Auto-stabilisation et Protocoles Réseau

Colette Johnen* Franck Petit** Sébastien Tixeuil*

* Laboratoire de Recherche en Informatique, (UMR CNRS 8623)
Université Paris Sud-XI, 91405 Orsay cedex, France
{colette, tixeuil}@lri.fr

** Laboratoire de Recherche en Informatique d'Amiens,
Université de Picardie Jules Verne, Amiens, France
petit@laria.u-picardie.fr

Résume Dans [Dij74], E.W. Dijkstra définit l'auto-stabilisation comme la propriété pour un système de retrouver de lui-même un comportement correct en un nombre fini d'étapes, et ce, quel que soit son état initial. L'auto-stabilisation constitue donc un moyen simple et efficace de tolérer des fautes ou des pannes transitoires. Cet article est consacré aux travaux proposant des solutions auto-stabilisantes à des problèmes classiques dans le domaine des réseaux informatiques, notamment dans les domaines du routage, des communications point-à-point (transport de données) et des protocoles de contrôle. Des techniques de transformations automatiques de protocoles en leur équivalents auto-stabilisants sont également présentées, ainsi que des améliorations portant sur le temps de stabilisation lorsque le nombre de défaillances est faible.

Mots clés: Auto-stabilisation, réseaux, protocoles, routage, communications point-à-point, contrôle réparti, transformation automatique.

Abstract In [Dij74], E.W. Dijkstra defined self-stabiliation as the property for a distributed system to recover by itself a correct behavior in a finite number of steps, starting from any initial state. Thus, self-stabilization is a simple and efficient way to tolerate transient faults or failures. This paper surveys works that propose self-stabilizing solutions to problems arising in Computer Networks, such as routing and transport layers, or network control protocols. We also review automatic transformers that turn protocols into their self-stabilizing counterparts, and self-stabilization refinements that provide fast stabilization time when the number of faults that hit the network is small.

Keywords: Self-stabilization, networks, protocols, routing, transport, distributed control, automatic transformers

1 Introduction

Les réseaux d'ordinateurs constituent un champ d'application naturel des algorithmes répartis. Un algorithme (ou protocole) réparti est un algorithme destiné à un ensemble de processeurs (ou processus, ordinateurs, nœuds) interconnectés entre eux et coopérant pour exécuter une tâche. La complexité en temps, en nombre de messages et en espace mémoire sont des mesures de performances fondamentales dans le domaine des systèmes répartis. La tolérance aux fautes ou aux pannes est une autre mesure d'efficacité des protocoles. La meilleure façon de tolérer des fautes transitoires est de produire des protocoles auto-stabilisants. Le concept d'auto-stabilisation dans les systèmes répartis a été introduit par Dijkstra en 1974 [Dij74] (voir également [Dol00, Her02]). Un système réparti est auto-stabilisant s'il retourne à un état légitime en un nombre fini d'étapes, indépendamment de son état initial, et que le système reste dans un état légitime jusqu'à ce qu'une nouvelle défaillance survienne (Figure 1). Par conséquent, un protocole auto-stabilisant tolère des défaillances transitoires de processeurs et de liens Lamport [Lam83] l'a signalé à la communauté des "systèmes répartis"dix ans après le séminal papier de Dijkstra. Ces défaillances incluent les corruptions de variables, les corruptions du pointeur de programme (qui mènent un processeur à exécuter temporairement n'importe quelle partie de son code) et les corruptions des canaux de communication.

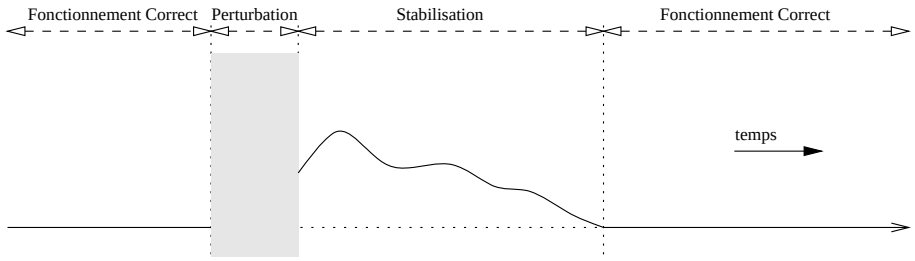

Figure 1: L'auto-stabilisation selon Dijkstra

Dans le contexte spécifique des réseaux d'ordinateurs qui sont des systèmes répartis, retrouver un comportement correct après l'occurrence d'une défaillance peut être très coûteux. Le réseau tout entier devra - peut être - s'arrêter puis se réinitialiser dans un état initial correct [Per00]. Si cette approche est possible dans le cas de petits réseaux, elle est impraticable dans les réseaux de grande taille comme Internet. L'auto-stabilisation propose une manière de retrouver un comportement correct après une faute, sans le coût et les inconvénients d'une intervention humaine généralisée. En effet, après qu'une faute est diagnostiquée, il suffit d'enlever, de réparer, ou de réinitialiser le ou les éléments défaillants et le système, de lui-même, retournera à un état global correct en un temps raisonnable.

De manière plus spécifique, [VJ00] montre que si les processeurs d'un système peuvent tomber en panne puis redémarrer dans l'état où ils se trouvaient avant la panne, alors le système peut atteindre un état global où l'état de chaque processeur est arbitraire. Si de plus les liens de communication ne sont pas fiables, alors c'est le système entier qui peut atteindre un état global incorrect, c'est à dire que les processeurs *et* les canaux de communication se trouvent dans un état arbitraire. En partant d'un tel état, seul un protocole auto-stabilisant est capable de ramener le système dans un état légitime à partir duquel son comportement est correct.

La propriété d'auto-stabilisation est également en rapport avec la propriété de dynamicité des protocoles réseau. Un protocole est dit *dynamique* si son code n'a pas besoin d'être modifié en cas d'ajout ou de retrait de nœuds ou d'arcs dans le réseau. Alors, le réseau tolère des changements de topologie pendant son fonctionnement. Il convient de remarquer que tous les protocoles auto-stabilisants dont la complexité en nombre d'états par processeur ne dépend pas d'un paramètre global sont dynamiques [DIM93]. Avec ce type de système, lorsqu'un nouveau canal de communication est relié à un routeur, la taille de la mémoire du routeur n'a pas à être modifiée, ni le code changé, pour s'adapter à la nouvelle topologie.

Bien que n'ayant pas été conçus dès l'origine dans ce but, plusieurs protocoles utilisés dans Internet sont en fait auto-stabilisants. Par exemple, les protocoles *Netchange, RIP* et *OSPF* sont auto-stabilisants [Taj77].

Le reste de l'article peut être décrit comme suit : dans la Section 2, nous présentons deux modèles de communication courants dans le domaine de l'auto-stabilisation, les registres et les échanges de messages. Par la suite, nous présentons successivement les résultats ayant trait au routage (Section 3), aux communications point à point (Section 4), et aux protocoles de contrôle dans les réseaux (Section 5). Dans la Section 6, nous discutons de la possibilité d'ajouter automatiquement la propriété d'auto-stabilisation à un protocole ne l'étant pas. Ensuite, nous présentons des améliorations pour accélérer le temps de convergence lorsque le nombre de défaillances est faible (Section 7). Nous concluons cet article à la Section 8.

2 Modèles de communication

Le fait que les processeurs communiquent par *échanges de messages* et de manière *asynchrone* sont des hypothèses souvent implicites dans le domaine des réseaux. L'asynchronisme vient du fait que le temps d'exécution d'un pas de programme d'un processeur n'est en général pas connu, pas plus que le délai d'acheminement d'un message d'un point à un autre du réseau, y compris entre deux voisins.

