Gestion de groupe partitionnable dans les réseaux mobiles spontanés

Léon Lim

Doctorat en co-accréditation
Télécom SudParis et Université d'Évry-Val-d'Essonne

Spécialité : Informatique

École doctorale : Science et Ingénierie

Thèse présentée par

Léon Lim

Pour obtenir le grade de
Docteur de Télécom SudParis

Gestion de groupe partitionnable dans les réseaux mobiles spontanés

Soutenue le 29 Novembre 2012

Devant le jury composé de :

Président :	Pierre Sens	Professeur à l'Université Pierre et Marie Curie
Rapporteurs :	Pascal Felber	Professeur à l'Université de Neuchâtel
	Achour Mostefaoui	Professeur à l'Université de Nantes
Examinateurs :	André Schiper	Professeur à l'École Polytechnique Fédérale de Lausanne
	Guy Bernard	Professeur Émérite à Télécom SudParis (Directeur de thèse)
	Denis Conan	Maître de Conférences à Télécom SudParis (Encadrant)

Thèse n° 2012TELE0032

Mis en page avec la classe thloria-tsp adaptée pour Télécom SudParis.

Remerciements

J'adresse mes vifs remerciements à Monsieur Achour Mostefaoui, Professeur à l'Université de Nantes, et Monsieur Pascal Felber, Professeur à l'Université de Neuchâtel, pour avoir accepté d'être rapporteurs de ce travail. Je tiens également à remercier Monsieur Pierre Sens, Professeur à l'Université Pierre et Marie Curie, et Monsieur André Schiper, Professeur à l'École Polytechnique Fédérale de Lausanne, qui me font l'honneur de participer à ce jury.

Je remercie chaleureusement et profondément Monsieur Denis Conan, qui a encadré cette thèse. Il a toujours su m'indiquer les directions de recherche pertinentes à suivre. J'ai énormément appris à son contact et ai sincèrement apprécié travailler avec lui. Je lui suis très reconnaissant de ses conseils, de sa disponibilité, de sa patience et de son dynamisme qui m'ont été très profitables et qui m'ont permis d'avancer tout au long de cette thèse.

Je remercie également Monsieur Guy Bernard, mon directeur de thèse, pour m'avoir conseillé, encouragé et soutenu tout au long de la thèse avec patience et disponibilité, ainsi que pour la confiance qu'il m'a accordée.

Je remercie Monsieur Bruno Defude, le directeur du Département Informatique de Télécom SudParis, pour m'avoir accueilli au sein de son département et m'avoir permis d'effectuer cette thèse dans de bonnes conditions.

Je tiens à exprimer ma gratitude à Monsieur Michel Simatic et Madame Claire Lecocq pour avoir accepté de relire une grande partie de ce manuscrit.

Je tiens également à remercier l'ensemble du Département Informatique pour l'ambiance joyeuse qui y règne et pour les conditions de travail idéales qui m'ont permis de mener à bien cette thèse. Je voudrais remercier plus particulièrement les membres de l'équipe MARGE pour l'intérêt qu'ils ont porté à mes travaux de recherche, pour leur conseils, pour leur gentillesse et pour leur aide.

Je remercie mes amis pour leur soutien, leur présence et leur compréhension. Ils m'ont apporté les moments de réconfort et de distraction nécessaires lors du déroulement d'un tel projet.

Enfin, je souhaite remercier ma famille pour son soutien. Mes plus profonds remerciements vont à mes parents en France et au Cambodge. Ils m'ont offert l'occasion de faire des études longues, qui ont abouti à cette thèse.

Je dédie cette thèse
à ma défunte mère, Madame Sophany Lao.

Table des matières

Table des figures

Introduction

Contexte scientifique

Cette thèse s'inscrit dans le domaine des systèmes répartis construits au dessus des réseaux mobiles sans fil.

D'une manière très générale, deux catégories de réseaux mobiles sans fil existent : les réseaux cellulaires et les réseaux mobiles spontanés. Les réseaux cellulaires sont caractérisés par la présence de passerelles filaires ou sans fil (stations de bases) qui ont pour fonction de router les messages. Au contraire, les réseaux mobiles spontanés sont des réseaux auto-organisés composés uniquement de nœuds mobiles. Par la suite, nous nous focalisons sur les réseaux mobiles spontanés (en anglais, *Mobile Ad Hoc NETworks* ou MANETs).

Réseaux mobiles spontanés. Les MANETs permettent une nouvelle forme de communication et d'accès à l'information pour les utilisateurs nomades sans l'aide d'une infrastructure pré-existante et sans administration centralisée. De tels réseaux sont très dynamiques. Les défaillances, les déconnexions et la mobilité des nœuds peuvent momentanément isoler un nœud ou un groupe de terminaux du reste des participants à l'application répartie. Le partitionnement du réseau entraîne une dégradation de service, mais pas nécessairement une interruption de service. Les groupes de terminaux partitionnés doivent pouvoir continuer de fonctionner comme des systèmes répartis autonomes en offrant autant de services que possibles. [Forman and Zahorjan, 1994, Basile et al., 2003, Srivastava et al., 2008] présentent les caractéristiques majeures des MANETs. Cela inclut les problèmes de communication sans fil dus à la couche physique peu fiable, aux problèmes de routage, aux problèmes de sécurité des données, et aux limitations des ressources en termes d'énergie, de capacité de calcul et de stockage. Dans nos travaux, nous nous intéressons au problème du partitionnement du réseau.

Dans les MANETs, les nœuds communiquent entre eux à travers des chemins représentés par des séquences de liens sans fil établis entre les différents nœuds du réseau. Un lien est créé entre deux nœuds s'ils sont situés à une distance inférieure ou égale à leur portée de transmission. Les liens de communication unidirectionnels sont prédominants [Srivastava et al., 2008]. Par exemple, un nœud peut recevoir un message d'un autre nœud, mais à cause de son niveau de batterie insuffisant, ne peut pas y répondre. Ainsi, le graphe du réseau n'est pas nécessairement fortement connecté. C'est pourquoi les MANETs sont souvent mentionnés comme des réseaux mobiles spontanés à communications multisauts. Par conséquent, les nœuds ont besoin

de coopérer afin de maintenir la connectivité du réseau et chaque nœud peut jouer le rôle d'un routeur.

Réseaux partitionnables. À cause des arrivées, des départs, des défaillances et des mouvements des nœuds, les MANETs sont considérés comme des réseaux très dynamiques. [Srivastava et al., 2008] montre que le degré de changements de la topologie et du routage dans les MANETs est important même si la mobilité des nœuds est faible. Donc, les nœuds mobiles peuvent rejoindre et quitter une partition aussi arbitrairement que rapidement. Les défaillances, les déconnexions et les mouvements des nœuds peuvent momentanément isoler un nœud ou un groupe de terminaux du reste des participants à l'application répartie. Par conséquent, les MANETs sont des réseaux partitionnables : il existe des scénarios dans lesquels deux nœuds non défaillants ne peuvent pas communiquer entre eux. Plus précisément, le partitionnement du réseau correspond à la division du réseau en plusieurs groupes disjoints.

Le partitionnement du réseau est une caractéristique intrinsèque des MANETs. Les résultats des études expérimentales dans [Hähner and Dudkowski, 2007] confirment la fréquence importante de cette entrave. En plus du partitionnement du réseau provoqué par des événements imprévus (tels que les défaillances et les mouvements des nœuds), les applications doivent également supporter les opérations de déconnexion volontaire. En effet, un nœud mobile peut volontairement décider de se déconnecter du reste des participants afin d'exécuter les applications en local. Ce type de scénario est une cause supplémentaire de partitionnement du réseau.

Problème du partitionnement du réseau. De par leur nature, les applications réparties dans les MANETs impliquent la coopération de plusieurs nœuds mobiles exécutant des activités concurrentes sans mémoire physique partagée et avec partitionnement du réseau. Donc, les applications réparties robustes pour les MANETs doivent tolérer les partitionnements. Selon le théorème CAP (pour *Consistency*, *Availability* et *Partition-tolerance*) [Brewer, 2000, Gilbert and Lynch, 2002], un système réparti asynchrone sujet à des défaillances par arrêt franc ne peut pas fournir en même temps les trois propriétés suivantes :

- la cohérence : les données sont cohérentes et à jour ;
- la haute disponibilité : le service est toujours disponible ;
- la tolérance au partitionnement du réseau : l'application répartie progresse malgré le fait que certaines entités ne sont plus accessibles à cause d'un partitionnement du réseau.

La propriété de cohérence implique que chaque réponse est atomique même en cas d'accès concurrents. La propriété de haute disponibilité implique que chaque nœud ayant reçu une requête doit fournir une réponse même en cas de défaillance. La propriété de tolérance au partitionnement du réseau reflète la tolérance aux pertes de messages dans le réseau. Les trois paragraphes suivant précisent ces termes.

La cohérence d'un service réparti peut être définie par la notion d'objet ou donnée atomique [1].

1. « *D'une certaine manière, la cohérence atomique est différente de la cohérence d'une base de donnée respectant les propriétés ACID (en anglais,* Atomicity*,* Consistency*,* Isolation*, et* Durability*). En particulier, la cohérence dans ACID est une propriété requise pour les transactions d'une base de données tandis que la cohérence atomique est une propriété seulement requise pour une séquence d'opérations de requêtes/réponses.* » [Gilbert and Lynch, 2002].

L'atomicité des données est la condition requise pour les systèmes répartis qui demandent une propriété de cohérence forte. Pour satisfaire cette propriété, les opérations sur les données doivent être totalement ordonnées de telle sorte que chaque opération apparaît comme si elle était exécutée à un instant unique. Autrement dit, les requêtes sur la mémoire virtuelle partagée se comportent comme si elles étaient exécutées dans un seul fil d'exécution.

La disponibilité d'un service se traduit par le fait que chaque requête reçue par un nœud correct du système doit donner lieu à une réponse. En d'autres termes, l'algorithme réparti qui implante le service doit ultimement terminer. À première vue, cette condition pourrait correspondre à une version faible de la propriété de disponibilité puisqu'aucune borne supérieure sur le délai de terminaison de l'algorithme n'est imposée. Cependant, quand le système doit être tolérant au partitionnement du réseau, cette condition apparaît comme une propriété de disponibilité forte dans le sens où même si le réseau se partitionne, chaque requête doit donner lieu à une réponse.

La tolérance au partitionnement du réseau peut être modélisée comme la tolérance aux pertes arbitraires de messages. Quand le réseau se partitionne, tous les messages envoyés aux nœuds dans une autre partition sont perdus.

Tolérance au partitionnement. Si le système considéré est partiellement asynchrone alors il est possible d'assurer deux des trois propriétés C, A et P : par exemple, la tolérance au partitionnement du réseau tout en assurant un certain compromis entre la cohérence et la disponibilité. Autrement dit, la nature du partitionnement du réseau détermine la qualité de service des applications. Ainsi, le partitionnement du réseau entraîne une dégradation de service, mais pas nécessairement son interruption.

Dans les réseaux filaires, la solution classique pour tolérer le partitionnement du réseau est de masquer les défaillances de processus et de liens de communication qui sont considérées comme des événements rares. En revanche, dans les MANETs, les événements tels que les partitionnements sont fréquents et ne peuvent pas être traités d'une manière transparente. C'est pourquoi, une approche différente est utilisée qui consiste à exposer aux applications les événements du réseau. De telles applications sont alors dites sensibles au contexte, c'est-dire à l'environnement qui les entoure [Chen and Kotz, 2000, Coutaz et al., 2005].

Les applications sensibles au partitionnement sont une catégorie d'applications sensibles au contexte capables de s'adapter et de continuer leur exécution dans les situations dans lesquelles plusieurs partitions réseaux existent. Dans la littérature, les deux approches les plus utilisées pour faire face au problème du partitionnement sont : *1)* la prévision et *2)* la tolérance. Ce manuscrit s'inscrit dans la seconde approche. Pour montrer les différences entre ces deux approches, nous les présentons respectivement dans les deux paragraphes suivants.

L'approche de prévision du partitionnement tente de prédire le partitionnement pour permettre d'anticiper les reconfigurations. Plusieurs travaux adoptent cette stratégie « mieux vaut prévenir que guérir » [Roman et al., 2001, Milic et al., 2005, Wang and Baochun, 2002, Zhang et al., 2009]. Typiquement, les informations telles que le mouvement et l'emplacement des nœuds sont analysés avec l'idée générale que les nœuds mobiles qui appartiennent au même groupe adoptent le même patron de mouvement tandis que les nœuds de groupes différents

possèdent des patrons de mouvement différents. Par exemple, dans [Wang and Baochun, 2002, Milic et al., 2005], le mouvement d'un groupe est déterminé par les informations de position et de vitesse des nœuds. Dans [Roman et al., 2001, Zhang et al., 2009], la distance entre les nœuds est estimée à partir de la portée de transmission, de la vitesse et de la direction de chacun des nœuds, et les nœuds qui se trouvent à proximité les uns des autres sont considérés comme appartenant au même groupe.

