallel and Distributed Processing*, pages 70–78, 1990.

[Ang80] D Angluin. Local and global properties in networks of processors. In *STOC80, the 12th Annual ACM Symposium on Theory of Computing*, pages 82–93. ACM, 1980.

[AS99a] G Antonoiu and PK Srimani. Mutual exclusion between neighboring nodes in an arbitrary system graph tree that stabilizes using read/write atomicity. In *Euro-Par'99 Parallel Processing Proceeding, Springer LNCS :1685*, pages 823–830, 1999.

[AS99b] G Antonoiu and PK Srimani. Self-stabilizing protocol for mutual exclusion among neighboring nodes in a tree structured distributed system. *Parallel Algorithms and Applications*, 14(1) :1–18, 1999.

[AW98] H Attiya and JL Welch. *Distributed computing : fundamentals, simulations and advanced topics.* McGraw-Hill, Inc., 1998.

[Bas99] S. Basagni. Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. In *VTC'99, the IEEE 50th International Vehicular Technology Conference*, pages 889–893, 1999.

[BCD95] J Beauquier, S Cordier, and S Delaët. Optimum probabilistic self-stabilization on uniform rings. In *WSS95, the 2nd Workshop on Self-Stabilizing Systems*, pages 15.1–15.15, 1995.

[BD95] J. Beauquier and O. Debas. An optimal self-stabilizing algorithm for mutual exclusion on bidirectional non uniform rings. In *WSS95, the 2nd Workshop on Self-Stabilizing Systems*, pages 17.1–17.13, 1995.

[BE81] D. J. Baker and A. Ephremides. Architectural organization of a mobile radio network via a distributed algorithm. *IEEE Transactions on Communications*, 29(11) :1694–1701, 1981.

[BF03] C. Bettstetter and B Friedrich. Time and message complexities of the generalized distributed mobility-adaptive clustering (GDMAC) algorithm in wireless multihop networks. In *VTC 2003-spring, the 57th IEEE Semiannual Vehicular Technology Conference*, pages 176–180, 2003.

[BGJ99] J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols. In *PODC99, the 18th Annual ACM Symposium on Principles of Distributed Computing*, pages 199–208, 1999.

[BGJ01] J. Beauquier, M. Gradinariu, and C. Johnen. Cross-over composition - enforcement of fairness under unfair adversary. In *WSS01, the 5th International Workshop on Self-Stabilizing Systems, Springer LNCS : 2194*, pages 19–34, 2001.

[BGJ07] J. Beauquier, M. Gardinariu, and C. Johnen. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. *Distributed Computing*, 20(1) :75–93, 2007.

[BGM93] JE Burns, MG Gouda, and RE Miller. Stabilization and pseudo-stabilization. *Distributed Computing*, 7(1) :35–42, 1993.

[BJ02] J. Beauquier and C. Johnen. Analyze of randomized self-stabilizing algorithms under non-deterministic scheduler classes. Technical Report 1327, L.R.I, July 2002.

[BJM06] J. Beauquier, C. Johnen, and S. Messika. All k-bounded policies are equivalent for self-stabilization. In *SSS'06, the 8th International Symposium on Stabilization, Safety, and Security of Distributed Systems, Springer LNCS : 4280*, pages 82–94, 2006.

[BK01] S. Bannerjee and S. Khuller. A clustering scheme for hierarchical control in wireless networks. In *INFOCOM'01, the 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1028–1037, 2001.

[BKL01] P Basu, N Khan, and T D C Little. A mobility based metric for clustering in mobile ad hoc networks. In *ICDCSW'01, Workhops of the 21st International Conference on Distributed Computing Systems*, page 413, 2001.

[BP89] JE Burns and J Pachl. Uniform self-stabilizing rings. *ACM Transactions on Programming Languages and Systems*, 11(2) :330–344, 1989.

[BPV04] C Boulinier, F Petit, and V Villain. When graph theory helps self-stabilization. In *PODC04, the 23th Annual ACM symposium on Principles of distributed computing*, pages 150–159. ACM Press, 2004.

[BR03] E. M. Belding-Royer. Multi-level hierarchies for scalable ad hoc routing. *Wireless Networks*, 9(5) :461–478, 2003.

[CDT02] M. Chatterjee, S. Das, and D. Turgut. WCA : A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing*, 5(2) :193–204, 2002.

[CFS96] D Coppersmith, U Feige, and JB Shearer. Random walks on regular and irregular graphs. *SIAM Journal on Discrete Mathematics*, 9(2) :301–308, 1996.