Or, la plupart des travaux du domaine de l'auto-stabilisation s'effectue à partir de modèles dans lesquels les problèmes engendrés par les échanges de messages sont *a priori* ignorés. Les processeurs sont alors supposés communiquer par l'intermédiaire de *registres*, mémoire partagée entre processeurs voisins dans le réseau d'interconnexion. Dans le prochain paragraphe, nous développons le modèle à registres et ses variantes. Ensuite, nous exposons différents résultats obtenus dans le modèle à passage de messages, lesquels, suivant les hypothèses de travail retenues, permettent raisonnablement d'envisager l'implantation de communications auto-stabilisantes.

2.1 Communication par registres

Dans son article séminal [Dij74], Dijkstra utilise un modèle de communication par registres. Dans ce modèle, un processeur est capable de lire en une étape atomique la valeur des variables de tous ses voisins. Si ce modèle présente l'avantage de la simplicité dans l'expression des protocoles, il implique implicitement des coûts de communication importants.

Par la suite, le modèle de communication par mémoire partagée a été amélioré en un modèle de communication par registres [DIM93] où chaque lecture et chaque écriture est atomique. Dans ce modèle, il n'est plus possible d'avoir une vision instantanée de

son voisinage car entre le moment où on lit la valeur du registre d'un premier voisin et le moment où on lit la valeur du registre d'un deuxième voisin, il est possible que le premier voisin ait déjà changé la valeur de son registre.

2.2 Communication par échange de messages

Considérons un protocole de communication auto-stabilisant P conçu pour un canal bidirectionnel (*full-duplex*) reliant un émetteur à un récepteur. Dans [GM91], il est montré que si une borne majorant la taille du canal n'est pas connue (c'est à dire que la taille du canal est finie mais non bornée), alors le nombre de configurations légitimes de P est infini. De plus, toute exécution de P doit contenir une infinité de configurations légitimes distinctes [DIM97a]. Evidemment, ces résultats théoriques semblent *a priori* constituer un obstacle majeur au développement des protocoles auto-stabilisants dans la "pratique"car la mémoire requise par ces protocoles serait potentiellement infinie. Cependant, des solutions peuvent être apportées en combinant entre-elles les hypothèses suivantes : *(1)* la taille des canaux a une borne connue, *(2)* les canaux sont fiables, *(3)* il est envisageable d'implanter une protocole probabiliste.

Un protocole est dit *message driven* si un processeur effectue des opérations (envoyer des messages, changer son état) uniquement lorsqu'il reçoit un message. Un protocole auto-stabilisant qui est *message driven* nécessite au moins une action déclenchée par un autre mécanisme, généralement un *timeout*. Sinon le système est totalement bloqué, dans le cas où tous les canaux sont vides.

3 Routage

3.1 Routage *store-and-forward*

Dans la couche réseau d'Internet, les protocoles de routage utilisés sont du type *store-and-forward*. Les paquets arrivant à un routeur sont stockés dans une file d'attente et ensuite traités. Un routeur décide à qui transmettre un paquet en transit en fonction du destinataire du parquet et aussi de sa table de routage.

Si la plupart des protocoles de routage auto-stabilisants font l'hypothèse de canaux bidirectionnels, il peut arriver que seuls des canaux unidirectionels soient disponibles; par exemple dans le cas de réseau *ad hoc*, où les communications se font par ondes radio. Ainsi, un élément du réseau peut recevoir des messages d'un autre membre sans être capable de lui répondre directement. Dans [CG01, DT01], plusieurs protocoles auto-stabilisants de routage sur des réseaux unidirectionnels sont présentés.

Par ailleurs, plusieurs problèmes connexes au routage possèdent des solutions auto-stabilisantes : le problème de la construction de topologie l'objectif est que tous les routeurs connaissent la topologie courante du réseau [Mas95, AACD$^+$96, DH97] et le problème du recensement, le but étant que chaque routeur connaisse l'ensemble des routeurs du réseau [BDT99, DT02]. Le protocole proposé dans [AACD$^+$96] est particulièrement adapté aux réseaux à haut débit.

Dans les sections suivantes, nous détaillons les résultats obtenus dans le cadre général du routage dans les réseaux à liens bidirectionnels.

3.1.1 Métriques

La tâche principale d'un protocole de routage est de maintenir une table de routage sur chaque routeur qui assure en plus des propriétés de base, assurer l'acheminement des paquets au destinataire, une propriété spécifique aux chemins pris par les paquets. Cette propriété s'énonce généralement comme *maximiser une métrique*. Par exemple, les protocoles de routage tels que OSPF and RIP génèrent des tables de routage qui garantissent que le chemin entre deux nœuds quelconques du réseau est le plus court possible. Lorsque la topologie du réseau change les tables de routage sont automatiquement reconstruites de sorte que les chemins obtenus sont de nouveau optimaux pour la métrique considérée. [Dol97] présente un protocole de routage auto-stabilisant basé sur la métrique *minimiser le nombre de routeurs dans les chemins*. Cette métrique est similaire à la métrique du *plus court chemin* dans le cas où tous les canaux sont de même coût.

Dans [GGP97], un protocole auto-stabilisant pour la métrique du *débit maximal* sur des graphes acycliques est présenté. Il s'agit de maximiser le débit maximum de chaque chemin, sachant que la valeur associée à chaque arc est le débit du canal de communication. Le débit d'un chemin est donc le plus petit débit des arcs qui le composent.

Dans [GS99] est présenté un protocole auto-stabilisant générique de construction de tables de routage dans des réseaux où chaque processeur a un identifiant distinct. Par exemple, il peut construire et maintenir les tables de routage pour la métrique du plus court chemin, du débit maximal. Le protocole de [GGP97] utilisent des numéros de session non bornés et donc une mémoire infinie.

3.1.2 Garantie du service

Dans [AGH90] sont présentés deux protocoles de routage auto-stabilisants basés sur la métrique du plus court chemin. Un de ces protocoles est *tolérant au changement de coût associé aux canaux*, car il n'introduit pas de cycle dans les chemins de routage lorsque le coût des canaux change. Supposons qu'il existe un chemin entre les nœuds p et q. Après une ou des modifications du coût sur un ou plusieurs canaux, le protocole modifie les tables de routage pour construire un nouveau chemin de coût minimal entre p et q. Durant cette phase de construction, le protocole garantit qu'il existe toujours un chemin entre p et q; ainsi même pendant la phase de reconstruction, des paquets peuvent être échangés entre p et q.

Les protocoles auto-stabilisants présentés dans [GS95, CW97] construisent et maintiennent les tables de routage pour la métrique du *débit maximal* des graphes de topologie quelconque. Ces protocoles sont tolérants aux changements de débits des canaux. Dans [GS95], chaque routeur doit connaître une borne supérieure sur la taille du réseau. Les protocoles [GS95, CW97] nécessitent aussi des numéros de session non bornés.

Dans [CG02] est présenté un protocole auto-stabilisant générique de construction de tables de routage dans des réseaux; ce dernier protocole est tolérant au changement de "valeur" des canaux pour cela chaque routeur doit connaître une borne supérieure sur la taille du réseau.

3.1.3 Métriques composées

Pour une destination, plusieurs métriques peuvent simultanément exister : une qui minimise le coût du chemin (*shortest path*), une qui maximise le débit du chemin (*maximum*

flow), une qui sélection le chemin ayant le plus petit taux de perte de paquets. La métrique associée à IGRP est la composition de quatre métriques: minimiser le coût, maximiser le débit, minimiser la charge des canaux et minimiser le taux de perte des paquets. Malheureusement, il a été prouvé [GS98] qu'il n'existe pas de protocole auto-stabilisants qui maintiennent des chemins ayant cette métrique.

3.1.4 Routages Hiérarchiques

Le routage hiérarchique permet de réduire grandement la taille des table de routage et donc le temps nécessaire pour routeur les paquets. Une bonne métrique pour ce type de routage consiste à trouver un compromis entre la taille de la table de routage sur chaque routeur et la distance par rapport à un routage optimal pour une métrique particulière. Un protocole auto-stabilisant de routage hiérarchique pour la métrique des plus courts chemins est présenté dans [DDLT00]. Dans le cas particulier du routage inter-domaine dans Internet, une solution auto-stabilisante a été proposée dans [CDT02].