L'approche de tolérance au partitionnement traite l'effet du partitionnement du réseau sur les applications réparties en unifiant tout le contexte du réseau sous la forme d'une abstraction : le service de communication de groupe partitionnable [Babaoğlu et al., 2001, Chockler et al., 2001, Khazan, 2004, Boulkenafed et al., 2005, Briesemeister and Hommel, 2006, Filali et al., 2006a, García et al., 2009]. Les applications sensibles au partitionnement sont alors programmées de telle sorte qu'elles puissent se reconfigurer elles-mêmes et ajuster leur comportement en utilisant la composition des nœuds du groupe courant comme une information de contexte. Pour cela, la perception de la composition du groupe doit être cohérente afin de ne pas compromettre les besoins fonctionnels des applications. C'est dans le contexte de la problématique des services de communication de groupe partitionnables dans les MANETs que s'inscrit cette thèse.

Problématique scientifique

D'un point de vue théorique, « un système réparti consiste en une collection de processus distincts qui sont spatialement séparés et communiquent les uns avec les autres en échangeant des messages » [Lamport, 1978]. Un processus peut représenter un ordinateur, un processus de l'ordinateur, c'est-à-dire un programme en cours d'exécution, ou un fil d'exécution (en anglais, *thread*). Un système composé d'un ensemble de processus est dit réparti lorsque le délai de transfert de messages n'est pas négligeable par rapport à la durée d'exécution d'un pas / d'une action au sein d'un processus.

Un système réparti est dit complètement asynchrone lorsqu'aucune propriété temporelle n'est garantie [Dolev et al., 1987, Dwork et al., 1988, Guerraoui and Schiper, 1997, Aguilera, 2010] : il n'y pas de borne sur la durée des actions locales ni sur le délai de transfert des messages. Les applications réparties robustes doivent évoluer même si le système peut subir des défaillances, c'est-à-dire qu'un ou plusieurs processus, ou un ou plusieurs liens défaillent. Comme il est impossible de distinguer un processus lent d'un processus qui a défailli dans un système asynchrone, la construction des systèmes répartis est difficile. Pour aider la construction des applications réparties, les chercheurs ont proposés des blocs de construction qui fournissent différents services. Les systèmes de communication de groupe font partie de ces blocs de construction de base.

Un service de communication de groupe (en anglais, *Group Communication System* ou GCS) comprend un service de diffusion dans un groupe et un service de gestion du groupe. Le premier service fournit la diffusion fiable des messages aux membres du groupe. Le second service fournit la vue d'un processus sur l'ensemble des processus avec lesquels il peut échanger des messages. L'objectif de ce service est de proposer une séquence cohérente de vues cohérentes. Pour ce faire, la gestion de groupe s'appuie sur un algorithme de consensus entre les membres. Dans cette thèse, nous nous focalisons sur la gestion de groupe.

Il existe deux types de gestion de groupe : partition primaire et partitionnable. Dans la gestion de groupe de partition primaire, seul un groupe existe. Dans la gestion de groupe partitionnable, plusieurs groupes disjoints évoluent concurremment et indépendamment les uns des autres. Comme le partitionnement du réseau est une caractéristique intrinsèque des MANETs, nous étudions la gestion de groupe partitionnable. Concernant la gestion de groupe partitionnable, bien que le problème soit étudié depuis longtemps [Anceaume et al., 1995, Chandra et al., 1996b], il constitue toujours un défi pour des réseaux dynamiques avec forte volatilité des nœuds et est toujours au cœur de nombreux travaux théoriques [Chockler et al., 2001, Babaoğlu et al., 2001, Schiper and Toueg, 2006, Boulkenafed et al., 2005, Filali et al., 2006a, Filali et al., 2006b, Pleisch et al., 2008], et expérimentaux [Roman et al., 2001, Franceschetti and Bruck, 2001, García et al., 2009].

Cette thèse se veut une contribution à la gestion de groupe partitionnable en environnement réseau très dynamique, mettant plus particulièrement en œuvre des MANETs.

Plan de la thèse

Ce manuscrit présente l'ensemble des travaux et des résultats obtenus pendant ma thèse. Il est organisé comme suit.

Dans le premier chapitre de la thèse, nous analysons l'état de l'art des spécifications de la gestion de groupe partitionnable et présentons les problèmes identifiés dans ces spécifications. Aucune spécification de la littérature ne satisfait les deux buts antagonistes suivants : *1)* la spécification doit être assez forte pour faciliter la conception des applications réparties dans les systèmes partitionnables ; et *2)* elle doit être assez faible pour être résoluble (implantable).

Dès lors, il nous semble intéressant de chercher une solution qui corresponde au meilleur compromis possible entre ces deux exigences. La solution proposée doit permettre la perception cohérente de la composition du groupe afin de ne pas compromettre les besoins fonctionnels des applications tout en fournissant une qualité de service acceptable et en restant résoluble.

Dans le deuxième chapitre de la thèse, nous définissons un modèle de système qui caractérise la dynamicité des MANETs. Contrairement aux systèmes statiques, dans les systèmes dynamiques, les processus ne possèdent *a priori* pas de connaissance sur l'ensemble des participants du système. Les paramètres tels que le nombre total de participants ainsi que le nombre maximal de processus pouvant être défaillants ne sont pas connus. Dans nos travaux, nous considérons les systèmes dynamiques partitionnables avec recouvrement dans lesquels nous définissons une condition de stabilité faible du système : seulement un ensemble de α processus stables est requis pour exécuter l'application répartie dans une partition. Les α processus sont sélectionnés via un critère de stabilité. Intuitivement, α exprime le compromis entre l'accord et la progression parmi les participants d'un groupe. Cette condition de stabilité faible rend notre spécification implantable et permet de capturer la dynamicité du système tout en autorisant l'application répartie à exprimer ses besoins fonctionnels.

Dans le troisième chapitre de la thèse, nous exposons le problème de la gestion de groupe de partition primaire résolu en le transformant en une séquence de consensus, où chaque consensus est exécuté par les processus de la vue courante et la décision retournée par

le consensus est l'ensemble des processus de la vue suivante [Guerraoui and Schiper, 2001, Schiper, 2006, Schiper, 2004]. Parmi les solutions de la littérature, Paxos permet la gestion de groupe de partition primaire en mettant en œuvre de façon native une séquence de consensus [Lamport, 1998, Lamport, 2001]. Notre intuition de départ est que, puisque le consensus peut être utilisé pour résoudre le problème de la gestion de groupe de partition primaire, un autre type de consensus peut permettre de trouver une solution pour la gestion de groupe partitionnable.

Dans le quatrième chapitre de la thèse, nous décrivons les principes d'utilisation de Paxos pour la gestion de groupe partitionnable. Nous y présentons l'architecture logicielle de la gestion de groupe partitionnable à base de consensus abandonnable. Le consensus abandonnable est une combinaison de deux modules qui sont le détecteur ultime des α participants d'une partition $\diamondsuit\mathcal{PPD}$ et le registre ultime par partition $\diamondsuit\mathcal{RPP}$. $\diamondsuit\mathcal{PPD}$ capture la vivacité dans une partition même si la partition n'est pas complètement stable en détectant ultimement les α participants stables dans une partition tandis que $\diamondsuit\mathcal{RPP}$ préserve la sûreté dans la même partition en matérialisant une mémoire stable partagée par les participants de la partition. Le résultat est une spécification de la gestion de groupe partitionnable adaptée pour les systèmes dynamiques partitionnables tels que les MANETs.

Dans le cinquième chapitre, nous présentons une implantation des abstractions présentées. Nous implantons chacun des modules $\diamondsuit\mathcal{PPD}$ et $\diamondsuit\mathcal{RPP}$ de façon indépendante. Ensuite, l'implantation du consensus abandonnable consiste à utiliser ces deux modules comme blocs de construction. Enfin, munis du module de consensus abandonnable, nous implantons la gestion de groupe partitionnable. Chacune des implantations est prouvée.

Dans le dernier chapitre de la thèse, nous évaluons les performances du détecteur $\diamondsuit\mathcal{PPD}$ par simulation en utilisant différents modèles de mobilité. Les modèles de mobilité sont sélectionnés via des critères généraux puis spécifiques aux réseaux partitionnables. Les simulations ont pour objectif de montrer la validation de notre approche dans différents contextes d'exécution.

Nous terminons ce manuscrit en présentant une synthèse générale et quelques perspectives à nos travaux.

Chapitre 1

État de l'art de la gestion de groupe partitionnable

Sommaire

1.1 Introduction

Un système de communication de groupe est un service de base des systèmes répartis. C'est un intergiciel qui fournit un moyen de communication multipoint à multipoint en organisant les processus dans des groupes [Chockler et al., 2001]. Par exemple, un groupe peut être un

ensemble de processus qui coopèrent afin de réaliser une tâche commune, un ensemble de processus qui partagent des intérêts communs, ou simplement un ensemble de processus corrects et opérationnels. Chaque groupe est identifié par un nom logique. Le système de communication de groupe comprend un service de gestion de groupe et un service de diffusion de messages dans le groupe. Pour chaque processus, la gestion de groupe fournit la vue sur l'ensemble des processus dans le groupe. Cette vue est dynamique et l'objectif de ce service est d'assurer la « cohérence » des vues. Pour ce faire, la gestion de groupe s'appuie sur un algorithme d'accord entre les membres. Le deuxième service fournit la diffusion de messages aux membres du groupe en respectant, par exemple, des propriétés de fiabilité et d'ordre. Dans nos travaux, nous nous intéressons à la gestion de groupe en environnement réseau très dynamique tel que les MANETs.

Dans la littérature, il existe deux types de services de gestion de groupe : *1)* primaire et *2)* partitionnable. Intuitivement, la gestion de groupe dite primaire ou de partition primaire maintient une vue unique de la composition courante du groupe. Dans ces systèmes, seuls les processus de la partition primaire incluant une majorité des membres du système peuvent continuer à travailler. Les processus corrects [2] se mettent d'accord sur une séquence de vues unique de la partition primaire [Anceaume et al., 1995, Chandra et al., 1996b, Schiper, 2006]. La gestion de groupe de partition primaire requiert des hypothèses de synchronie fortes sur le système afin de satisfaire la propriété d'accord. Selon [Chandra et al., 1996b], la gestion de groupe de partition primaire est impossible à résoudre dans un système asynchrone pouvant subir des défaillances par arrêts francs. Cela est dû au fait que les processus doivent se mettre d'accord sur le contenu de la $n^{\text{è}}$ vue d'un groupe. Le résultat d'impossibilité reste valable même si les processus qui sont faussement détectés comme étant défaillants peuvent être retirés. Contrairement aux systèmes de partition primaire, dans les systèmes dits partitionnables, plusieurs partitions évoluent concurremment et indépendamment les unes des autres [Anceaume et al., 1995, Chockler et al., 2001, Babaoğlu et al., 2001]. Plusieurs ensembles de processus disjoints co-existent tels que les processus dans chaque ensemble se mettent d'accord sur les membres corrects courants de leur partition. En autorisant les opérations `fusion` et `séparation` ainsi que l'exécution concurrente dans des groupes distincts, la gestion de groupe partitionnable échappe au résultat d'impossibilité de [Chandra et al., 1996b]. Cependant, la gestion de groupe partitionnable est confrontée à un autre problème fondamental. Elle doit faire face à deux buts antagonistes [Anceaume et al., 1995, Pleisch et al., 2008] : la spécification doit être *i)* assez forte pour éviter des implantations triviales et inutiles n'aidant pas à la conception d'applications réparties tolérantes au partitionnement et *ii)* assez faible pour être implantable.

La gestion de groupe pour les systèmes partitionnables, bien qu'étant étudiée depuis longtemps [Anceaume et al., 1995], constitue toujours un défi dans les réseaux dynamiques avec volatilité des nœuds, et est toujours au cœur de nombreux travaux théoriques [Chockler et al., 2001, Babaoğlu et al., 2001, Schiper and Toueg, 2006, Pleisch et al., 2008], et expérimentaux [Roman et al., 2001, Franceschetti and Bruck, 2001, García et al., 2009].

Comme indiqué dans [Anceaume et al., 1995], les nombreuses spécifications de GCS uti-

2. Dans une exécution, les processus qui ne défaillent pas sont dits corrects.

lisent des terminologies et des notations différentes qui compliquent l'étude du sujet. [Chockler et al., 2001] est un état de l'art des GCS ayant pour but d'unifier les propriétés des GCS existants. Dans la suite de ce chapitre, pour l'expression des propriétés, nous adoptons la terminologie introduite dans [Chockler et al., 2001]. Concernant la décomposition architecturale du système réparti, nous utilisons la terminologie de [Guerraoui and Rodrigues, 2006] avec par exemple les concepts de composant et événement.

Ce chapitre est organisé comme suit. Dans la section 1.2, nous présentons le modèle de système réparti considéré pour décrire le système intergiciel de communication de groupe. Nous présentons l'intergiciel de communication de groupe dans la section 1.3. Dans la section 1.4, nous étudions deux spécifications de référence de la gestion de groupe partitionnable et discutons des problèmes de ces spécifications. Enfin, nous concluons ce chapitre dans la section 1.5.