[CG01] JA Cobb and MG Gouda. Stabilization of routing in directed networks. In *WSS01, the 5th International Workshop on Self-Stabilizing Systems, Springer LNCS : 2194*, pages 51–66, 2001.

[CG02] J. A. Cobb and M. G. Gouda. Stabilization of general loop-free routing. *Journal of Parallel and Distributed Computing*, 62(5) :922–944, 2002.

[CS96] M Choy and AK Singh. Localizing failures in distributed synchronization. *IEEE Transactions on Parall ans Distributed Systems*, 7(7) :705–716, 1996.

[CYC02] G-H Chen C-Y Chiu, E H-K Wu. Stability aware cluster routing protocol for mobile ad-hoc networks. In *ICPADS'02, 9th International Conference on Parallel and Distributed Systems*, page 471, 2002.

[DFG06] V. Drabkin, R. Friedman, and M. Gradinariu. Self-stabilizing wireless connected overlays. In *OPODIS06, the 10th International Conference On Principles Of DIstributed Systems, Springer LNCS :4305*, pages 425–439, 2006.

[DGS96] S Dolev, MG Gouda, and M Schneider. Memory requirements for silent stabilization. In *PODC96, the 15th Annual ACM Symposium on Principles of Distributed Computing*, pages 27–34, 1996.

[DGT00] AK Datta, M Gradinariu, and S Tixeuil. Self-stabilizing mutual exclusion using unfair distributed scheduler. In *IPDPS00, the 14th IEEE International Parallel and Distributed Processing Symposium*, pages 465–470, 2000.

[DGT04] AK Datta, M Gradinariu, and S Tixeuil. Self-stabilizing mutual exclusion with arbitrary scheduler. *The Computer Journal*, 47(3) :289–298, 2004.

[DH95a] S Dolev and T Herman. Superstabilizing protocols for dynamic distributed systems. In *WSS95, the 2nd Workshop on Self-Stabilizing Systems*, pages 3.1–3.15, 1995.

[DH95b] S Dolev and T Herman. Superstabilizing protocols for dynamic distributed systems. In *PODC95, the 14th Annual ACM Symposium on Principles of Distributed Computing*, page 255, 1995.

[DH97] S Dolev and T Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago Journal of Theoretical Computer Science*, 3(4), 1997.

[DH01] S Dolev and T Herman. Dijkstra's self-stabilizing algorithm in unsupportive environments. In *WSS01, the 5th International Workshop on Self-Stabilizing Systems, Springer LNCS : 2194*, pages 67–81, 2001.

[DHT04] P Duchon, N Hanusse, and S Tixeuil. Optimal randomized self-stabilizing mutual exclusion on synchronous rings. In *DISC04, the 18th International Conference on Distributed Computing, Springer LNCS 3274*, pages 216–229. Springer, 2004.

[Dij74] EW Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17 :643–644, 1974.

[DIM93] S Dolev, A Israeli, and S Moran. Self-stabilization of dynamic systems assuming only Read/Write atomicity. *Distributed Computing*, 7(1) :3–16, 1993.

[DIM97] S Dolev, A Israeli, and S Moran. Uniform dynamic self-stabilizing leader election. *IEEE Transactions on Parall ans Distributed Systems*, 8(4) :424–440, 1997.

[DL00] J. Durand-Lose. Randomized uniform self-stabilizing mutual exclusion. *Information Processing Letters*, 74(5-6) :203–207, 2000.

[DLGP05] A. K. Datta, P. Linga, M. Gradinariu, and P. Raipin Parvédy. Self distributed query region covering in sensor networks. In *SRDS05, the 24th IEEE Symposium on Reliable Distributed Systems*, pages 50–59, 2005.

[Dol97] S Dolev. Self-stabilizing routing and related protocols. *Journal of Parallel and Distributed Computing*, 42(2) :122–127, 1997.

[Dol00] S. Dolev. *Self-Stabilization*. The MIT Press, 2000.

[DT01] B Ducourthial and S Tixeuil. Self-stabilization with r-operators. *Distributed Computing*, 14(3) :147–162, 2001.

[DT06] S. Dolev and N. Tzachar. Empire of colonies : Self-stabilizing and self-organizing distributed algorithms. In *OPODIS06, the 10th International Conference On Principles Of DIstributed Systems, Springer LNCS :4305*, pages 230–243, 2006.