3.2 Routage *wormhole*

Dans le routage *wormhole*, les messages sont découpés en unités de contrôle de flux (ou *flits*), chaque *flit* n'exédant par quelques octets. Toute l'information de routage et de contrôle des messages est stockée dans le premier *flit* (*header flit*). Quand le premier *flit* se déplace à travers le réseau vers sa destination, il réserve le canal traversé pour permettre le passage ultérieur des *flits* de données (*data flits*). Les autres *flits* du messages suivent le *header flit* à la manière d'un pipeline. Quand le dernier *flit* (*tail flit*) traverse un canal, la réservation du canal pour ce message est libérée. Si un *header flit* atteint un processeur ne possédant aucun canal de sortie disponible, les autres *flits* du message restent où ils sont jusqu'à ce que le *header flit* avance de nouveau.

Dans [DGKT02], un protocole auto-stabilisant de routage *wormhole* est proposé. Ce protocole permet la communication entre deux ordinateurs d'un réseau, et peut à ce titre être utilisé par un protocole auto-stabilisant de plus haut niveau. En particulier, ce protocole tolère que des *flits* arbitraires, sans l'organisation *header flit*, puis *data flits*, puis *tail flit*, soient initialement présents dans le réseau suite à une défaillance transitoire.

3.3 Routage *cut-through*

Le routage *cut-through* est utilisé dans de nombreux réseaux organisés en anneau parmi lesquels les réseaux *Token Ring* et *FDDI*. Dans ce schéma de routage, un processeur peut commencer à transférer une partie d'un message à un autre processeur situé sur le chemin vers la destination avant d'avoir reçu le message entièrement. Si ce message est le seul trafic sur le chemin vers la destination, le délai total du message est égal à la somme du temps de transmission calculé sur le lien le plus lent du chemin et du délai de propagation. Par conséquent, il est proportionnel à la longueur du message et au nombre de liens sur le chemin vers la destination. Cette approche supprime le besoin d'avoir une mémoire locale à chaque processeur de taille supérieure à quelques bits, et réduit également le délai total de chaque message à une petite valeur bornée par la taille des tampons de transfert des processeurs.

En l'état actuel, le temps nécessaire pour recevoir/émettre des *bits* depuisvers un médium de communication est beaucoup plus élevé que le temps nécessaire à l'exécution d'opérations élémentaires (calculs sur des entiers, tests, écriture et lecture de registres,

etc.). Aussi les travaux en rapport font l'hypothèse qu'un processeur donné est capable d'exécuter un nombre limité de pas de calcul entre les réceptions de deux parties d'un message. Les travaux menés dans le cadre de l'auto-stabilisation dans un modèle *cut-through* sont doubles :

1. D'une part, [TB96] et [CV99] partent d'un protocole existant dans le modèle *cut-through* (le *token ring* pour [TB96] et le *FDDI* pour [CV99]) et y ajoutent des techniques bien connues issues de l'auto-stabilisation pour obtenir un protocole auto-stabilisant et résolvant le même problème. Ces deux articles se concentrent sur l'étude du surcoût engendré par l'ajout de la propriété d'auto-stabilisation.

2. D'autre part, [BDT99] part d'un protocole facilement auto-stabilisant (la quantité d'informations véhiculée par chaque message est importante) et montre comment plusieurs techniques non-triviales permettent d'utiliser efficacement le modèle *cut-through* pour réduire l'espace mémoire et le temps de stabilisation de ce protocole d'un facteur de l'ordre de la taille du réseau.

4 Communications Point à Point

La communication fiable d'un point à l'autre du réseau consiste à délivrer en un temps fini une série de paquets entre deux nœuds distants dans l'ordre d'émission, sans perte, ni duplication.

4.1 Protocole du Bit Alterné

En supposant des canaux finis mais dont on ne connaît pas la borne sur la taille, et dans le cas où des messages peuvent être perdus, des versions du protocole du *bit alterné* sont proposés dans [AB93]. Cependant, la taille de la mémoire des processeurs est là encore infinie ou le protocole est probabiliste. En supposant que les canaux de communication sont de taille bornée et *quasi-fiable*, une nouvelle version auto-stabilisante du bit alterné est proposé dans [HNM02]. Un canal quasi-fiable est un canal FIFO qui ne perd des messages que s'il le nombre de messages dans le canal dépasse une certaine limite. La mémoire des processeurs peut être fini, et ils montrent facilement comment leur protocole peut être implanté.

Dans [DIM97a], les auteurs présentent trois protocoles de synchronisation auto-stabilisants de deux processeurs. les deux premiers protocoles sont proches des protocoles auto-stabilisants de la fenêtre glissante et du bit alterné proposés respectivement dans [GM91] et dans [AB93]. Le troisième protocole fait l'hypothèse originale que les deux processeurs ont un nombre fini d'états mais que la taille du canal croit infiniment.

4.2 Protocole de la fenêtre glissante

Dans [GM91], est proposé une variante auto-stabilisante des protocoles de la *poignée de main en deux temps* (*Two-Way Handshake*) et et de la *fenêtre glissante* (*Sliding Window*). Ils supposent que les canaux de communication sont fiables. La stabilisation de ces deux protocoles est obtenue en numérotant chaque message avec une séquence infinie d'entiers. Il s'en suit que la taille des canaux et de la mémoire des processeurs ne peut être qu'infinie. Dans [CV96, Spi97] d'autres versions du protocole de la *fenêtre glissante*

sont proposés. Les protocoles supposent que la taille de la mémoire des processeurs est finie et que la taille du FIFO canal bidirectionnel est bornée par une borne B. Ils associent à chaque message un numéro de séquence dont la borne est supérieur à B. Le protocole de [CV96] utilise la technique du *counter flushing* présenté dans [Var00] : l'émetteur va inévitablement envoyer une paquet ayant un numéro de séquence distinct de tous les autres paquets en transit. Lorsqu'il recevra l'accusé de réception de ce paquet le canal contiendra seulement des paquets effectivement envoyés durant cette connexion (le canal est donc nettoyé des "vieux"paquets). Dans [Spi97], lorsque un paquet n'ayant pas le numéro de séquence attendu est reçu, le destinataire de ce paquet envoie un grand nombre de "reset"paquets (plus grand que la taille du canal plus la taille de la fenêtre d'anticipation). Après que tous ces "reset"paquets soient acquittés alors le canal est nettoyé.

4.3 Protocole d'ouverture de connexion

Dans [Her92], un protocole auto-stabilisant d'ouverture et de maintenance de connexion est proposé dans le cadre d'un réseau dynamique. Ce protocole suppose que le protocole chargé de la maintenance des tables de routage est lui-même auto-stabilisant. Périodiquement, l'émetteur et la destination envoient des paquets de contrôle le long du circuit virtuel pour vérifier que le circuit est fonctionnel. Si pour n'importe quelle raison, le circuit n'est plus opérationnel (par exemple dysfonctionnement des sous-réseaux ou pannes des routeurs intermédiaires), le circuit est fermé. L'émetteur établit un nouveau circuit virtuel pour continuer le transfert de données. Le protocole garantit qu'au plus N paquets envoyés sur un circuit n'arrivent pas à destination avant que le dysfonctionnement du circuit ne soit détecté. Cette approche est peu efficace dans le cas de réseaux de très grande taille. En effet, plus le réseau est de grande taille, plus la topologie du réseau change fréquemment. Donc les circuits virtuels sont opérationnels de moins en moins longtemps. Ainsi, le débit de transfert de données d'un protocole basé sur l'approche présentée dans [Her92] se réduit peu à peu à un "filet d'eau".