1.2 Modèle de système réparti

Cette section a pour objectif d'introduire les concepts utiles nous permettant de modéliser le système et de raisonner sur le comportement de ce dernier. Dans les sections 1.2.1 et 1.2.2, nous définissons respectivement les processus qui communiquent entre eux en échangeant des messages et ces messages. Puis, dans la section 1.2.3, nous décrivons la notion d'événement. Ensuite, dans la section 1.2.4, nous présentons la notion de pile logicielle. Enfin, dans la section 1.2.5, nous présentons différentes notions de base telles que l'exécution, l'histoire globale et l'ordre causal.

1.2.1 Processus

Le système réparti considéré consiste en un ensemble $\mathbb{P}$ de processus identifiés de manière unique qui exécutent des programmes. Les processus sont autonomes et coopèrent pour atteindre un objectif commun. À moins d'être défaillant par arrêt franc, un processus est supposé exécuter l'algorithme qui est associé à son programme. L'exécution s'opère à travers un ensemble de composants qui implantent l'algorithme au sein du processus. Dans notre étude, l'unité de défaillance est le processus. Un processus est dit correct dans une exécution s'il ne défaille pas dans cette exécution. Nous nous limitons aux défaillances par arrêt franc : quand un processus défaille, tous ses composants défaillent également et en même temps. Chaque processus peut accéder à une mémoire locale stable qui permet au processus de sauvegarder une partie (ou la totalité) de son état. Cela permet au processus de redémarrer (avec le même identifiant) après une défaillance en recouvrant son état. Il s'agit du modèle défaillance avec recouvrement (en anglais, *crash-recovery*). Il est important de noter que, contrairement au modèle dynamique/sans recouvrement/avec partitionnement, le modèle dynamique/avec recouvrement/avec partitionnement est très peu considéré dans les spécifications des GCS [Babaoğlu et al., 2001, Chockler et al., 2001, Schiper, 2004]. Le principe de base du système de communication de groupe est de fournir la tolérance aux fautes par réplication plutôt que par capture des états locaux pouvant servir de points de reprise en cas de défaillances ultérieures. Cependant, comme indiqué dans [Schiper, 2004], les points de reprise nous permettent de tolérer les défaillances catastrophiques comme par exemple la défaillance simultanée de toutes les répliques. Par conséquent, dans nos travaux, le modèle de système réparti

qui nous intéresse est le modèle dynamique/avec recouvrement/avec partitionnement.

1.2.2 Messages

Les processus communiquent en échangeant des messages pris parmi l'ensemble $\mathbb{M}$ des messages possibles via des liens de communication. Quand ce n'est pas précisé explicitement, les liens de communication sont supposés non fiables : les messages peuvent être perdus. Mais, les messages qui sont reçus par les processus sont supposés non corrompus. En outre, il n'existe pas de borne sur le délai de transmission d'un message. Nous supposons qu'il existe une borne supérieure et une borne inférieure sur la durée d'exécution d'un pas (en nombre d'événements exécutés par unité de temps) des processus corrects. Ainsi, pour simplifier la présentation et sans perdre en généralité, nous supposons que les exécutions locales ne prennent pas de temps. Seuls les transferts de messages prennent du temps. Ainsi, le système est asynchrone. Les messages sont identifiés de manière unique, par exemple, par l'utilisation de l'identité et d'un numéro de séquence posé par le processus expéditeur du message.

1.2.3 Événements

Les processus du système réparti exécutent le même algorithme. L'union de ces algorithmes locaux constitue l'algorithme réparti. Un automate est associé à chaque processus. Cet automate représente le comportement du processus. Un algorithme réparti est donc un ensemble d'automates avec un automate pour chaque processus. $\mathbb{E}$ est l'ensemble des événements qui inclut au moins les événements de types *send* et *receive*. Dans la suite de ce chapitre, l'ensemble $\mathbb{E}$ est enrichi tout au long de la description des propriétés de GCS.

1.2.4 Pile logicielle

Nous utilisons le modèle de composants orienté événement de [Guerraoui and Rodrigues, 2006] pour spécifier les interfaces et les propriétés ainsi que les algorithmes. Dans ce modèle, chaque processus est composé d'un ensemble de modules logiciels, appelés composants. Chaque composant est identifié par son nom et est caractérisé par une interface qui expose un ensemble de propriétés. Le composant fournit une interface avec différents types d'événements que le composant peut accepter et produire. Un composant peut être de type primitif ou composite. Dans ce dernier cas, le composant correspond à un assemblage d'autres composants (composites ou primitifs). Les composants sont destinés à être utilisés ensemble afin de satisfaire des propriétés. Par exemple, les composants peuvent être composés pour construire des piles logicielles telles que montrées dans la figure 1.1. Dans chaque processus, un composant représente une couche spécifique dans la pile. La couche application constitue le sommet de la pile tandis que la couche réseau est positionnée au bas de la pile. Les couches correspondant aux abstractions du système réparti sont typiquement celles qui se trouvent entre les couches application et réseau. Dans la figure 1.1, l'abstraction d'un service réparti, par exemple le GCS, est représenté par un composant composite.

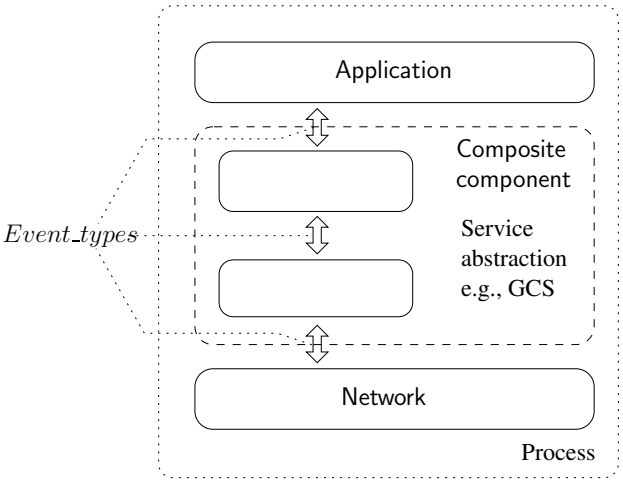

FIGURE 1.1 – Pile logicielle.

1.2.5 Exécution, histoire globale et ordre causal

L'exécution d'un processus est modélisé par une séquence d'événements. L'histoire locale du processus p durant une exécution est une séquence d'événements ou d'absences d'événement (chaque absence étant notée ϵ) $h_p = (e_p^1 e_p^2 e_p^3 \dots)$, où e_p^i correspond au i^e événement du processus p. L'ordre des événements dans chaque processus p est une énumération canonique qui correspond à l'ordre total imposé par l'exécution séquentielle des événements locaux à p. Un préfixe initial de l'histoire locale du processus p contenant k événements est dénoté par $h_p^k = (e_p^1 e_p^2 e_p^3 \dots h_p^k)$. h_p^0 correspond à une séquence vide, c'est-à-dire $h_p^0 = ()$. Dans la suite, l'indice p et l'exposant i sont omis lorsque le contexte le permet : par exemple, dans la définition du prédicat $alive(p)$ à la page 19, l'événement e^j concerne, selon le contexte, le processus p et l'indice p est omis.

Nous considérons que le système asynchrone ne possède pas d'horloge globale. Cependant, afin de faciliter la présentation et sans perdre en généralité, nous considérons une horloge globale qui n'est pas accessible aux processus. $\mathbb{T}$ est l'ensemble des tics d'horloge. $\mathbb{T}$ est inclus dans l'ensemble des entiers naturels $\mathbb{N}$.

L'histoire globale d'une exécution de l'algorithme réparti est une fonction $H : \mathbb{P} \times \mathbb{T} \to \mathbb{E} \cup \epsilon$. Si p exécute un événement $e \in \mathbb{E}$ à l'instant t alors $H(p,t) = e$ sinon $H(p,t) = \epsilon$. $H(p,t) = \epsilon$ signifie que p n'exécute aucun événement à l'instant t. Soit un intervalle $I \subseteq \mathbb{T}$, nous écrivons $e \in H(p,I)$ si p exécute l'événement e dans l'intervalle I dans l'histoire globale H, c'est-à-dire $\exists t \in I : H(p,t) = e$.

L'ordre causal des événements d'une exécution est basé sur la notion de « cause à effet ». Dans l'histoire globale, deux événements sont reliés par la relation de précédence causale définie ci-dessous si le premier provoque l'apparition du second. L'ordre des événements est défini par la relation binaire de précédence causale, notée $\rightarrow$, introduite par [Lamport, 1978] : soient e_p^k, e_q^l, e_p^l trois événements,

$$e_p^k \in h_p \wedge e_q^l \in h_q \wedge e_p^k = e_q^l \implies p = q \wedge k = l$$

$\wedge\ e_p^k, e_p^l \in h_p \wedge k < l \implies e_p^k \rightarrow e_p^l$
$\wedge\ e' = \texttt{send}(p, m) \wedge e = \texttt{receive}(q, m) \implies e' \rightarrow e$
$\wedge\ e' \rightarrow e'' \wedge e'' \rightarrow e \implies e' \rightarrow e$

Cependant, il peut exister des événements dans l'histoire globale qui ne sont pas liés causalement, c'est-à-dire que nous n'avons ni $e' \rightarrow e$ ni $e \rightarrow e'$. De tels événements sont dits concurrents et sont dénotés par $e' \parallel e$. Par conséquent, l'ordre des événements est partiel.

La section suivante présente la terminologie et les notations de base d'un système intergiciel de communication de groupe.

1.3 Système intergiciel de communication de groupe

Dans la section 1.3.1, nous présentons les événements de base du GCS. Puis, dans la section 1.3.2, nous présentons l'interaction entre les composants locaux dans un même processus. Ensuite, dans la section 1.3.3, nous décrivons la notation utilisée pour présenter les définitions préliminaires de la gestion de groupe.

1.3.1 Terminologie

Tout d'abord, rappelons les quatre ensembles suivants :
- $\mathbb{P}$: l'ensemble des processus p, q, r, etc., défini dans la section 1.2.1 ;
- $\mathbb{M}$: l'ensemble des messages m, m_1, m_2, m_3, etc., défini dans la section 1.2.2 ;
- $\mathbb{V}$: l'ensemble des vues $v', v'', v, v_1, v_2, v_3$, etc., associés aux événements `view_change`. Une vue v est représentée par la paire $(v.membres, v.id)$, où $v.membres$ est l'ensemble des membres de cette vue et $v.id$ est l'identifiant de la vue ;
- $\mathbb{E}$: l'ensemble des événements défini dans la section 1.2.3.

L'ensemble $\mathbb{E}$ est enrichi pour devenir l'ensemble des événements suivants :
- $\{\texttt{send}(p, m) | p \in \mathbb{P}, m \in \mathbb{M}\}$ [3] ;
- $\{\texttt{receive}(p, m) | p \in \mathbb{P}, m \in \mathbb{M}\}$ [4] ;
- $\{\texttt{view\_change}(p, v, TS) | p \in \mathbb{P}, v \in \mathbb{V}, TS \in 2^{\mathbb{P}}\}$, où v est la nouvelle vue et TS est l'ensemble transitionnel qui correspond à un sous-ensemble de l'intersection entre les membres de la vue courante et ceux de la nouvelle vue v. La notion d'ensemble transitionnel est expliquée dans la section 1.4.5.1 qui présente la propriété de synchronie virtuelle du service de diffusion de messages dans le groupe ;
- $\{\texttt{crash}(p, m) | p \in \mathbb{P}\}$;
- $\{\texttt{recover}(p) | p \in \mathbb{P}\}$.

Observons que le premier argument de chaque événement dans $\mathbb{E}$ est un processus dans $\mathbb{P}$. Nous définissons donc la fonction $pid : \mathbb{E} \rightarrow \mathbb{P}$ qui retourne l'identifiant du processus associé à un événement.

3. Dans [Chockler et al., 2001], l'événement de type *send* est utilisé pour l'envoi ou la diffusion d'un message.
4. Dans [Chockler et al., 2001], l'événement de type *receive* correspond à la réception ou à la livraison d'un message.

1.3.2 Interface

La figure 1.2 représente l'interface d'un GCS. Les événements de types *send*, *receive* et *view_change* sont échangés entre l'application et le GCS. L'application utilise le GCS pour envoyer ou diffuser des messages ainsi que livrer les notifications de changement de vue. Les événements de types *crash* ainsi que *recover* correspondent à l'interaction entre le processus et l'environnement.