[EWB87] A. Ephremides, J.E. Wieselthier, and D.J. Baker. A design concept for reliable mobile radio networks with frequency-hopping signaling. *IEEE Transactions on Wireless communications*, pages 56–73, 1987.

[FJ01] F.E. Fich and C. Johnen. A space optimal, deterministic, self-stabilizing, leader election algorithm for unidirectional rings. In *DISC01, the 15th International Symposium on Distributed Computing, Springer LNCS :2180*, pages 224–239, 2001.

[GCK07] R. Guerraoui G. Chockler and I. Keirar. Amnesic distributed storage. In *DISC'07, the 21th International Symposium on Distributed Computing, Springer LNCS :4731*, pages 139–151, 2007.

[GH96] MG Gouda and FF Haddix. The stabilizing token ring in three bits. *Journal of Parallel and Distributed Computing*, 35(1) :43–48, 1996.

[GH97] MG Gouda and FF Haddix. The linear alternator. In *WSS97, the 3rd Workshop on Self-Stabilizing Systems*, pages 31–47. Carleton University Press, 1997.

[GH99] MG Gouda and F Haddix. The alternator. In *WSS99, the 4th Workshop on Self-Stabilizing Systems*, pages 48–53. IEEE Computer Society, 1999.

[GLA93] J. J. Garcia-Luna-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Transactions Networking*, 1(1) :130–141, 1993.

[GS99] M. G. Gouda and M. Schneider. Stabilization of maximal metric trees. In *WSS99, the 4th Workshop on Self-Stabilizing Systems*, pages 10–17, 1999.

[GT95] M Gerla and J T-C Tsai. Multicluster, mobile, multimedia radio network. *Journal of Wireless Networks*, 1(3) :255–265, 1995.

[HC01] Shing-Tsaan Huang and Bau-Wen Chen. Optimal 1-fair alternators. *Information Processing Letters*, 80(3) :159–163, 2001.

[Her90] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35(2) :63–67, 1990.

[Her92] T Herman. Self-stabilization : randomness to reduce space. *Distributed Computing*, 6(2) :95–98, 1992.

[HG05] FF Haddix and MG Gouda. A general alternator. In *PDCS05, the 17th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 409–413, 2005.

[HHH03] S-T Huang, Y-S Huang, and S-S Hung. Alternators on uniform rings of odd size. *Distributed Computing*, 16(4) :263–268, 2003.

[HJ06a] L. Higham and C Johnen. Relationships between communication models in networks using atomic registers. In *IPDPS'06, the 20th IEEE International Parallel & Distributed Processing Symposium*, 2006.

[HJ06b] L. Higham and C. Johnen. Relationships between communication register models in networks. Technical Report 1419, L.R.I, 2006.

[HLMI03] M Herlihy, V Luchangco, M Moir, and WN Scherer III. Software transactional memory for dynamic-sized data structures. In *PODC03, the 22th Annual ACM symposium on Principles of distributed computing*, pages 92–101, 2003.

[Hoe98] JH Hoepman. Self-stabilizing ring orientation using constant space. *Information and Control*, 144(1) :18–39, 1998.

[HP05] FF Haddix and W Peng. A uniform general alternator. In *CIIT05, The 4th IASTED International Conference on Communications, Internet, and Information Technology*, 2005.

[HPT02] JH Hoepman, M Papatriantafilou, and P Tsigas. Self-stabilization of wait-free shared memory objects. *Journal of Parallel and Distributed Computing*, 62(5) :818–842, 2002.

[Hua00] ST Huang. The fuzzy philosophers. In *Parallel and Distributed Processing (IPDPS Workshops 2000), Springer LNCS :1800*, pages 130–136, 2000.

[HV95] S Haldar and K Vidyasankar. Constructing 1-writer multireader multivalued atomic variables from regular variables. *Journal of the Association of the Computing Machinery*, 42(1) :186–203, 1995.

[HW90] MP Herlihy and J.M. Wing. Linearizability : a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 12(3) :463–492, 1990.

[IJ90a] A Israeli and M Jalfon. Self-stabilizing ring orientation. In *WDAG90, the Distributed Algorithms 4th International Workshop Proceedings, Springer LNCS :486*, pages 1–14, 1990.

[IJ90b] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *PODC90, the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 119–131, 1990.

[IJ93] A Israeli and M Jalfon. Uniform self-stabilizing ring orientation. *Information and Computation*, 104 :175–196, 1993.