4.4 Protocole de transmission de données via plusieurs chemins

Des protocoles de communication point à point auto-stabilisants sont présentés dans [APSV96, DW97]. Le réseau consiste en un ensemble de nœuds/routeurs connectés par les canaux FIFO de taille borné pouvant perdre des messages mais pas les dupliquer. Ces protocoles utilisent plusieurs chemins entre l'émetteur et le destinataire; ainsi même si un chemin est hors service, la délivrance des paquets n'est pas stoppée.

Dans [APSV96], il est supposé que chaque canal de communication est un *UDL*. Un UDL est un canal de taille 1, ayant donc au plus un paquet en transit. Sachant qu'un UDL peut être réalisé sur n'importe quel canal de taille borné *via* une version auto-stabilisante du protocole du bit alterné [AB93], cette hypothèse n'est pas limitante. Supposant que chaque canal est un UDL, un protocole de type *glissement de paquets* (*slide protocol*) réduit le réseau à un C-*channel* de l'émetteur au destinataire. Un C-*channel* est un canal contenant au plus C paquets, il assure la délivrance des paquets mais il peut les réordonner. L'émetteur et la destination se synchronisent au moyen du bit alterné auto-stabilisant. Une fois synchronisés, l'émetteur envoie $X \geq 2C + 1$ fois le même paquet *via* le C-*channel* (c'est à dire dans le réseau) à la destination. Parmi les X prochains paquets reçus par le destinataire, au moins $X - C$ sont identiques. Ainsi,

le destinataire peut extraire le paquet à délivrer *via* un mécanisme de vote à la majorité simple sur les X paquets reçus.

Dans [DW97], chaque paquet contient la description complète du chemin qu'il doit prendre et un numéro de séquence lié au chemin. Le principe de ce protocole est que deux paquets envoyés sur le même chemin arrivent à destination dans l'ordre d'envoi s'il ne sont pas perdus car les canaux de communication sont FIFO. Lorsqu'un paquet arrive à la destination, cette dernière vérifie qu'il a le numéro de séquence attendu pour le chemin pris par ce paquet. Si oui, le paquet est délivré et un acquittement est envoyé. Cet acquittement contient le numéro de séquence que devra avoir le prochain paquet pour être délivré. Le protocole nettoie périodiquement les chemins utilisés dans le réseau : il les vide des "vieux"paquets. C'est pourquoi le protocole peut utiliser des numéros de séquence bornés.

5 Protocoles de contrôle du réseau

La plupart des tâches à réaliser en vue du contrôle de réseau reviennent souvent à amener les processeurs à s'entendre sur une valeur commune ou à se synchroniser. Parmi les problèmes à résoudre en vue d'atteindre ces objectifs, on trouve l'*élection d'un leader* (*LE* pour *Leader Election*) et les *protocoles à vagues*. Le premier consiste à contraire un protocole qui amène l'ensemble des processeurs à choisir un des leurs comme leader. Une fois que le réseau dispose d'un unique leader, de nombreuses tâches demandant un contrôle centralisé peuvent être réalisées, par exemple, la maintenance d'une base de données réparties, la détection de cycle dans les tables de routage, etc. Les seconds entrent dans la résolution de nombreux problèmes de synchronisation comme le partage ou l'allocation de ressources, la diffusion d'informations, etc.

Dans ce qui suit, nous présentons les résultats les plus récents pour ces deux paradigmes obtenus dans le domaine de l'auto-stabilisation. La plupart de ces travaux sauf indication contraire fonctionnent dans le modèle à registres (Section 2.1).

5.1 Protocoles d'élection d'un leader

De nombreux travaux ont montrés l'importance des hypothèses faites sur le modèle choisi, comme la présence ou non d'identité sur les processeurs, le type d'ordonnanceur (modélisation d'un adversaire chargé de mettre en péril la solution au problème étudié), synchronisation globale, etc. Nous présentons ci-dessous différentes familles de résultats obtenus en fonction des hypothèses faites.

5.1.1 Protocoles déterministes sur des réseaux avec identifiants

Dans un réseau avec identifiants, chaque routeur possède une adresse réseau propre qui ne peut être corrompue car elle est stockée en ROM. Il existe plusieurs protocoles auto-stabilisants de LE déterministes où l'espace mémoire dépend de la taille du réseau [AKY90, AK93, AG94].
Un protocole auto-stabilisant est *silencieux* ([DGS99]) si une fois que le système est stabilisé, l'état des processeurs n'est pas modifié. Les processeurs vont uniquement vérifier périodiquement que l'état de leurs voisins est inchangé. Ainsi, un protocole silencieux est moins coûteux en terme de communication, une fois que le réseau est stabilisé, qu'un protocole non silencieux. Dans [DGS99], il a été établi que l'espace

mémoire nécessaire pour un protocole de LE silencieux est $O(\lg N)$ dans le cas général, N étant la taille de l'anneau.

5.1.2 Protocoles déterministes sur des réseaux anonymes

Un réseau anonyme est un réseau où les nœuds n'ont pas d'identifiant propre et exécutent le même code. Un protocole auto-stabilisant doit converger vers une configuration où il y a un leader à partir de n'importe quelle configuration. Sur un anneau anonyme, un protocole de LE peut être conçu uniquement *(i)* si l'anneau est de taille première et *(ii)* si l'ordonnanceur est centralisé [Ang80, BP89]. Sous un *ordonnanceur centralisé*, à chaque étape de calcul, un seul routeur exécute son code local. Ce type d'hypothèse implique implicitement un contrôle centralisé du réseau. Dans [ILS95], un protocole de LE déterministe sous un ordonnanceur centralisé pour les anneaux bidirectionnels nécessitant un espace mémoire constant est présenté (c'est à dire que l'espace mémoire nécessaire au protocole est indépendant de la taille de l'anneau). Dans [BGJ99a], il est prouvé qu'un protocole déterministe auto-stabilisant de LE pour les anneaux unidirectionnels nécessite au moins $\log_2(N)$ bits sur chaque élément du réseau. Un protocole nécessitant $4 + \log_2(N)$ bits par élément du réseau est présenté dans [FJ01].

5.1.3 Protocoles probabilistes

Pour s'abstraire des difficultés rencontrées dans la section précédente, les protocoles fonctionnant sur les réseaux anonymes sont en général probabilistes. Dans le cas de protocoles probabilistes, le résultat d'une action d'un nœud est aléatoire. Formellement, il dépend de la valeur d'un variable aléatoire.

Un protocole auto-stabilisant de LE sur des réseaux en anneau est présenté dans [MOOY92]. Ce protocole suppose que la "vivacité"du réseau est assuré de manière externe un mécanisme externe insère un message dans l'anneau uniquement lorsqu'il aucun paquet/message circulant dans l'anneau. Ce protocole est le seul protocole supposant que les éléments du réseaux communiquent par messages. Dans [BGJ99a], un protocole auto-stabilisant de LE sur un anneau est présenté. Dans [DIM97b, BDLGJ02], des protocoles auto-stabilisants de LE pour des réseaux de topologie quelconque sont présentés. Il a été prouvé que l'espace mémoire nécessaire aux protocoles [BGJ99a, BDLGJ02] est minimal.

5.2 Protocoles à vagues

Les *protocoles à vagues* sont des outils de synchronisation à l'origine de nombreuses solutions dans la résolution de problèmes de contrôle du réseau. Une vague est en fait une exécution du protocole impliquant tous les processeurs du réseau et se terminant par une action particulière communément appelée *décision*. Les deux familles de protocoles à vagues les plus utilisés sont la *circulation d'un jeton* (*TC* pour *Token Circulation*) et la *propagation d'information avec retour* (*PIF* pour *Propagation of Information with Feedback*).