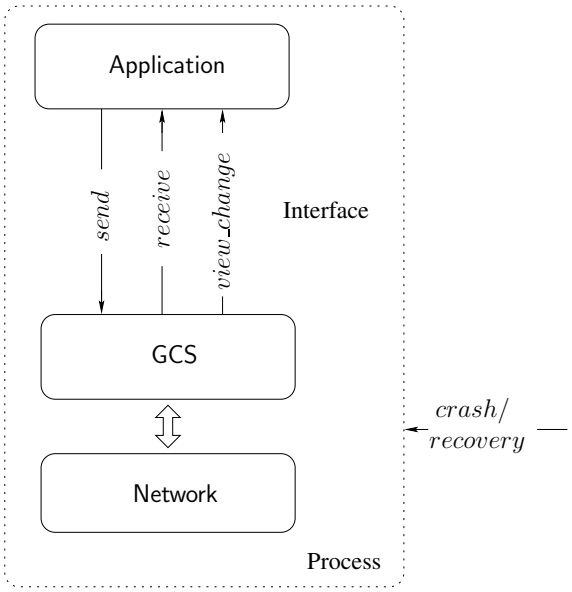

FIGURE 1.2 – Interface du GCS.

1.3.3 Définitions

Le traitement (l'exécution) d'un événement intervient dans le contexte d'une vue. La fonction $viewof : \mathbb{E} \rightarrow \mathbb{V} \cup \perp$ donne la vue dans laquelle un événement est exécuté. $\perp$ correspond à une vue vide et signifie que l'événement n'est exécuté dans aucune vue : par exemple, à l'initialisation ou suite à l'événement de recouvrement, un processus n'est pas considéré appartenir à une vue. La fonction *viewof* est définie comme suit [Chockler et al., 2001] :

Définition 1. viewof. *La vue de l'événement e exécuté par le processus p est la vue livrée par p dans l'événement $e' =$* `view_change` *qui précède e et tel qu'il n'existe aucun événement e'' (qui correspond à l'événement* `view_change` *ou* `crash`*) de p entre e' et e. La vue est $\perp$ s'il n'existe pas un tel événement e'. Formellement,*

$$viewof(e) \stackrel{\text{déf}}{=} \begin{cases} v \quad si\ \exists t\ \exists t'\ \exists TS\ \exists e'\ \nexists e''\ \nexists t'' : \\ \qquad H(pid(e),t) = e \\ \qquad \wedge\ H(pid(e),t') = e' = \texttt{view\_change}(pid(e), v, TS) \\ \qquad \wedge\ t' < t'' < t \\ \qquad \wedge \Big(H(pid(e),t'') = e'' = \texttt{crash}(pid(e)) \\ \qquad\quad \vee\ \exists TS'\ \exists v' : e'' = H(pid(e),t'') \\ \qquad\qquad \wedge\ e'' = \ \texttt{view\_change}\ (pid(e), v', TS')\Big) \\ \bot \quad sinon \end{cases}$$

Nous reprenons aussi les prédicats suivants de [Chockler et al., 2001] :

- le processus p reçoit le message m :
  $receive(p,m) \stackrel{\text{déf}}{=} \exists t : H(p,t) = \texttt{receive}(p,m)$
- le processus p reçoit le message m dans la vue v :
  $receive\_in(p,m,v) \stackrel{\text{déf}}{=} \exists t : H(p,t) = \texttt{receive}(p,m) = e \wedge viewof(e) = v$
- le processus p envoie le message m :
  $send(p,m) \stackrel{\text{déf}}{=} \exists t : H(p,t) = \texttt{send}(p,m)$
- le processus p envoie le message m dans la vue v :
  $send\_in(p,m,v) \stackrel{\text{déf}}{=} \exists t : H(p,t) = \texttt{send}(p,m) = e \wedge viewof(e) = v$
- le processus p installe la vue v :
  $install(p,v) \stackrel{\text{déf}}{=} \exists t\ \exists TS : H(p,t) = \texttt{view\_change}(p,v,TS)$
- le processus p installe la vue v dans la vue v' :
  $install\_in(p,v,v') \stackrel{\text{déf}}{=} \exists t\ \exists TS : H(p,t) = \texttt{view\_change}(p,v,TS) = e \wedge viewof(e) = v'$
- l'événement e' du processus p est l'événement précédant l'événement e du processus p :
  $previous\_event(p,e',e) \stackrel{\text{déf}}{=} \exists t\ \exists t'\ \nexists t'' : t' < t'' < t \wedge H(p,t') = e' \wedge H(p,t) = e \wedge H(p,t'') = e''$
- l'événement e' du processus p est l'événement suivant l'événement e du processus p :
  $next\_event(p,e',e) \stackrel{\text{déf}}{=} \exists t'\ \exists t\ \nexists t'' : t < t'' < t' \wedge H(p,t') = e' \wedge H(p,t) = e \wedge H(p,t'') = e''$
- le processus p est défaillant dans la vue v :
  $crash\_in(p,v) \stackrel{\text{déf}}{=} \exists t : H(p,t) = \texttt{crash}(p) = e \wedge viewof(e) = v$

Nous considérons qu'il n'existe pas d'événement entre deux événements `crash` et `recovery` dans l'histoire d'un processus. Cette propriété d'intégrité d'exécution est définie comme suit.

Propriété 1. Intégrité d'exécution. *L'événement dans un processus après l'événement* `crash` *est l'événement* `recovery` *et l'événement avant l'événement* `recovery` *est un événement* `crash`. *Formellement,*

$\Big(next\_event(p,e,e') \wedge e' = \texttt{crash}(p) \Rightarrow e = \texttt{recovery}(p)\Big)$

$\wedge\ \Big(e' = \texttt{recovery}(p) \Rightarrow \exists e : previous\_event(p,e,e') \wedge e = \texttt{crash}(p)\Big)$.

La section suivante présente l'état de l'art des spécifications de la gestion de groupe partitionnable. En particulier, nous présentons les deux spécifications de référence de la littérature.

1.4 Spécifications de la gestion de groupe partitionnable

Dans la gestion de groupe partitionnable, plusieurs groupes disjoints évoluent concurremment et indépendamment les uns des autres. La gestion de groupe partitionnable doit gérer de façon cohérente les changements dynamiques de la composition des groupes. Chaque processus possède une vue locale (fournie par le service de gestion de groupe). Les processus qui sont mutuellement connectés [5] dans le groupe doivent se mettre d'accord sur les vues à installer.

Cette section est organisée comme suit. Dans les sections 1.4.1 et 1.4.2, nous présentons respectivement les propriétés de sûreté et de vivacité de la gestion de groupe partitionnable. Ensuite, dans les sections 1.4.3 et 1.4.4, nous distinguons deux conditions de stabilité forte et faible du système qui permettent de garantir les propriétés de vivacité. Enfin, dans la section 1.4.5, nous présentons et discutons des problèmes des deux spécifications de référence de la gestion de groupe partitionnable.

1.4.1 Propriétés de sûreté

Les trois premières propriétés fournissent des garanties de base concernant l'installation de vues. Elles n'imposent pas l'ordre sur les vues installées. Remarquons que ces propriétés sont aussi satisfaites par la gestion de groupe de partition primaire.

Dans la première propriété qui suit, l'auto-inclusion, nous souhaitons que la gestion de groupe partitionnable informe un processus seulement des vues dont il est membre. En effet, la vue reflète la capacité des processus dans le groupe à communiquer entre eux, et un processus est toujours capable de communiquer avec lui-même.

Propriété 2. Auto-inclusion. *Si le processus p installe la vue v alors p est un membre de v. Formellement, $install(p, v) \Rightarrow p \in v.members$.*

Dans la seconde propriété, la monotonicité locale, nous imposons un ordre strictement croissant sur les identifiants des vues installées par un processus. Ceci implique les deux conséquences suivantes : *1)* un processus installe une vue au plus une fois et *2)* si deux processus installent deux mêmes vues alors ils les installent dans le même ordre. Lorsqu'un processus se recouvre après une défaillance, il installe la dernière vue qui a été sauvegardée dans sa mémoire stable locale. En outre, il existe plusieurs façons de générer les identifiants de vues. Certaines solutions sont présentées dans [Chockler et al., 2001].

Propriété 3. Monotonicité locale. *Si le processus p installe la vue v après avoir installé la vue v' alors l'identifiant de v est plus grand que celui de v'. Formellement,*
$\Big(H(p,t) = \texttt{view\_change}(p, v, TS)) \quad \wedge\ H(p,t') = \texttt{view\_change}(p, v', TS') \wedge t > t'\Big)$
$\Rightarrow v.id > v'.id$.

La propriété qui suit, l'événement de vue initiale, impose que tout événement doit être exécuté dans une vue donnée. Pour satisfaire cette propriété, le comportement de l'application

5. Dans [Chockler et al., 2001], les processus « mutuellement connectés » (en anglais, *mutually connected processes*) correspondent aux processus qui peuvent communiquer entre eux.

doit être spécifié de telle manière qu'un événement applicatif `send` ou `receive` n'apparaisse pas avant le premier événement `view_change`. La vue initiale peut être fournie de deux façons. Soit elle est déterminée, comme pour n'importe quelle autre vue, par le service de gestion de groupe partitionnable. Dans ce cas, aucune connaissance préalable sur les autres processus n'est nécessaire. Soit chaque processus décide de la vue initiale du groupe indépendamment des autres processus (c'est-à-dire sans communiquer avec les autres processus). Dans ce cas, la vue initiale peut être soit une vue singleton, soit une vue qui contient les processus supposés exister.

Propriété 4. Événement de vue initiale. *Chaque événement* `send`, `receive` *est exécuté dans le contexte d'une vue donnée. Formellement,*
$\Big(e = \texttt{send}(p, m) \vee e = \texttt{receive}(p, m)\Big) \Rightarrow viewof(e) \neq \perp$.

1.4.2 Propriétés de vivacité

Seules, les propriétés de sûreté ne sont pas suffisantes pour définir des abstractions utiles. En effet, les propriétés de sûreté peuvent être satisfaites par des implantations triviales qui consistent, par exemple, à ne rien faire. Donc, il est nécessaire d'ajouter des propriétés de vivacité afin d'assurer que des actions utiles se produisent ultimement. Concernant la gestion de groupe partitionnable, la définition de propriétés de vivacité qui soient suffisamment faibles pour être implantables et suffisamment fortes pour être utiles, c'est-à-dire pour faciliter la mise en place des applications réparties, est un défi.

Idéalement, le service de gestion de groupe partitionnable doit fournir des informations (vues) précises. Il doit livrer des vues qui correspondent à la situation des processus corrects et connectés au sein d'une partition réseau. Ce comportement idéal ne peut pas être garanti dans toutes les exécutions. Les propriétés de vivacité de la gestion de groupe partitionnable ne peuvent être satisfaites que sous certaines conditions d'exécution. Ces dernières correspondent aux conditions de stabilité du réseau qui sont externes à l'implantation du service de gestion de groupe partitionnable. À cause de l'asynchronisme du modèle et de l'instabilité du réseau, le service de gestion de groupe peut livrer des vues différentes aux membres d'un groupe.

Les vues livrées aux membres d'un groupe durant les périodes instables du système ne constituent pas un problème car « *ce qui est important est que le service de gestion de groupe partitionnable fournisse la même vue précise aux membres du groupe*[6] *lorsque la composition du groupe se stabilise.* » [Khazan, 2004].

Rappelons que, comme indiqué dans [Chockler et al., 2001], il est impossible d'implanter ce service de gestion de groupe partitionnable précis dans les systèmes asynchrones pouvant subir des défaillances. Pour contourner ce problème, les travaux de la littérature [Chockler et al., 2001, Babaoğlu et al., 2001] proposent d'enrichir les systèmes répartis avec différentes versions du détecteur de défaillances non fiable $\Diamond\mathcal{P}$ [Chandra and Toueg, 1996][7]. Cela aboutit aux deux so-

6. Notons que [Khazan, 2004] considère l'existence d'un groupe unique.

7. $\Diamond\mathcal{P}$ est un détecteur de défaillances « ultimement parfait ». Il possède les propriétés suivantes : *1)* la *complétude forte* stipulant que tout processus qui défaille est ultimement suspecté de façon permanente par tous les processus corrects et *2)* la *précision forte ultime* imposant qu'il existe un instant après lequel les processus corrects ne sont pas suspectés d'être défaillants par les processus corrects.

lutions suivantes. Dans la première, les propriétés de vivacité sont garanties seulement pour les exécutions dans lesquelles le réseau est ultimement complètement stable : intuitivement, le réseau est ultimement complètement stable s'il existe un instant t après lequel aucun processus n'est défaillant ou ne se recouvre, il n'y a pas de perte de message, et aucun changement dans la topologie du réseau ne se produit. De telles propriétés de vivacité sont proposées par [Chockler et al., 2001], et nous les présentons dans la section 1.4.3. La seconde solution, proposée par [Babaoğlu et al., 2001], est présentée dans la section 1.4.4. À la différence de la solution de [Chockler et al., 2001], la vivacité n'est pas seulement garantie dans les partitions complètement stables, mais dans toutes les partitions. En ce sens, la spécification de [Babaoğlu et al., 2001] fournit des garanties de vivacité plus fortes que celles de la spécification de [Chockler et al., 2001] car elle accepte une plus faible stabilité du réseau.

Dans les sections qui suivent, nous présentons les deux solutions, en mettant l'accent sur la modélisation des liens du réseau.