[ILS95] G. Itkis, C. Lin, and J. Simon. Deterministic, constant space, self-stabilizing leader election on uniform rings. In *WDAG95, the Distributed Algorithms 9th International Workshop, Springer LNCS :972*, pages 288–302, 1995.

[JN06a] C. Johnen and L. H. Nguyen. Clustering algorithm for ad hoc networks. In *AlgoSensors'2006, Second International Workshop on Algorithmic Aspects of Wireless Sensor Networks, Springer LNCS :4240*, pages 83–94, 2006.

[JN06b] C. Johnen and L. H. Nguyen. Robust self-stabilizing clustering algorithm. In *OPODIS06, the 10th International Conference On Principles Of DIstributed Systems, Springer LNCS :4305*, pages 408–422, 2006.

[JN07] C. Johnen and L. H. Nguyen. Self-stabilizing bounded size clustering algorithm. In *ISPA07, The 5th International Symposium on Parallel and Distributed Processing and Applications*, 2007.

[Joh02] C. Johnen. Service time optimal self-stabilizing token circulation protocol on anonymous unidirectional. In *SRDS 2002, the 21st Symposium on Reliable Distributed Systems*, pages 80–89. IEEE Computer Society Press, 2002.

[Joh04] C. Johnen. Bounded service time and memory space optimal self-stabilizing token circulation protocol on unidirectional rings. In *IPDPS'04, the 18th IEEE International Parallel & Distributed Processing Symposium*, 2004.

[JT03] C. Johnen and S. Tixeuil. Route preserving stabilization. In *SSS03, the 6th Symposium on Self-Stabilizing Systems, Springer LNCS : 2704*, pages 183–197, 2003.

[Kar05] MH Karaata. An optimal self-stabilizing strarvation-free alternator. *Journal of Computer and System Sciences*, 71(4) :480–494, 2005.

[KG02] Taek Jin Kwon and Mario Gerla. Efficient flooding with passive clustering (pc) in ad hoc networks. *ACM SIGCOMM Computer Communication Review*, 32(1) :44–56, 2002.

[KRKV03] S Kuppa, M Ramakrishnan, S Krishnamurthy, and S. Venkatesan. Brief announcement : cluster-based control mechanism for communication networks. In *PODC03, the 22th Annual ACM symposium on Principles of distributed computing*, page 83, 2003.

[KVCP97] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan. A cluster-based approach for routing in dynamic networks. *ACM SIGCOMM Computer Communication Review*, 27(2) :49–64, 1997.

[KY97] H. Kakugawa and M. Yamashita. Uniform and self-stabilizing token rings allowing unfair daemon. *IEEE Transactions on Parall ans Distributed Systems*, 8(2) :154–162, 1997.

[KY02a] H Kakugawa and M Yamashita. Self-stabilizing local mutual exclusion on networks on which process identifiers are not distinct. In *SRDS02, the 21th IEEE Symposium on Reliable Distributed Systems*, pages 202–211, 2002.

[KY02b] H. Kakugawa and M. Yamashita. Uniform and self-stabilizing fair mutual exclusion on unidirectional rings under unfair distributed daemon. *Journal of Parallel and Distributed Computing*, 62(5) :885–898, 2002.

[Lam74] L. Lamport. A new solution of dijkstra's concurrent programming problem. *Communications of the Association of the Computing Machinery*, 17(8) :453–455, 1974.

[Lam85] L. Lamport. The mutual exclusion problem–part i : A theory of interprocess communication, part ii : Statement and solutions. *Communications of the Association of the Computing Machinery*, 33(2) :453–455, 1985.

[Lam86] L Lamport. On interprocess communication. *Distributed Computing*, 1(2) :77–101, 1986.

[LL05] TJ Liu and CL Lee. 2-state alternator for uniform rings with arbitrary size. In *AINA05, the IEEE 19th International Conference on Advanced Information Networking and Applications*, pages 847– 852, 2005.

[Lov96] L. Lovász. *Random Walks on Graphs : A Survey*, chapter 1, pages 353–398. János Bolyai Mathematical Society, d. miklós, v. t. sós, t. szőnyi edition, 1996.

[LS92] C Lin and J Simon. Observing self-stabilization. In *PODC92, the 11th Annual ACM Symposium on Principles of Distributed Computing*, pages 113–123, 1992.

[LTV96] M Li, J Tromp, and PMB Vitanyi. How to share concurrent wait-free variables. *Journal of the Association of the Computing Machinery*, 43(4) :723–746, 1996.