Dans le premier cas, un unique processeur initialise la circulation d'un unique jeton. Les circulations de jeton les plus étudiées sont :

- la circulation en profondeur (*DFTC* pour *Depth-First Token Circulation*) car elle offre l'avantage d'être déterministe et équitable.

- la circulation de jeton probabiliste car, contrairement à la circulation de jeton en profondeur, elle offre l'avantage de fonctionner sur les réseaux anonymes.

La TC est particulièrement adaptée à la résolution de problèmes ayant un comportement séquentiel, comme l'exclusion mutuelle.

Le PIF peut être décrit de la manière suivante : un processeur initialise une vague de PIF en "diffusant"un message m. La vague se termine lorsque l'émetteur de m a reçu l'accusé de réception de m de la part de tous les processeurs du réseau. Le PIF est donc un mécanisme de synchronisation exploitant au mieux le parallélisme offert par la topologie du réseau.

5.2.1 La circulation en profondeur de jeton

De nombreux protocoles auto-stabilisants ont été proposés sur des réseaux de topologies spécifiques : la chaîne et l'anneau ; par exemple [BGW89, BP89, Dij74, Gho93, GH96]. Des protocoles DFTC auto-stabilisants ont également été proposés sur des topologies plus complexes comme les arbres couvrants [DIM93, Pet97, PV99, PV00] ou quelconques [HC93, JB95, PV97a, DJPV00, Pet01]. La plupart des travaux récents portent sur la minimisation de l'espace mémoire nécessaire au bon fonctionnement du protocole ou du temps nécessaire au système pour stabiliser.

L'intérêt porté à la réduction de l'espace mémoire s'explique par le fait que contrairement aux solutions fondés sur la connaissance d'une donnée globale au réseau (la taille N, le diamètre D...), en obtenant un protocole nécessitant seulement un espace constant par canal de communication, celui devient transparent à l'évolution de la topologie du réseau. AInsi, le protocole est dynamique. Les protocoles présentés dans [Pet97, PV99, PV00] sont des DFTC à mémoire constante par canal, leur optimalité en espace est prouvée dans [PV00]. Cependant, ils ne fonctionnent que sur des arbres couvrants. Les DFTC présentés dans [JB95, PV97a, DJPV00] sont à mémoire constante ([DJPV00] ayant la meilleure complexité connue à ce jour) par canal et fonctionnent sur des réseaux quelconques.

En terme de temps de stabilisation, les deux protocoles pour les réseaux dont la topologie est un arbre qui sont présentés dans [PV99] sont *instantanément stabilisants* [BDPV99b], c'est à dire qu'ils stabilisent en temps 0. Le fait qu'ils soient instantanément stabilisés signifie que indépendamment de l'état initial, durant une circulation, tous les nœuds ont le jeton Les protocoles proposés dans [PV99] sont donc optimaux en temps de stabilisation (en plus de l'être en espace mémoire). A ce jour, la meilleure complexité en temps de stabilisation dans un graphe quelconque est $O(N)$ [Pet01].

Tous les protocoles présentés ci-dessus ont été écrits dans le modèle à registres. Les protocoles de [JB95] et de [PV97a] sont respectivement adaptés aux communications par échange de messages dans [PV97c] et dans [PV97b].

5.2.2 La circulation de jeton probabiliste

La plupart des protocoles pour les anneaux unidirectionnels sont basés sur la technique : "retarder pour une période aléatoire la circulation du jeton"[Her90, BCD95, KY97, BGJ99b, Ros00, DGT00]. Le processeur ayant le jeton décide aléatoirement de garder ou de passer le jeton à son unique voisin. Une fois l'anneau stabilisé, la circulation du jeton est également retardé. L'algorithme de [DGT00] garantit une borne supérieure en $O(N^3)$ au *temps de service*. Le temps de service est est le temps nécessaire à un jeton

pour faire le tour de l'anneau. Les autres protocoles n'ont pas de borne supérieur au temps de service.

Les protocoles présentés dans [KY97, KY02, Joh02], sont basés sur une autre technique. Un jeton T est bloqué sur un nœud lorsque l'anneau a plusieurs jetons, les autres jetons vont continuer à circuler. Finalement, le jeton bloqué va être rattrapé par un autre jeton; les deux jetons vont fusionner un un seul jeton. Ainsi le nombre de jetons va diminuer. Une fois l'anneau stabilisé, la circulation de l'unique jeton n'est pas retardé ou peu, c'est pourquoi le temps de service est optimal dans le cas des protocoles présentés dans [KY97, Joh02]. Le protocole de [Joh02] fonctionne sous tout ordonnanceur alors que le protocole de [KY97] fonctionne que sous un ordonnanceur centralisé.

D'une manière générale, la marche aléatoire [AKL$^+$79] est une technique fort utilisée pour implanter une circulation de jeton dans un réseau quelconque : le nœud ayant le jeton choisit aléatoirement le voisin à qui il va passer le jeton. Des protocoles auto-stabilisants fondés sur cette technique sont proposés dans [IJ90, DL00].

Il y a peu de travaux sur la circulation de jeton dans le cas où les communications sont réalisés par échanges de messages [MOOY92, HM98, Ros00]. Les protocoles existants fonctionnent uniquement sur des réseaux en anneaux et supposent qu'un mécanisme externe et fiable insère une jeton dans l'anneau s'il n'y a pas.

5.2.3 La propagation d'information avec retour

Le problème du PIF est équivalent au problème de la TC sur l'anneau et sur la chaîne. De nombreux protocoles auto-stabilisants de PIF ont été apportés dans la résolution de problèmes, comme l'élection de leader [DIM97b, Var93], la ré-initialisation [AKY90, AG94, AKM$^+$93] et la synchronisation [ABDT98, AKM$^+$93, AV91] du système. Tous ces travaux supposent la construction ou l'existence d'un arbre couvrant du réseau. Dans le cas d'un protocole de PIF seulement auto-stabilisant, en fonction de l'état initial, une vague se terminant peut ne pas avoir atteint tous les processeurs, en autres mots certains processeurs peuvent ne pas avoir reçu le message envoyé par l'initiateur de la vague. L'auto-stabilisation garantit seulement qu'après un nombre fini de vagues, toutes les vagues suivantes seront correctes et atteindront tous les processeurs

De même que pour la circulation de jeton en profondeur, plusieurs protocoles de PIF sur des arbres couvrants à la fois optimaux en espace et instantanément stabilisants ont été proposés dans [BDPV99a, BDPV99b]. Le fait que ces protocoles soient instantanément stabilisés signifie que lorsqu'un processeur décide d'envoyer un message à tous les autres processeurs du réseau en initialisant une vague de PIF, il est sûr que lorsque la vague de termine, tous les processeurs ont bien reçu son message.

Les seuls protocoles auto-stabilisants de PIF pour des réseaux quelconques fonctionnant sans l'hypothèse d'un arbre couvrant sont ceux présentés dans [Var00, CDPV01a, CDPV02]. Le protocole de [CDPV02] est instantanément stabilisant.

6 Auto-stabilisation automatique de protocoles

Bien souvent, un protocole ne possède pas la propriété d'auto-stabilisation, mais doit toutefois être exécuté dans un environnement où des défaillances peuvent survenir, ou bien la topologie du réseau être modifiée pendant l'exécution du protocole. Il est alors intéressant de disposer d'une "machine à auto-stabiliser ", c'est à dire un compilateur qui prend en entrée un protocole non-auto-stabilisant et qui fournit en sortie un autre

protocole, qui effectue la même tâche que le protocole fourni en entrée, mais qui possède lui la propriété d'être auto-stabilisant.

Pour proposer un tel compilateur, les auteurs supposent en général qu'une incohérence peut être détectée, et que cette détection déclenche la réinitialisation complète du système. On peut considérer cette technique comme une forme d'auto-stabilisation. En effet, partant d'un état quelconque, le protocole de calcul de l'état global détecte l'inconsistance du système puis lance une réinitialisation du système en cas de réponse positive.