1.4.3 Hypothèse de stabilité forte

La figure 1.3 représente l'architecture du GCS proposée par [Chockler et al., 2001] avec la présence du composant Liveness et du composant Safety. Le composant Safety est composé des composants Liveness et Network and Failure detector. Le composant Network and Failure detector capture les événements générés par l'interaction entre l'environnement et le détecteur de défaillances.

Le réseau est modélisé par un ensemble de liens logiques unidirectionnels qui lient chaque paire de processus dans le système. Un lien logique entre deux processus représente une collection de chemins physiques entre ces processus. De plus, les liens peuvent être actifs (en anglais, *up*) ou inactifs (en anglais, *down*). Un lien inactif ne transporte aucun message (c'est-à-dire, tout message transporté est perdu). Si un lien est actif et le reste pour toujours à partir de l'instant t alors tout message acheminé par ce lien après t est ultimement reçu par son destinataire.

L'ensemble des événements $\mathbb{E}$ défini dans la section 1.2.3 est étendu et inclut les événements suivants :

- $\{\texttt{channel\_down}(p, q) | p, q \in \mathbb{P}\}$;
- $\{\texttt{channel\_up}(p, q) | p, q \in \mathbb{P}\}$;
- $\{\texttt{net\_send}(p, m) | p \in \mathbb{P}, m \in \mathbb{M}\}$;
- $\{\texttt{net\_receive}(p, m) | p \in \mathbb{P}, m \in \mathbb{M}\}$;
- $\{\texttt{net\_reachable\_set}(p, S) | p \in \mathbb{P}, S \in 2^{\mathbb{P}}\}$;

Ces événements sont représentés dans la figure 1.3. Les événements `channel_up` et `channel_down` reflètent les états des liens de communication, respectivement actif et inactif (cf. définition formelle ci-dessous). Les événements `net_send` et `net_receive` correspondent respectivement à l'envoi et à la réception des messages par le GCS via des liens logiques du réseau sous-jacent. Le GCS reçoit également des informations du détecteur de défaillances qui correspondent aux événements `net_reachable_set`. Plus précisément, l'événement $\texttt{net\_reachable\_set}(p, S)$ dénote l'ensemble S des processus (et seulement ces processus) que le détecteur de défaillances de p croît être l'ensemble des processus corrects et mutuellement

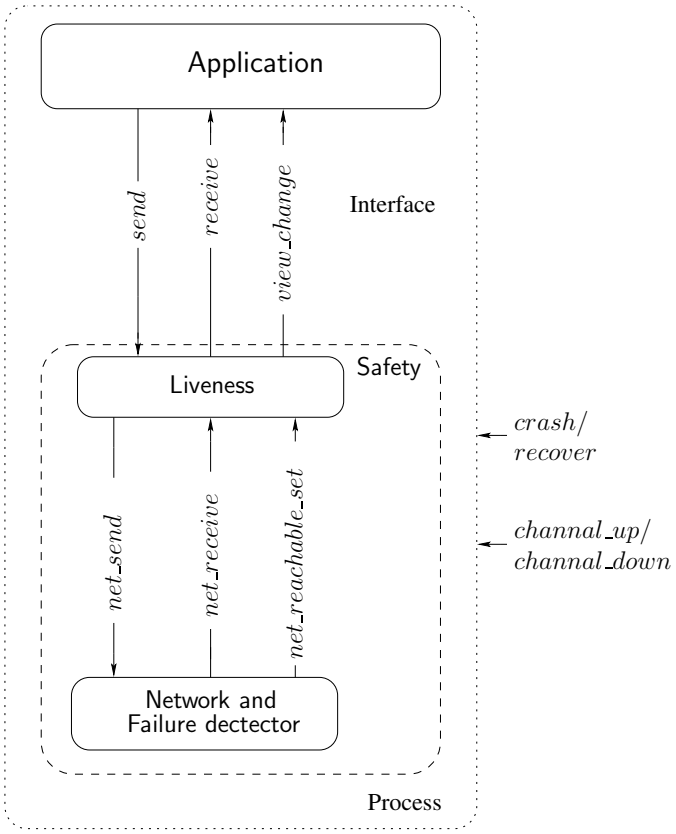

FIGURE 1.3 – Architecture logicielle du GCS.

connectés de sa partition.

Les prédicats suivants sont utilisés par la suite pour présenter les propriétés de vivacité du GCS :

- le processus p est toujours correct :
  $alive(p) \stackrel{\text{déf}}{=} \nexists j : e^j = \texttt{crash}(p)$
- le processus p est correct après le i^e événement :
  $alive\_after(p,i) \stackrel{\text{déf}}{=} \nexists j : e^j = \texttt{crash}(p) \vee \exists j \leqslant i \; \nexists k > j : e^j = \texttt{recover}(p) \; \wedge \; e^k = \texttt{crash}(p)$
- le processus p est défaillant après le i^e événement :
  $crashed\_after(p,i) \stackrel{\text{déf}}{=} \exists j \leqslant i \; \nexists k > j : e^j = \texttt{crash}(p) \; \wedge \; e^k = \texttt{recover}(p)$
- le lien de p à q est actif après le i^e événement :
  $up\_after(p,q,i) \stackrel{\text{déf}}{=} \exists j \leqslant i \; \nexists k > j : e^j = \texttt{channel\_up}(p,q) \wedge e^k = \texttt{channel\_down}(p,q)$
- le lien de p à q est inactif après le i^e événement :
  $down\_after(p,q,i) \stackrel{\text{déf}}{=} \exists j \leqslant i \; \nexists k > j : e^j = \texttt{channel\_down}(p,q) \wedge e^k = \texttt{channel\_up}(p,q)$
- le processus p installe la vue v au i^e événement et n'installe aucune vue après v :
  $last\_view(p,v) \stackrel{\text{déf}}{=} \exists i \; \nexists j > i \; \exists v \; \exists v' \; \exists TS \; \exists TS' : e^i = \texttt{view\_change}(p,v,TS) \wedge e^j = \texttt{view\_change}(p,v',TS')$

Les propriétés de vivacité d'un GCS dépendent de la stabilité du réseau. Cette hypothèse de stabilité du réseau est définie comme suit.

Hypothèse 1. Vivacité du réseau. *S'il existe un instant après lequel deux processus p et q sont corrects, et le lien de p à q est actif, alors tout message envoyé par p à q est reçu par q ultimement. Formellement,*
$\Big(\exists t' \; \forall t'' \geqslant t' : H(p,t') = alive\_after(p,i) \wedge H(q,t') = alive\_after(q,j) \wedge up\_after(p,q,i) \wedge H(p,t'') = \texttt{net\_send}(p,m)\Big) \Rightarrow \exists t > t'' : H(q,t) = \texttt{net\_receive}(q,m)$.

La propriété de vivacité du GCS est conditionnelle. Elle requiert le comportement idéal du GCS seulement dans les exécutions où il existe une partition complètement stable et lorsque le détecteur de défaillances se comporte de manière similaire à $\diamondsuit\mathcal{P}$. Ce détecteur est dénoté par $\diamondsuit\mathcal{P}$-*like* dans [Chockler et al., 2001]. Voici les définitions d'une partition ultimement complètement stable et du détecteur $\diamondsuit\mathcal{P}$-*like*.

Définition 2. Partition ultimement complètement stable. *Une partition ultimement complètement stable est un ensemble de processus S qui sont ultimement corrects et mutuellement connectés, et pour lesquels tous les liens entre les processus dans cet ensemble sont actifs, et les liens entre les processus de cet ensemble et les autres processus se trouvant à l'extérieur de l'ensemble sont inactifs. Formellement,*
$stable\_partition(S) \stackrel{\text{déf}}{=} \exists i \; \forall p \in S \; \forall q \in S \; \forall r \in \mathbb{P} \setminus S : \Big(alive\_after(p,i) \wedge up\_after(p,q,i) \wedge down\_after(r,p,i) \vee crashed\_after(r,i)\Big)$.

Ainsi, la stabilité de la partition contenant les processus mutuellement connectés est requise pour toujours. Cependant, en pratique, cette stabilité n'est nécessaire que pour une période de temps suffisamment longue afin qu'une exécution de l'algorithme de gestion de groupe partitionnable puisse être terminée et que le module de détection de défaillances se stabilise, comme indiqué dans [Chockler et al., 2001] et expliqué dans [Guerraoui and Schiper, 1997, Aguilera, 2010].

Définition 3. *$\Diamond\mathcal{P}$-like. Pour chaque partition stable S, et pour tout processus $p \in S$, l'ensemble des processus atteignables par p fourni par le détecteur est ultimement S. Formellement,*
$\Diamond\mathcal{P}\text{-like} \stackrel{\text{déf}}{=} \forall S \forall p \in S \nexists S' \neq S : \Big(stable\_partition(S) \Rightarrow \exists i : e^i = \texttt{net\_reachable\_set}(p, S) \wedge \nexists j > i : e^j = \texttt{net\_reachable\_set}(p, S')\Big)$.

La propriété de vivacité du service de gestion de groupe partitionnable garantit que chaque processus installe une vue finale dite « précise » qui correspond aux membres de la partition stable. La notion de vue précise est définie comme suit.

Définition 4. Vue précise. *Si le détecteur de défaillances se comporte comme $\Diamond\mathcal{P}$-like, alors pour chaque partition stable S, il existe une vue v avec $v.members = S$ tel que pour tout processus $p \in S$, p installe v en tant que sa dernière vue : Formellement,*
$\Big(\Diamond\mathcal{P}\text{-like} \wedge stable\_partition(S)\Big) \Rightarrow \Big(\exists v : v.members = S \wedge \forall p \in S : last\_view(p, v)\Big)$.

1.4.4 Hypothèse de stabilité faible

Les propriétés de vivacité présentées dans la section 1.4.3 assurent que si une partition est ultimement complètement stable, la gestion de groupe partitionnable fournit une vue précise à tous les membres de cette partition. Cependant, une partition stable, telle que définie dans [Chockler et al., 2001], peut ne jamais exister. Un tel scénario dans le réseau est possible et peut empêcher la progression des processus dans une partition, et dans le pire des scénarios, bloquer le système. La spécification de [Babaoğlu et al., 2001] fournit des propriétés de vivacité de la gestion de groupe partitionnable qui ne sont pas seulement satisfaites dans les partitions complètement stables.

[Babaoğlu et al., 2001] s'est inspiré des spécifications du système Transis. En outre, les spécification de [Babaoğlu et al., 2001] correspondent à un sur-ensemble des spécifications de [Chockler et al., 2001]. Avant de présenter les propriétés de vivacité de [Babaoğlu et al., 2001], nous présentons la notion d'atteignabilité entre processus introduite par les auteurs. La propriété d'atteignabilité exprime la capacité de communication entre deux processus. Elle est définie comme suit.

Définition 5. Atteignabilité. *Si le processus p envoie un message m au processus correct q à l'instant t alors q reçoit m si et seulement si q est atteignable par p à l'instant t. Formellement,*
$reachable(p, q, t) \stackrel{\text{déf}}{=} \Big(\exists t' \exists t'' \geqslant t' : H(p, t') = alive\_after(p, i) \wedge H(q, t') = alive\_after(q, j) \wedge H(p, t'') = \texttt{send}(p, m)\Big) \Rightarrow \exists t > t'' : H(q, t) = \texttt{receive}(q, m)$.

Dans [Babaoğlu et al., 2001], contrairement à [Chockler et al., 2001], les processus défaillants ne se recouvrent pas. Il est important de noter que, contrairement à la notion de processus corrects, la notion d'atteignabilité n'est pas un invariant dans le temps. Un processus q peut être atteignable par p à l'instant t', mais peut ne pas l'être à l'instant $t > t'$.

Une version étendue du détecteur $\diamondsuit\mathcal{P}$ est considérée par les auteurs afin de garantir les propriétés de vivacité dans toutes les partitions et non seulement uniquement dans les partitions complètement stables. Dénotons ce détecteur par $\diamondsuit\tilde{\mathcal{P}}$. Chaque processus possède son propre module $\diamondsuit\tilde{\mathcal{P}}$. À chaque requête, $\diamondsuit\tilde{\mathcal{P}}$ retourne à p l'ensemble des processus atteignables par p que nous notons par $\tilde{S}_p$. Nous notons $\tilde{S}_p(t)$ la valeur retournée à p par $\diamondsuit\tilde{\mathcal{P}}$ à l'instant t.

Définition 6. *$\diamondsuit\tilde{\mathcal{P}}$ est défini par une propriété de complétude forte et une propriété de précision forte ultimes comme suit :*

- complétude forte ultime : *pour chaque processus p, si un processus q devient pour toujours ultimement inatteignable par p alors $q \notin \diamondsuit\tilde{S}_p$ est ultimement vrai pour toujours. Formellement,* $\Big(\exists t_1 \forall t > t_1 : \neg(reachable(p,q,t))\Big) \Rightarrow \exists t_2 \geqslant t_1 \forall t > t_2 : q \notin \diamondsuit\tilde{S}_p(t)$.
- précision forte ultime : *pour chaque processus p, si un processus q devient pour toujours ultimement atteignable par p alors $q \in \diamondsuit\tilde{S}_p$ est ultimement vrai pour toujours. Formellement,* $\Big(\exists t_1 \forall t > t_1 : reachable(p,q,t)\Big) \Rightarrow \exists t_2 \geqslant t_1 \forall t > t_2 : q \in \diamondsuit\tilde{S}_p(t)$.