[LW06] T-J Liu and L-C Wuu. Randomized three-state alternator for uniform rings. *Journal of Parallel and Distributed Computing*, 66(10) :1347–1351, 2006.

[LYT06] HS Lin, CB Yang, and KT Tseng. 1-fair alternator designs for the de bruijn network. In *PDCAT06, the 7th International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2006.

[MBF04] N Mitton, A Busson, and E Fleury. Self-organization in large scale ad hoc networks. In *MED-HOC-NET'04, The 3rd Annual Mediterranean Ad Hoc Networking Workshop*, 2004.

[MGM03] P. Kristiansen M. Gairing, S. T. Hedetniemi and A A. McRae. Self-stabilizing algorithms for k-domination. In *SSS03, the 6th Symposium on Self-Stabilizing Systems, Springer LNCS : 2704*, pages 49–60, 2003.

[MN98] M Mizuno and M Nesterenko. A transformation of self-stabilizing serial model programs for asynchronous parallel computing environments. *Information Processing Letters*, 66(6) :285–290, 1998.

[MOOY92] A Mayer, Y Ofek, R Ostrovsky, and M Yung. Self-stabilizing symmetry breaking in constant-space. In *STOC92, the 24th Annual ACM Symposium on Theory of Computing*, pages 667–678, 1992.

[MOOY02] A Mayer, R Ostrovsky, Y Ofek, and M Yung. Self-stabilizing symmetry breaking in constant-space. *SIAM Journal on Computing*, 31(5) :1571–1595, 2002.

[NA02a] M Nesterenko and A Arora. Local tolerance to unbounded byzantine faults. In *SRDS02, the 21th IEEE Symposium on Reliable Distributed Systems*, 2002.

[NA02b] M Nesterenko and A Arora. Stabilization-preserving atomicity refinement. *Journal of Parallel and Distributed Computing*, 62(5) :766–791, 2002.

[Ros00] L. Rosaz. Self-stabilizing token circulation on asynchronous uniform unidirectional rings. In *PODC00, the 19th Annual ACM Symposium on Principles of Distributed Computing*, pages 249–258, 2000.

[SCP03] AK Datta S Cantarell and F Petit. Self-stabilizing atomicity refinement allowing neighborhood concurrency. In *SSS03, the 6th Symposium on Self-Stabilizing Systems, Springer LNCS : 2704*, pages 102–112, 2003.

[SG02] S Srivastava and R. K. Ghosh. Cluster based routing using a k-tree core backbone for mobile ad hoc networks. In *DIALM '02, the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, 2002.

[SPS00] P A G Sivilotti, S M Pike, and N Sridhar. A new distributed resource-allocation algorithm with optimal failure locality. In *PDCS00, the 12th IASTED International Conference on Parallel and Distributed Computing and Systems*, volume 2, pages 524–529, 2000.

[TB94] Y-K Tsay and R Bagrodia. An algorithm with optimal failure locality for the dining philosophers problem. In *WDAG94, the Distributed Algorithms 8th International Workshop, Springer LNCS :857*, pages 296–310. Springer-Verlag, 1994.

[TV05] F. Theoleyre and F. Valois. About the self-stabilization of a virtual topology for self-organization in ad hoc networks. In *SSS05, the 7th Symposium on Self-Stabilizing Systems, Springer LNCS : 3764*, pages 214–228, 2005.

[TW91] P Tetali and P Winkler. On a random walk arising in self-stabilizing token management. In *PODC91, the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 273–280, 1991.

[VET07] G. Venkataraman, S. Emmanuel, and S. Thambipillai. Size-restricted cluster formation and cluster maintenance technique for mobile ad hoc networks. *International Journal of Network Management*, 17(2) :171–194, 2007.

[VJ00] G. Varghese and M. Jayaram. The fault span of crash failures. *Journal of the Association of the Computing Machinery*, 47(2) :244–293, 2000.

[XHGS03] Z Xu, S T Hedetniemi, W Goddard, and P K Srimani. A synchronous self-stabilizing minimal domination protocol in an arbitrary network graph. In *IWDC 2003, the 5th International Workshop on Distributed Computing, LNCS 2918*, pages 26–32, 2003.

[YF04a] O. Younis and S. Fahmy. Distributed clustering in ad-hoc sensor networks : A hybrid, energy-efficient approach. In *INFOCOM'04, the 23th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 629–640, 2004.

[YF04b] O. Younis and S. Fahmy. Heed : A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Transactions on mobile computing*, 3(4) :366–379, 2004.

Index

Table des matières