De nombreux travaux du domaine de l'auto-stabilisation sont fondés sur cette technique, les plus représentatifs étant [AG94, AKY90, KP93, CDPV01b]. Toutes ces solutions utilisent un contrôle centralisé : un processus unique - le leader - vérifie la consistance du système vis-à-vis de la spécification du problème à résoudre. En cas d'inconsistance, le leader déclenche une réinialisation globale du système. Alors que les solutions proposées dans [AG94, AKY90, CDPV01b] fonctionnent dans le modèle à registres, la solution proposée dans [KP93] fonctionne dans le modèle à passage de messages. Malheureusement, ce protocole engendre un surcoût important en nombre de messages, le rendant ainsi inutilisable dans la pratique. En effet, sur un réseau complet, sa complexité en nombre de messages est $\Omega(N!)$ une fois stabilisé, N étant le nombre de processus dans le réseau. Dans les mêmes conditions, une version implantable est proposée dans [FV97] puisque, dans les mêmes conditions d'analyse, elle n'utilise que $O(N^3)$ messages.

Fonctionnant selon le même principe que [KP93], même s'il ne fonctionne dans le modèle à registres, le compilateur proposé dans [CDPV01b] construit une version instantanément stabilisante d'un protocole stabilisant.

Les techniques proposées ci-dessus comporte l'inconvénient évident de ré-initialiser l'ensemble du système. Or, il convient de noter que circonscrire localement l'effet d'une faute transitoire ou pas intervenue sur l'un des éléments du réseau (processeur, lien de communication) équivaut à circonscrire un changement de topologie. En outre, les changements de topologie ne concernent généralement qu'une région assez restreinte du réseau. En fait, lorsque que cela est possible, il serait plus avantageux de circonscrire l'effet de l'incohérence lié au changement de topologie à son voisinage le plus proche. Cette approche, d'abord suggérée dans [AKY90], est abordée sous différents angles dans [GGHP96, DH97, BDDT98] (Section 7).

7 Auto-stabilisation améliorée

Etant donné que l'auto-stabilisation permet de tolérer un nombre arbitrairement grand de défaillances, cette technique souffre fréquemment de plusieurs défauts, parmi lesquels on peut compter le processus de récupération après une erreur peut perturber tout le réseau même si uniquement un petit nombre de processeurs étaient défaillants, par exemple [APSV91, APSVD94, AKY97, DH97, KP93].

7.1 La k-stabilisation

Les protocoles k-stabilisants [BGK99] sont des protocoles stabilisants pour le cas où une borne supérieure $k \leq N$ sur le nombre de fautes est connue où N est le nombre total de processeurs. Pour modéliser une faute, on utilise la distance de Hamming entre les configurations [DH97]. Plus spécifiquement, si C_1 et C_2 sont des configurations, la

distance entre elles est le nombre de processeurs dont les états locaux sont différents dans C_1 et dans C_2. Soit C une configuration non-légitime et L une configuration légitime dont la distance à C est la plus petite (L n'est pas forcément unique). Le nombre de fautes dans C est la distance à L. Les processeurs défaillants sont ceux dont l'état dans C et dans L sont différents.

7.2 L'adaptabilité en temps

Le fait que le processus de récupération sur erreur puisse impliquer le réseau tout entier empêche son utilisation dans des réseaux de grande taille tels qu'Internet. Pour permettre le passage à l'échelle, il a été suggéré (par exemple dans [GH02, KPS99, KP99]) que plus le nombre de fautes touchant le réseau était réduit, et plus le temps de reprise sur erreur devrait être faible. De tels protocoles (appelés *fault local*, *fault scalable* ou encore *time adaptive* dans la littérature) ont été proposés tout d'abord pour des cas simples. Ces cas simples peuvent être *(i)* qu'un processeur est capable de se rendre compte qu'il est défaillant [AD02], ou *(ii)* considérer uniquement le cas des tâches non-réactives [KPS99, KP99]. Une tâche non-réactive est un calcul réparti dont le résultat est fonction des données en entrée. Ainsi, pour un ensemble de données en entrée, le calcul est effectué une seule fois.

Par la suite, des études ont été menées pour le problème du passage de jeton [BGK99, Gen00, GT01]. Cependant, [GT02] montre que pour les systèmes dynamiques (tels que le passage de jeton), il existe une borne inférieure de l'ordre de N pas de calcul avant un retour à la normale, et ce même en présence d'une seule faute. Ce résultat implique que les protocoles adaptables en temps n'ont de sens que dans les systèmes synchrones ou si on mesure la complexité en termes de tours d'exécution.

7.3 La k-adaptabilité en temps

Les protocoles auto-stabilisants peuvent être vus comme un cas particulier des protocoles k-stabilisants où $k = N$ est la taille du réseau. Similairement, les protocoles adaptables en temps peuvent être vus comme un simple raffinement des protocoles auto-stabilisants. Dans [GT01] est introduite la notions de k-adaptabilité pour dénoter la capacité à retrouver un comportement correct à partir d'une configuration comprenant k processeurs corrompus en un temps qui dépend uniquement du nombre effectif de fautes. Pour des raisons historiques, la plupart des protocoles qualifiés d'adaptables en temps [BGK99, KPS99, KP99] sont k-adaptables en temps selon cette terminologie : ils présupposent qu'il existe une borne sur le nombre de fautes qui atteignent le système pour garantir un temps de stabilisation rapide.

7.4 Une classification

Du point de vue des protocoles, notre classification est représentée ci-après où les flèches signifient "est un cas particulier de". Par exemple, la flèche entre "adaptabilité en temps"et "auto-stabilisation"signifie que tout protocole adaptable en temps est également auto-stabilisant.

Auto-stabilisation	$\leftarrow$	Adaptabilité en temps
$\downarrow$		$\downarrow$
k-stabilisation	$\leftarrow$	k-adaptabilité en temps

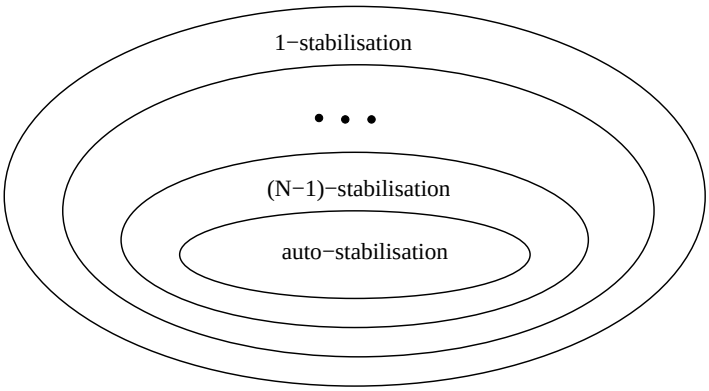

Figure 2: Classification des problèmes

Du point de vue des problèmes, notre classification est représentée Figure 2. Par exemple, tout problème qui admet une solution auto-stabilisante admet également une solution $(N-1)$-stabilisante, il suffit de prendre la solution auto-stabilisante. Donc, l'ensemble des problèmes qui peut être résolu au moyen de protocoles auto-stabilisants est inclus dans l'ensemble des problèmes qui peut être résolu par un protocole $(N-1)$-stabilisant. Pour la même raison, l'ensemble des problèmes k-adaptifs en temps (c'est à dire les problèmes qui peuvent être résolus par un protocole k-adaptif en temps) est inclus dans l'ensemble des problèmes k-stabilisants qui peuvent être trivialement résolus en utilisant le protocole k-adaptif en temps.

8 Conclusion

Nous avons présenté plusieurs travaux proposant des solutions auto-stabilisantes à des problèmes classiques dans le domaine des réseaux informatiques, notamment dans les domaines du routage, des communications point-à-point et des protocoles de contrôle. Nous avons également présenté des techniques de transformations automatiques de protocoles en leurs équivalents auto-stabilisants, ainsi que des améliorations portant sur le temps de stabilisation lorsque le nombre de défaillances est faible.