La propriété de vivacité de la gestion de groupe partitionnable ne peut être satisfaite que sous certaines conditions de stabilité dans lesquelles les relations d'atteignabilité persistent. Si ces conditions, qui sont capturées par $\diamondsuit\tilde{\mathcal{P}}$, sont ultimement vraies alors le service de gestion de groupe partitionnable fournit ultimement une vue précise et complète de la partition pour toujours [8]. Les propriétés de précision et de complétude de vue sont spécifiées comme suit.

Définition 7. Précision de vue. *S'il existe un instant après lequel le processus q reste atteignable par le processus correct p, alors la vue courante v de p inclut ultimement q pour toujours. Formellement,*

$$\exists t_1 : \Big(\forall t \geqslant t_1 : alive(p) \wedge alive(q) \wedge recheable(p,q,t)\Big) \Rightarrow \Big(\exists t_2 \geqslant t_1 \forall t \geqslant t_2 : last\_view(p,v) \wedge q \in v.members\Big).$$

Définition 8. Complétude de vue. *S'il existe un temps après lequel tous les processus dans un ensemble S restent inatteignables par les autres processus, alors ultimement la vue courante v de chaque processus correct n'appartenant pas à S n'inclut aucun processus dans S. Formellement,*

$$\exists t_1 : \Big(\forall t \geqslant t_1 \forall q \in S\, \forall p \notin S : \neg(recheable(p,q,t)\Big) \Rightarrow \Big(\exists t_2 \geqslant t_1 \forall t \geqslant t_2 : alive(p) \wedge last\_view(p,v) \ \wedge \ v.members \cap S = \emptyset\Big).$$

La propriété de vivacité suivante exclut une implantation triviale dans laquelle aucun message n'est livré. Elle n'est pas limitée aux partitions complètement stables.

8. En pratique, comme indiqué pour la définition 2, les conditions de stabilité ne sont nécessaires que pour une période de temps suffisamment longue.

Propriété 5. Propriétés de vivacité des diffusions de messages.

- 1) *un processus correct livre toujours son propre message diffusé. Formellement,* $\exists t' : \Big(alive(p) \wedge H(p,t') = \texttt{send}(p,m)\Big) \Rightarrow \Big(\exists t > t' : H(p,t) = \texttt{receive}(p,m)\Big)$.
- 2) *soit p un processus correct qui livre le message m dans la vue v contenant entre autres le processus q. Si q ne livre jamais m, alors p installe ultimement la nouvelle vue v en tant que successeur immédiat de v'. Formellement,* $\Big(send\_in(p,m,v') \wedge q \in v'.memberes\Big) \Rightarrow \Big(\texttt{receive}(q,m) \vee \exists v : install\_in(p,v,v')\Big)$.

1.4.5 Problèmes des spécifications

Les spécifications proposées par [Chockler et al., 2001] et [Babaoğlu et al., 2001] permettent d'assurer les propriétés de sûreté de base. Pour assurer les propriétés de vivacité, [Chockler et al., 2001] et [Babaoğlu et al., 2001] étendent la définition du détecteur de défaillances non fiable $\Diamond\mathcal{P}$ en deux versions différentes. Ces versions étendues de $\Diamond\mathcal{P}$ détectent ultimement tous les processus mutuellement atteignables dans une partition. Ils sont différents dans le sens où, dans [Chockler et al., 2001], la vivacité est garantie seulement dans les partitions complètement stables, tandis que dans [Babaoğlu et al., 2001], la vivacité est assurée dans toutes les partitions, qu'elles soient complètement stables ou non.

Prenons un point de vue partique par rapport à ces spécifications. L'utilisation seule du service de gestion de groupe partitionnable dans la réalisation des applications est limitée [Babaoğlu et al., 1998]. En général, de telles applications requièrent une coopération entre les processus du même groupe qui est facilitée par l'utilisation d'un service de diffusion de messages dans le groupe. Or, la primitive de diffusion fiable de base dans le contexte d'une vue est la synchronie virtuelle qui est spécifiée dans [Chockler et al., 2001] et [Babaoğlu et al., 2001]. Par conséquent, avant de présenter les problèmes des spécifications de [Chockler et al., 2001] et [Babaoğlu et al., 2001] (respectivements dans les sections 1.4.5.2 et 1.4.5.3), nous détaillons d'abord la propriété de synchronie virtuelle dans la section 1.4.5.1.

1.4.5.1 Propriété de synchronie virtuelle

La propriété de synchronie virtuelle stipule que deux processus qui installent deux mêmes vues consécutives v' et v livrent le même ensemble de messages dans la vue v' [Chockler et al., 2001, Babaoğlu et al., 2001, Pleisch et al., 2008]. Elle a été introduite d'abord dans le système de gestion de groupe de partition primaire Isis [Birman and Joseph, 1987]. De nombreux travaux ont étendu cette propriété afin de l'adapter au service de gestion groupe partitionnable [Chockler et al., 2001]. Elle est définie comme suit.

Propriété 6. Synchronie virtuelle. *Si deux processus p et q installent la même vue v dans la vue précédente v' alors chaque message m reçu par p dans la vue v' est aussi reçu par q dans la vue v'. Formellement,*
$install\_in(p,v,v') \wedge install\_in(q,v,v') \wedge receive\_in(p,m,v') \Rightarrow receive\_in(q,m,v')$.

Observons que la propriété de synchronie virtuelle n'impose pas d'ordre entre les réceptions de messages. Elle concerne les réceptions de messages par rapport aux changements de vue. La propriété de synchronie virtuelle peut être exploitée par des applications déterministes par morceaux qui répliquent des données en utilisant l'approche machines à états. Les applications changent leur état suite aux traitements des messages reçus. Afin de préserver les répliques dans un état cohérent, les messages sont diffusés selon un ordre total. Lorsque le réseau se partitionne, les répliques des processus déconnectés divergent. Ce scénario est intrinsèque aux systèmes partitionnables. Quand les processus précédemment déconnectés se reconnectent, ils exécutent une opération de transfert d'état qui leur permet d'atteindre un état commun cohérent. Cependant, le transfert d'état n'est pas toujours nécessaire pour chaque nouvelle installation de vue. Comme le soulignent [Amir et al., 1997, Chockler et al., 2001], quand un processus p installe une vue v, p doit déterminer d'abord le sous-ensemble des processus TS dans $v.membres$ qui sont aussi dans la vue v' qui précède directement v. Par exemple, selon la propriété de synchronie virtuelle, tous les processus p dans TS ont reçu le même ensemble de messages dans v' et se trouvent donc dans le même état après avoir installé v. Dès lors, le transfert d'état n'est pas nécessaire entre ces processus.

Il est important de noter que l'ensemble de processus TS ne correspond pas nécessairement à l'intersection des ensembles des processus des deux vues. Il peut exister des vues dites « cachées », comme le montre la figure 1.4. Les processus p et q sont initialement dans la même partition et installent la même vue $v' = (1, \{p, q\})$. Le réseau se partitionne. q détecte le partitionnement du réseau et installe la nouvelle vue $v'' = (2, \{q\})$, ce qui n'est pas le cas pour p. Quand les liens de communications sont rétablis, les deux processus installent la même vue $v = (3, \{p, q\})$. Sans algorithme de fusion de groupe particulier, p ne sait pas que q a installé une vue intermédiaire. Autrement dit, la vue v'' est « cachée » pour p qui n'en a pas connaissance. Le problème provient alors du fait que les processus p et q n'installent pas deux mêmes vues consécutives. En effet, même si la propriété de synchronie virtuelle est vérifiée, l'installation de la vue v par p et q ne permet pas d'imposer la réception par p et q des mêmes messages dans v'.

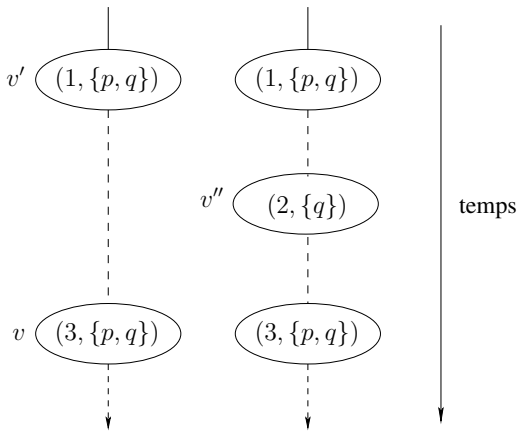

FIGURE 1.4 – Ensemble transitionnel et vue cachée.

Ainsi, la synchronie virtuelle est une propriété observée par un observateur externe. Si le composant local du service de gestion de groupe partitionnable de p ne fournit pas d'information à propos des vues installées par les autres processus, alors p ne peut pas déduire TS à partir seulement de deux vues consécutives v et v' : p ne peut pas savoir si la propriété de synchronie virtuelle est satisfaite ou non. Il existe deux solutions pour remédier à ce problème. Ces solutions sont basées sur l'une des deux propriétés suivantes : *1)* ensemble transitionnel et *2)* accord sur les successeurs.

L'ensemble transitionnel permet aux processus de déterminer localement si une opération de transfert d'état est nécessaire pour satisfaire la propriété de synchronie virtuelle.

Propriété 7. Ensemble transitionnel.

1) *Si le processus p installe la vue v dans la vue v' alors l'ensemble des processus transitionnels pour la vue v de p est un sous-ensemble TS de l'ensemble des processus correspondant à l'intersection des membres dans v' et v. Formellement,*
$\Big(\texttt{view\_change}(p, v, TS) = e \wedge viewof(e) = v'\Big) \Rightarrow TS \subseteq v.members \ \cap \ v'.members.$

2) *Si deux processus p et q installent la même vue v alors q est inclus dans l'ensemble transitionnel TS de p dans la vue v si et seulement si la vue précédente de p est identique à la vue précédente de q. Formellement,*
$\Big(\texttt{view\_change}(p, v, TS) = e \wedge viewof(e) = v' \wedge install\_in(q, v, v'')\Big) \Rightarrow \Big(q \in TS \Leftrightarrow v' = v''\Big).$

Dans le scénario présenté dans la figure 1.4, l'ensemble des processus transitionnels de p est $\{p\}$ après avoir installé la vue $(3, \{p, q\})$. Dans ce cas, p peut déduire que q a installé une vue cachée $(2, \{q\})$. Donc, une opération de transfert d'état est nécessaire pour satisfaire la propriété de synchronie virtuelle.

Notons que s'il existe un processus $r \in v'.members \cap v.members$ qui n'installe pas v' alors la propriété 7 ne spécifie pas si r est inclus dans les ensembles transitionnels des autres processus.

La propriété d'ensemble transitionnel utilisée en conjonction avec la propriété de synchronie virtuelle permet d'assurer que tous les processus dans l'ensemble transitionnel de p possèdent un état identique à celui de p. Ainsi, les applications peuvent exploiter cette information afin de déterminer si le transfert d'état est nécessaire entre deux installations de vues.

Une alternative à la propriété d'ensemble transitionnel pour l'utilisation conjointe avec la propriété de synchronie virtuelle est la propriété d'accord sur les successeurs. La propriété d'accord sur les successeurs assure que chaque processus dans l'intersection de la vue courante v de p avec la vue précédente v' de p est dans la vue précédente v' de p.

Propriété 8. Accord sur les successeurs. *Si le processus p installe la vue v dans la vue v' et si un processus q, membre de v', installe aussi v alors q installe v dans v'. Formellement,*
$\Big(install\_in(p, v, v') \wedge install(q, v) \wedge q \in v'.members\Big) \Rightarrow install\_in(q, v, v').$

En utilisant les propriétés présentées dans cette section, nous présentons respectivement dans les deux sections suivantes le problème des vues capricieuses dans la spécification de [Chockler et al., 2001] et le problème des liens équitables dans la spécification de [Babaoğlu et al., 2001].

1.4.5.2 Problème des vues capricieuses dans la spécification de [Chockler et al., 2001]

La gestion de groupe partitionnable gère l'installation des vues dans chaque groupe de processus. Chaque installation de vue doit être justifiée et refléter le résultat des événements qui se sont produits dans l'environnement [Anceaume et al., 1995]. En particulier, les installations des vues capricieuses, parasites ou arbitraires (en anglais, *spurious views*) ne doivent pas être autorisées. L'installation de vues capricieuses correspond au fait qu'un ensemble de processus arbitraire S est autorisé à installer une nouvelle vue v, avec $v.members = S$, à n'importe quel instant et sans aucune raison. Cela autorise alors la suppression arbitraire de processus corrects et mutuellement connectés. Dans ce cas, le service de gestion de groupe partitionnable n'aide pas les processus à obtenir des informations correspondant à la réalité de l'environnement. La figure 1.5 montre un scénario d'installation de vues capricieuses. Le processus p installe les vues capricieuses v_1, v_4 et v_7 en plus des vues « réalistes » v_3, v_6 et v_9. Aucune des vues v_1, v_4 et v_7 ne contient q. Dans le pire des scénarios représentés dans cette figure, p installe une vue arbitraire juste avant chaque installation d'une vue réaliste.