References

[AACD+96] H Abu-Amara, B Coan, S Dolev, A Kanevsky, and JL Welch. Self-stabilizing topology maintenance protocols for high-speed networks. *IEEE-ACM Transactions on Networking*, 4(6):902–912, 1996.

[AB93] Y. Afek and G. M. Brown. Self-stabilization over unreliable communication media. *Distributed Computing*, 7(1):27–34, 1993.

[ABDT98] L. O. Alima, J. Beauquier, A. K. Datta, and S. Tixeuil. Self-stabilization with global rooted synchronizers. In *ICDCS98 18th International Conference on Distributed Computing Systems*, pages 102–109, 1998.

[AD02] Y Afek and S Dolev. Local stabilizer. *Journal of Parallel and Distributed Computing*, 62(5):745–765, 2002.

[AG94] A. Arora and M. G. Gouda. Distributed reset. *IEEE Transactions on Computers*, 43(9):1026–1038, 1994.

[AGH90] A. Arora, M. G. Gouda, and T. Herman. Composite routing protocols. In *SPDP90 2nd IEEE Symposium on Parallel and Distributed Processing*, pages 70–78, 1990.

[AK93] S. Aggarwal and S. Kutten. Time optimal self-stabilizing spanning tree algorithm. In *FSTTCS93 13th Conference on Foundations of Software Technology and Theoretical Computer Science, Springer-Verlag LNCS:761*, pages 400–410, 1993.

[AKL+79] R. Aleliunas, R. M. Karp, R. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal traversal sequences and the complexity of the maze problem. In *FOCS79 20th Annual IEEE Symp. on Foundation of Computer Science*, pages 218–223, 1979.

[AKM+93] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese. Time optimal self-stabilizing synchronization. In *STOC93 25th Annual ACM Symposium on Theory of Computing*, pages 652–661, 1993.

[AKY90] Y. Afek, S. Kutten, and M. Yung. Memory-efficient self-stabilization on general networks. In *WDAG90 Distributed Algorithms 4th International Workshop, Springer-Verlag LNCS:486*, pages 15–28, 1990.

[AKY97] Y. Afek, S. Kutten, and M. Yung. The local detection paradigm and its applications to self-stabilization. *Theoretical Computer Science*, 186(1-2):199–229, 1997.

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

[APSV91] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction. In *FOCS91 31th Annual IEEE Symposium on Foundations of Computer Science*, pages 268–277, 1991.

[APSV96] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilizing end-to-end communication. *Journal of High Speed Networks*, 5(4):365–381, 1996.

[APSVD94] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev. Self-stabilization by local checking and global reset. In *WDAG94 Distributed Algorithms 8th International Workshop, Springer-Verlag LNCS:857*, pages 326–339, 1994.

[AV91] B. Awerbuch and G. Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols. In *FOCS91 31th Annual IEEE Symposium on Foundations of Computer Science*, pages 258–267, 1991.

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

[BDDT98] J. Beauquier, S. Delaët, S. Dolev, and S. Tixeuil. Transient fault detectors. In *DISC98 Distributed Computing 12th International Symposium, Springer LNCS:1499*, pages 62–74, 1998.

[BDLGJ02] J. Beauquier, J. Durand-Lose, M. Gradinariu, and C. Johnen. Token based self-stabilizing uniform algorithms. *Journal of Parallel and Distributed Computing*, 62(5):899–921, May 2002.

[BDPV99a] A. Bui, A. K. Datta, F. Petit, and V. Villain. Snap-stabilizing PIF algorithm in tree networks without sense of direction. In *SIROCCO99 6th International Colloquium On Structural Information and Communication Complexity*, pages 32–46. Carleton University Press, 1999.

[BDPV99b] A. Bui, A. K. Datta, F. Petit, and V. Villain. State-optimal snap-stabilizing PIF in tree networks. In *WSS99 fourth Workshop on Self-Stabilizing Systems*, pages 78–85. IEEE Computer Society Press, 1999.

[BDT99] J. Beauquier, A. K. Datta, and S. Tixeuil. Self-stabilizing census with cut-through constraint. In *WSS99 fourth Workshop on Self-Stabilizing Systems*, pages 70–77. IEEE Computer Society Press, 1999.

[BGJ99a] J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols. In *PODC99 18th Annual ACM Symposium on Principles of Distributed Computing*, pages 199–208, 1999.

[BGJ99b] J. Beauquier, M. Gradinariu, and C. Johnen. Randomized self-stabilizing and space optimal leader election under arbitrary scheduler on rings. Technical Report 1225, LRI - Laboratoire de recherche en Informatique, Université de Paris Sud, Orsay, France, 1999.

[BGK99] J. Beauquier, C. Genolini, and S. Kutten. Optimal reactive k-stabilization: the case of mutual exclusion. In *PODC99 18th Annual ACM Symposium on Principles of Distributed Computing*, pages 209–218, 1999.

[BGW89] G. M. Brown, M. G. Gouda, and C. L. Wu. Token systems that self-stabilize. *IEEE Transactions on Computers*, 38(6):845–852, 1989.

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

[CDPV01a] A. Cournier, A. K. Datta, F. Petit, and V. Villain. Self-stabilizing PIF algorithm in arbitrary rooted networks. In *ICDCS01 21th IEEE International Conference on Distributed Computing Systems*, pages 91–98, 2001.

[CDPV01b] A. Cournier, A. K. Datta, F. Petit, and V. Villain. Snap-stabilizing systems. Technical Report 11, LaRIA - Laboratoire de Recherche en Informatique d'Amiens, Université de Picardie Jules Verne, Amiens, France, 2001.

[CDPV02] A. Cournier, A. K. Datta, F. Petit, and V. Villain. Snap-stabilizing PIF algorithm in arbitrary networks. In *ICDCS02 22th IEEE International Conference on Distributed Computing Systems*, pages 199–206, 2002.

[CDT02] Y. Chen, A.K. Datta, and S. Tixeuil. Stabilizing inter-domain routing in the internet. In *Euro-Par'02 Parallel Processing, Springer-Verlag LNCS:2400*, pages 749–752, 2002.

[CG01] J. A. Cobb and M. G. Gouda. Stabilization of routing in directed networks. In *WSS01 Fifth 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.

[CV96] A. Costello and G. Varghese. Self-stabilization by window washing. In *PODC96 15th Annual ACM Symposium on Principles of Distributed Computing*, pages 35–44, 1996.

[CV99] A. M. Costello and G. Varghese. The FDDI MAC meets self-stabilization. In *WSS99 fourth Workshop on Self-Stabilizing Systems*, pages 1–9. IEEE Computer Society Press, 1999.

[CW97] J. A. Cobb and M. Waris. Propagated timestamps: a scheme for the stabilization of maximum-flow routing protocols. In *WSS97 Third Workshop on Self-Stabilizing Systems*, pages 185–200. Carleton University Press, 1997.

[DDLT00] A. K. Datta, J. L. Derby, J. E. Lawrence, and S. Tixeuil. Stabilizing hierarchical routing. *Journal of Interconnexion Networks*, 1(4):283–302, 2000.

[DGKT02] A. K. Datta, M. Gradinariu, A. B. Kenitzky, and S. Tixeuil. Self-stabilizing wormhole routing in ring networks. In *ICPADS02 International Conference on Parallel and Distributed Systems*, 2002.

[DGS99] S. Dolev, M. G. Gouda, and M. Schneider. Memory requirements for silent stabilization. *Acta Informatica*, 36(6):447–462, 1999.

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

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

[Dij74] E. W. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):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.

[DIM97a] S. Dolev, A. Israeli, and S. Moran. Resource bounds for self-stabilizing message-driven protocols. *SIAM Journal on Computing*, 26(1):273–290, 1997.