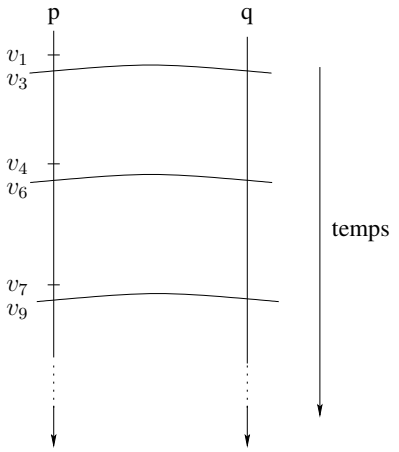

FIGURE 1.5 – Vues capricieuses.

Nous considérons donc que, pour être assez forte, la spécification de la gestion de groupe partitionnable doit éviter l'installation de vues capricieuses. Par conséquent, la gestion de groupe partitionnable doit répondre aux deux exigences qui suivent. Premièrement, si certains événements tels que des suspicions de défaillances (vraies ou fausses) se produisent, alors une nouvelle vue du groupe est ultimement installée afin de refléter ces événements. Deuxièmement, une nouvelle vue est installée seulement si certains de ces événements se sont produits. Nous reprenons maintenant l'argumentation de [Pleisch et al., 2008] montrant que la spécification de [Chockler et al., 2001] n'est pas assez forte.

[Pleisch et al., 2008] fournit deux algorithmes $\mathcal{A}_{ptGMS}$ et $\mathcal{A}_{vscast}$ qui satisfont la spécification de [Chockler et al., 2001]. $\mathcal{A}_{ptGMS}$ implante le service de gestion de groupe partitionnable. $\mathcal{A}_{vscast}$ est un algorithme pour la diffusion de messages qui satisfait la propriété de synchronie

virtuelle.

L'idée de l'algorithme $\mathcal{A}_{ptGMS}$, exécuté par le processus p, est la suivante :

- à chaque fois que la valeur de sortie du détecteur de défaillances change, p installe une nouvelle vue ;
- une vue singleton ne contenant que p est installée entre deux vues non singletons ;
- deux processus qui possèdent la même valeur de sortie du détecteur de défaillances installent une vue commune (la vue ayant le plus grand identifiant). Afin d'assurer l'accord sur la vue installée, chaque processus envoie périodiquement la vue qu'il perçoit aux autres processus dans cette vue.

L'idée de l'algorithme $\mathcal{A}_{vscast}$, exécuté par le processus p, est la suivante :

- l'algorithme $\mathcal{A}_{vscast}$ peut accéder aux variables de l'algorithme $\mathcal{A}_{ptGMS}$;
- quand un message m est diffusé par p, m est directement livré à p. p envoie le message m avec l'identifiant de sa vue courante aux autres processus dans cette vue. Cette opération d'envoi est répétée tant que la valeur de sortie du détecteur de défaillances ne change pas et que p n'a pas reçu les acquittements de tous les processus membres de la vue ;
- à la réception d'un message m associé à une vue v, p vérifie si l'identifiant de v est le même que celui de sa vue courante et si m n'a pas déjà été reçu. Si c'est le cas, p envoie un acquittement à l'expéditeur du message m.

[Pleisch et al., 2008] montre que ces deux algorithmes assurent les propriétés d'auto-inclusion (propriété 2), de monotonicité locale (propriété 3), d'événement de vue initiale (propriété 4) et de synchronie virtuelle (propriété 6). Remarquons que l'installation de vues capricieuses singletons est autorisée. En particulier, dans l'histoire locale du processus p, une vue non singleton est suivie d'une vue singleton ne contenant que p. Par conséquent, cela remet en cause la pertinence de la spécification de la gestion de groupe partitionnable présentée par [Chockler et al., 2001].

1.4.5.3 Problème de définition des liens équitables dans la spécification de [Babaoğlu et al., 2001]

Dans [Babaoğlu et al., 2001], une version étendue du détecteur $\diamondsuit\mathcal{P}$ précédemment notée $\diamondsuit\tilde{\mathcal{P}}$ est considérée afin de garantir les propriétés de vivacité dans toutes les partitions et non pas seulement dans les partitions complètement stables. [Pleisch et al., 2008] montre qu'avec le modèle de système de [Babaoğlu et al., 2001] complété de $\diamondsuit\tilde{\mathcal{P}}$, il n'existe pas d'algorithme qui assure en même temps les propriétés de précision de vue (propriété 7), de synchronie virtuelle[9] (propriété 6) et de vivacité (propriété 5). L'origine de cette impossibilité vient du fait qu'il existe des exécutions dans lesquelles l'oracle $\diamondsuit\tilde{\mathcal{P}}$ ne permet pas de distinguer correctement les processus qui sont ultimement atteignables pour toujours des processus qui alternent en permanence entre être atteignables et être inatteignables.

Plus précisément, selon la définition de l'atteignabilité, il existe des exécutions dans lesquelles le détecteur $\diamondsuit\tilde{\mathcal{P}}$ ne permet pas de distinguer des processus ultimement atteignables pour toujours de ceux qui alternent en permanence entre être atteignables et être inatteignables. Par exemple,

9. Dans [Babaoğlu et al., 2001], la propriété de synchronie virtuelle est appelée l'accord de messages (en anglais, *message agreement*).

[Pleisch et al., 2008] considère deux exécutions R et R' telles que dessinées dans la figure 1.6. Dans l'exécution R, le processus $q \neq p$ est atteignable par p pour toujours et v est la vue dans R telle que toutes les vues successeurs de v installées par p contiennent au moins les processus p et q. L'exécution R' est identique à R jusqu'à l'installation de la vue v par p, la vue v' successeur de v ne contient pas q, et la vue v'' successeur de v' contient à nouveau q. Cela est possible car q alterne en permanence entre être atteignable et être inatteignable par p dans R' et v' est précisément installée durant une période pendant laquelle q est inatteignable.

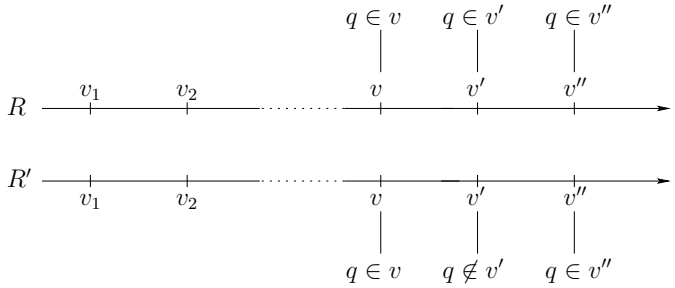

FIGURE 1.6 – Atteignable *versus* inatteignable.

[Babaoğlu et al., 2001] cherche à éviter ce scénario défavorable et ainsi à permettre des partitions non complètement stables qui ne sont pas autorisées par [Chockler et al., 2001]. Pour ce faire, les auteurs enrichissent le système avec la notion de liens équitables (en anglais, *fair channel*). Autrement dit, l'objectif est d'éviter les scénarios d'instabilité dans lesquels deux processus qui sont mutuellement atteignables par intermittence deviennent mutuellement inatteignables précisément aux moments où ils tentent de construire une vue. La notion de lien équitable est définie comme suit.

Définition 9. Lien équitable. *Soient p et q deux processus qui sont mutuellement atteignables entre eux. Le lien de p à q est dit équitable si lorsque p envoie un nombre non borné de messages à q et que q est correct alors q reçoit un nombre non borné de ces messages. Formellement,*
$\Big(\forall t \exists t_1 \geqslant t : reachable(p, q, t_1) \wedge \forall t \exists t_2 \geqslant t : H(p, t_2) = \texttt{net\_send}(p, m)\Big)$
$\wedge \Big(\forall t \exists t_3 \geqslant t : alive(q) \wedge H(q, t_3) = \texttt{net\_receive}(q, m') \wedge \texttt{net\_send}(p, m') \in H(p, [t, +\infty[)\Big)$.

Les deux premiers termes de la conjonction expriment le fait que q est infiniment souvent atteignable par p et que p envoie le message m à q infiniment souvent. Les troisième et quatrième termes stipulent que q reçoit un nombre infini de messages envoyés par p.

Cependant, la spécification de la gestion de groupe partitionnable est dorénavant basée sur une propriété dépendante du temps (définition 5 de la relation d'atteignabilité) dans un modèle de système réparti comprenant une propriété indépendante du temps (définition 9 des liens équitables). [Pleisch et al., 2008] démontre que le scénario de la figure 1.6 peut encore survenir et ainsi que la précision de vue n'est pas garantie.

Pour remédier au problème des définitions tantôt dépendantes du temps, tantôt indépendantes du temps, [Pleisch et al., 2008] suggère que la notion d'atteignabilité soit définie comme

proposé dans [Aguilera et al., 2000] : pour deux processus corrects p et q, q est atteignable par p si et seulement si le lien de p à q est équitable. Pour cela, un lien équitable devrait être défini comme suit.

Définition 10. Lien équitable (version révisée). *Soient p et q deux processus. Le lien de p à q est dit équitable si lorsque p envoie à q un nombre non borné de messages et que q est correct alors q reçoit un nombre non borné de ces messages. Formellement,*
$(\forall t \exists t_1 \geqslant t : H(p, t_1) = \texttt{net\_send}(p, m))$
$\wedge\ (\forall t \exists t_2 \geqslant t : alive(q) \wedge H(q, t_2) = \texttt{net\_receive}(q, m') \wedge \texttt{net\_send}(p, m') \in H(p, [t, +\infty[))$.

Ensuite, la propriété d'atteignabilité entre deux processus p et q pourrait être définie en utilisant la notion de lien équitable comme suit.

Définition 11. Atteignabilité (version révisée). *Soient deux processus p et q, q est atteignable par p si et seulement si le lien de p à q est équitable. Formellement,*
$atteignable(p, q) \stackrel{\text{déf}}{=} lien\_equitable(p, q)$.

En considérant ces définitions des liens équitables et d'atteignabilité indépendantes du temps, remarquons que la nouvelle définition des liens équitables n'est pas suffisante pour modéliser les systèmes très dynamiques tels que les MANETs. En effet, dans les MANETs, les nœuds peuvent rejoindre ou quitter le système aussi rapidement qu'aléatoirement. Ainsi, la topologie du réseau peut changer à des instants imprévisibles d'une manière rapide et aléatoire, et provoquer les deux conséquences suivantes : *1)* les liens de communication entre les nœuds d'une même partition n'existent pas dès l'initialisation mais sont créés dynamiquement et *2)* les liens qui sont créés peuvent ne pas durer assez longtemps pour tolérer les pertes de messages infinies autorisées par la propriété d'équitabilité. Alors, les partitions peuvent ne jamais devenir stables, c'est-à-dire qu'il est possible que les groupes de nœuds soient incapables de progresser dans leur exécution. En outre, dans les MANETs, les nœuds n'ont pas une connaissance globale du système et le nombre de processus participants à l'exécution n'est *a priori* pas connu à l'avance.

Par conséquent, la définition d'un modèle de système dynamique partitionnable avec recouvrement qui est implantable et assez fort pour garantir la vivacité non uniquement dans les partitions complètement stables, reste un défi. La notion de lien équitable doit être enrichie de contraintes temporelles afin de fournir des liens qui capturent la dynamicité des MANETs : les liens doivent permettre de capturer le fait que les nœuds d'une partition restent suffisamment longtemps ensemble. Ces nœuds sont alors dits stables pendant la période, les autres sont dits instables. En outre, comme le suggère [Mostefaoui et al., 2005] [10], la condition de stabilité dans les systèmes dynamiques caractérisés par la succession de périodes stables et instables doit être définie de la façon suivante : les exécutions utiles (telles que le consensus ou la gestion de groupe) doivent impliquer seulement un ensemble α de processus stables. Dans le chapitre 2, nous suivons cet argument et pensons comme [Mostefaoui et al., 2005] que cette condition de stabilité faible permet de mieux capturer la dynamicité des systèmes.

10. Cependant, [Mostefaoui et al., 2005] définit un modèle pour les systèmes de partition primaire dans lesquels il n'est pas possible d'avoir plusieurs partitions stables qui évoluent concurremment.

1.5 Conclusion

Dans ce chapitre, nous avons présenté l'état de l'art de la gestion de groupe partitionnable. La spécification d'un service de gestion de groupe partitionnable doit répondre aux exigences suivantes : elle doit être *1)* assez forte pour fournir des garanties utiles aux applications réparties et *2)* assez faible pour être résoluble (implantable).