[DIM97b] S. Dolev, A. Israeli, and S. Moran. Uniform dynamic self-stabilizing leader election. *IEEE Transactions on Parallel and Distributed Systems*, 8(4):424–440, 1997.

[DJPV00] A. K. Datta, C. Johnen, F. Petit, and V. Villain. Self-stabilizing depth-first token circulation in arbitrary rooted networks. *Distributed Computing*, 13(4):207–218, 2000.

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

[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.

[DT02] S. Delaët and S. Tixeuil. Tolerating transient and intermittent failures. *Journal of Parallel and Distributed Computing*, 62(5):961–981, 2002.

[DW97] S. Dolev and J. L. Welch. Crash resilient communication in dynamic networks. *IEEE Transactions on Computers*, 46(1):14–26, 1997.

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

[FV97] O. Flauzac and V. Villain. An implementable dynamic automatic self-stabilizing protocol. In *I-SPAN'97 Third International Symposium on Parallel Architectures, Algorithms and Networks*, pages 91–97. IEEE Computer Society Press, 1997.

[Gen00] C. Genolini. Optimal k-stabilization: the case of synchronous mutual exclusion. In *PDCS00 13th International Conference on Parallel and Distributed Computing Systems*, pages 371–376, 2000.

[GGHP96] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *PODC96 15th Annual ACM Symposium on Principles of Distributed Computing*, pages 45–54, 1996.

[GGP97] S. Ghosh, A. Gupta, and S. V. Pemmaraju. A self-stabilizing algorithm for the maximum flow problem. *Distributed Computing*, 10(4):167–180, 1997.

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

[GH02] S. Ghosh and X. He. Scalable self-stabilization. *Journal of Parallel and Distributed Computing*, 62(5):945–960, 2002.

[Gho93] S. Ghosh. An alternative solution to a problem on self-stabilization. *ACM Transactions on Programming Languages and Systems*, 15(4):735–742, 1993.

[GM91] M. G. Gouda and N. Multari. Stabilizing communication protocols. *IEEE Transactions on Computers*, 40(4):448–458, 1991.

[GS95] M. G. Gouda and M. Schneider. Maximum flow routing. In *WSS95 Second Workshop on Self-Stabilizing Systems*, pages 2.1–2.13, 1995.

[GS98] M. G. Gouda and M. Schneider. Maximizable routing metrics. In *ICPN98 Sixth International Conference on Network Protocols*, pages 71–78, 1998.

[GS99] M. G. Gouda and M. Schneider. Stabilization of maximal metric trees. In *WSS99 fourth Workshop on Self-Stabilizing Systems*, pages 10–17. IEEE Computer Society Press, 1999.

[GT01] C. Genolini and S. Tixeuil. Reactive k-stabilization and time adaptivity: possibility and impossibility results. Technical Report 1276, LRI - Laboratoire de recherche en Informatique, Université de Paris Sud, Orsay, France, 2001.

[GT02] C. Genolini and S. Tixeuil. A lower bound on dynamic k-stabilization in asynchronous systems. In *SRDS02 21th Symposium on Reliable Distributed Systems*, pages 211–221. IEEE Computer Society Press, 2002.

[HC93] S. T. Huang and N. S. Chen. Self-stabilizing depth-first token circulation on networks. *Distributed Computing*, 7(1):61–66, 1993.

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

[Her92] A. Herzberg. Connection-based communication in dynamic networks. In *PODC92 11th Annual ACM Symposium on Principles of Distributed Computing*, pages 13–24, 1992.

[Her02] T. Herman. A comprehensive bibliography on self-stabilization. *Chicago Journal of Theoretical Computer Science, available at* `http://www.cs.uiowa.edu/ftp/selfstab/bibliography/`, 2002.

[HM98] L. Higham and S. Myers. Self-stabilizing token circulation on anonymous message passing. In *OPODIS98 Second International Conference On Principles Of DIstributed Systems*, pages 115–128, 1998.

[HNM02] R. R. Howell, M. Nesterenko, and M. Mizuno. Finite-state self-stabilizing protocols in message-passing systems. *Journal of Parallel and Distributed Computing*, 62(5):792–817, 2002.

[IJ90] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *PODC90 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 119–131, 1990.

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

[JB95] C. Johnen and J. Beauquier. Space-efficient distributed self-stabilizing depth-first token circulation. In *WSS95 Second Workshop on Self-Stabilizing Systems*, pages 4.1–4.15, 1995.

[Joh02] C. Johnen. Service time optimal self-stabilizing token circulation protocol on anonymous unidirectional rings. In *SRDS02 21th Symposium on Reliable Distributed Systems*, pages 80–89. IEEE Computer Society Press, 2002.

[KP93] S. Katz and K. J. Perry. Self-stabilizing extensions for message-passing systems. *Distributed Computing*, 7(1):17–26, 1993.

[KP99] S. Kutten and D. Peleg. Fault-local mending. *Journal of Algorithms*, 30(1):144–165, 1999.

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

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

[KY02] 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, May 2002.

[Lam83] L. Lamport. Solved problems, unsolved problems and non-problems in concurrency, invited address. In *PODC84 Third Annual ACM Symposium on Principles of Distributed Computing*, pages 1–11, 1983.

[Mas95] T. Masuzawa. A fault-tolerant and self-stabilizing protocol for the topology problem. In *WSS95 Second Workshop on Self-Stabilizing Systems*, pages 1.1–1.15, 1995.

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

[Per00] R. Perlman. *Interconnexion Networks.* Addison-Wesley, 2000.

[Pet97] F. Petit. Highly space-efficient self-stabilizing depth-first token circulation for trees. In *OPODIS97 First International Conference On Principles Of DIstributed Systems*, pages 221–235, 1997.

[Pet01] F. Petit. Fast self-stabilizing depth-first token circulation. In *WSS01 Fifth International Workshop on Self-Stabilizing Systems, Springer LNCS:2194*, pages 200–215, 2001.

[PV97a] F. Petit and V. Villain. Color optimal self-stabilizing depth-first token circulation. In *I-SPAN'97 Third International Symposium on Parallel Architectures, Algorithms and Networks*, pages 317–323. IEEE Computer Society Press, 1997.

[PV97b] F. Petit and V. Villain. Color optimal self-stabilizing depth-first token circulation protocol for asynchronous message-passing. In *PDCS97 10th International Conference on Parallel and Distributed Computing Systems*, pages 227–233, 1997.

[PV97c] F. Petit and V. Villain. A space-efficient and self-stabilizing depth-first token circulation protocol for asynchronous message-passing systems. In *Euro-par'97 Parallel Processing, Springer-Verlag LNCS:1300*, pages 476–479, 1997.

[PV99] F. Petit and V. Villain. Time and space optimality of distributed depth-first token circulation algorithms. In *Proceedings of DIMACS Workshop on Distributed Data and Structures*, pages 91–106. Carleton University Press, 1999.

[PV00] F. Petit and V. Villain. Optimality and self-stabilization in rooted tree networks. *Parallel Processing Letters*, 10(1):3–14, 2000.

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

[Spi97] J. M. Spinelli. Self-stabilizing sliding window ARQ protocols. *IEEE-ACM Transactions on Networking*, 5(2):245–254, 1997.

[Taj77] W. D. Tajibnapis. A correctness proof of a topology information maintenance protocol for a distributed computer network. *Journal of the ACM*, 20(7):477–485, 1977.

[TB96] S. Tixeuil and J. Beauquier. Self-stabilizing token ring. In *ICSE96 International Conference on System Engineering*, 1996.

[Var93] G. Varghese. Self-stabilization by local checking and correction (Ph.D. thesis). Technical Report MIT/LCS/TR-583, MIT, 1993.

[Var00] G. Varghese. Self-stabilization by counter flushing. *SIAM Journal on Computing*, 30(2):486–510, 2000.

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