En deux décennies, plusieurs spécifications de la gestion de groupe partitionnable ont été proposées dans la littérature. Toutefois, aucune d'entre elles ne satisfait les deux exigences ci-dessus. Nous avons présenté plus particulièrement les spécifications de [Chockler et al., 2001] et de [Babaoğlu et al., 2001] qui sont considérées comme les deux références. Néanmoins, la spécification de [Chockler et al., 2001] ne satisfait pas la première exigence tandis que celle de [Babaoğlu et al., 2001] ne fournit pas la seconde. En outre, [Chockler et al., 2001] et [Babaoğlu et al., 2001] considèrent un modèle de système dans lequel les nœuds ne sont pas mobiles et le réseau est supposé fortement connexe dès l'initialisation. Un tel modèle n'est pas adapté aux MANETs. En effet, dans ces derniers, le nombre et l'identité des processus ne sont *a priori* pas connus à l'avance. En outre, les liens de communication n'existent pas dès l'initialisation, mais sont créés dynamiquement.

Chapitre 2

Modèle de système dynamique pour les réseaux mobiles spontanés

Sommaire

2.1 Introduction

Dans le chapitre précédent, nous avons étudié l'état de l'art de la gestion de groupe partitionnable, et plus particulièrement les deux spécifications de référence [Babaoğlu et al., 2001] et [Chockler et al., 2001]. Les modèles de systèmes considérés dans ces références ciblent plus les systèmes répartis basés sur des réseaux fixes que ceux basés sur des réseaux mobiles spontanés. Par exemple, le graphe du réseau est supposé fortement connexe à l'initialisation. La dynamicité de la topologie des MANETs et le fait que les participants apparaissent et disparaissent ne permettent pas d'utiliser ces modèles.

L'objectif de ce chapitre est de modéliser les systèmes dynamiques tels que les MANETs. Le modèle doit permettre une spécification de groupe partitionnable assez faible pour être implan-

table et assez forte pour garantir la propriété de vivacité dans les partitions non complètement stables. Pour ce faire, nous adaptons les concepts et définitions présentés dans le chapitre précédent. Nous introduisons de nouveaux concepts et propriétés qui capturent la dynamicité des MANETs.

Le chapitre est organisé comme suit. Dans la section 2.2, nous présentons le modèle de système réparti dynamique avec partitionnement. Dans la section 2.3, nous positionnons notre modèle par rapport aux autres modèles proposés dans la littérature avant de conclure dans la section 2.4.

2.2 Modèle de système réparti dynamique avec partitionnement

Dans cette section, nous étendons le modèle de système réparti présenté dans la section 1.2 afin de l'adapter aux réseaux mobiles spontanés. Les concepts et définitions présentés dans la section 1.2 sont admis :

- le modèle considéré est dynamique/avec recouvrement/avec partitionnement ;
- $\mathbb{P}$ est l'ensemble des processus, $\mathbb{M}$ est l'ensemble des messages, $\mathbb{V}$ est l'ensemble des vues et $\mathbb{E}$ est l'ensemble des événements. $\mathbb{E}$ est étendu pour inclure les événements suivants : $\{\texttt{add}(p,q,v) | p \in \mathbb{P}, q \in \mathbb{P}, v \in \mathbb{P}\}$ et $\{\texttt{remove}(p,q,v) | p \in \mathbb{P}, q \in \mathbb{P}, v \in \mathbb{P}\}$. Donc, $\mathbb{E}$ comprend les événements `send`, `receive`, `view_change`, `crash`, `recover`, `net_send`, `net_receive`, `add`, et `remove` ;
- nous reprenons les prédicats suivants : *receive*, *send*, *install*, *install_in*, *crash_in*, *alive*, $alive\_after$, $crashed\_after$, *last_view* ;
- les processus et les messages sont identifiés de manière unique. En outre, nous considérons que le contenu d'un message peut être modifié, mais pas son identifiant, et qu'un message est seulement identifié par son identifiant. Par exemple, deux messages m' et m sont identiques si et seulement s'ils possèdent le même identifiant (bien que leur contenu soit identique ou non). Pour garantir l'unicité des identifiants des messages, nous pouvons par exemple considérer que l'identifiant d'un message est une paire $(sender, counter)$, où *sender* est l'identifiant du processus expéditeur du message et *counter* est un compteur local à l'expéditeur ;
- le système est asynchrone : la durée de transmission des messages est inconnue et non bornée. En revanche, pour simplifier le modèle, la durée d'exécution d'un pas est bornée, et pour simplifier la présentation, elle est estimée nulle ;
- la notions de pile logicielle à base de composants (ou modules) est utilisée pour spécifier les interfaces et les propriétés ainsi que les algorithmes ;
- les événements sont ordonnés partiellement dans l'histoire globale du système.

Cette section est organisée comme suit. Nous définissons les processus qui communiquent entre eux par diffusion de messages sur des liens de communication sans fil dans les sections 2.2.1 et 2.2.2. Dans la section 2.2.3, nous présentons différents types de liens de communication et définissons leurs propriétés. Dans la section 2.2.4, nous illustrons ces premiers concepts et ces premières définitions. Dans la section 2.2.5, nous définissons le concept de partition avant d'introduire le concept de détecteur de participants d'une partition. Par la suite, dans les sections 2.2.6

et 2.2.7, nous introduisons respectivement la condition de stabilité et le critère de stabilité qui permettent d'assurer la progression et la terminaison des exécutions. Enfin, dans la section 2.2.8, nous complétons l'exemple de la section 2.2.4 pour illustrer notre définition d'une partition stable.

2.2.1 Modèle d'arrivée infinie avec accès simultané borné

Un système réparti conçu au dessus des MANETs est composé d'un ensemble infini de nœuds mobiles identifiés de façon unique. Pour simplifier la présentation, nous considérons qu'il existe un processus par nœud mobile. Les termes « nœud (mobile) » et « processus » sont donc interchangeables. Ainsi, le système est composé d'un ensemble infini de processus $\mathbb{P} = \{\ldots p_i, p_j, p_k \ldots\}$. Nous dénotons les processus aussi par p, q, r, etc.

Nous considérons le modèle d'arrivée infinie avec accès simultané borné (en anglais, *infinite arrival model with bounded concurrency*) [Merritt and Taubenfeld, 2000] : sur une période de temps finie, seul un nombre fini de nœuds exécutent l'algorithme réparti. Le nombre total de nœuds dans une exécution peut cependant croître à l'infini au fil du temps. Autrement dit, chaque exécution possède un niveau d'accès simultané fini mais inconnu. Contrairement aux systèmes statiques, dans les systèmes dynamiques, les processus ne connaissent *a priori* pas l'ensemble $\mathbb{P}$, c'est-à-dire que les processus dans $\mathbb{P}$ ne se connaissent pas nécessairement les uns les autres avant de se rencontrer. Un processus correct ne défaille pas. Un processus défaille par arrêt franc. Un processus défaillant peut se recouvrir et récupérer l'état stocké dans sa mémoire stable locale.

2.2.2 Graphe du réseau

Dans les réseaux MANETs, pour communiquer entre eux, les nœuds mobiles n'utilisent pas de primitive de communication de type point-à-point, mais diffusent des messages qui sont reçus par les nœuds voisins directs qui se trouvent dans leur portée de transmission. L'événement $\texttt{broadcast}_{nbg}(p, m)$ correspond à la diffusion du message m par p à l'ensemble de ses voisins directs dans le réseau. Nous complétons donc $\mathbb{E}$ avec l'événement $\texttt{broadcast}_{nbg}(p, m)$ qui correspond à la diffusion du message m par le processus p. Nous ajoutons aussi le prédicat suivant :

$$broadcast_{nbg}(p, m) \quad \stackrel{\text{déf}}{=} \quad \exists t : H(p, t) = \texttt{broadcast}_{nbg}(p, m)$$

Un lien de communication du processus p au processus q est créé si q se trouve dans la portée de transmission de p. Nous dénotons ce lien par $p \rightsquigarrow q$. Les liens de communication sont unidirectionnels. Rappelons que les processus et les messages sont identifiés de façon unique et que les délais de transmission des messages ne sont pas bornés. La communication entre deux nœuds non voisins est possible via des chemins qui sont modélisés par des séquences de liens sans fil établis entre les différents nœuds du réseau.

Nous modélisons le système réparti pour les MANETs par un graphe orienté $G = (\mathbb{P}, \mathbb{A})$ où l'ensemble des processus dans le système $\mathbb{P}$ correspond à l'ensemble des sommets dans le graphe et $\mathbb{A}$ modélise l'ensemble des liens unidirectionnels qui existent entre les sommets ($\mathbb{A} \subset \mathbb{P} \times \mathbb{P}$). Pour deux sommets p_1 et p_2, si $(p_1, p_2) \in \mathbb{A}$, cela signifie que $p \rightsquigarrow q$ existe pendant une période de temps $[t_1, t_2]$ avec $t_2 > t_1 \geqslant t \in \mathbb{N}^*$. La figure 2.1 présente une partie du graphe G pendant

une période de temps donnée durant laquelle la topologie du graphe du réseau ne change pas. Les cercles pointillés représentent les portées de transmission des nœuds. Les flèches représentent des liens de communication unidirectionnels entre ces nœuds. Par exemple, il existe un lien de communication du nœud x au nœud y et un autre de y vers x car x et y se trouvent dans la portée de transmission l'un l'autre. Cela n'est pas le cas pour les processus u et q. Il existe un lien de u vers q. Mais, il n'existe pas de lien de q vers u.

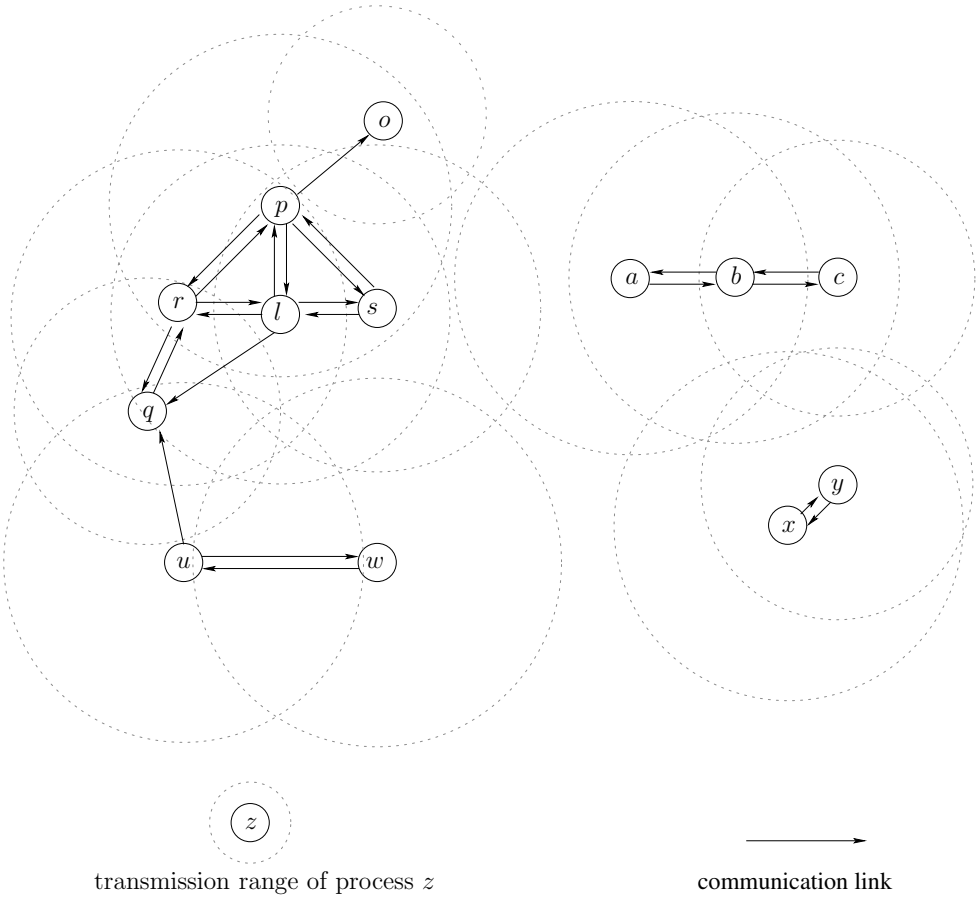

FIGURE 2.1 – Graphe du réseau et liens de communication.

Par la suite, nous utilisons la notion de lien et de chemin dynamique pour caractériser la dynamicité de la topologie du graphe du réseau.

2.2.3 Propriétés des liens de communication

Nous distinguons trois types de liens de communication : *1)* ultimement actif, *2)* ultimement inactif et *3)* actif par intermittence (pour toujours). Un lien *ultimement actif* transporte ultime-

ment les messages sans perdre aucun d'entre eux. Un lien *ultimement inactif* arrête ultimement de transporter des messages. Enfin, les messages transportés par un lien *actif par intermittence (pour toujours)* peuvent être perdus. Dans notre étude, les liens actifs par intermittence (pour toujours) sont la source de l'instabilité du système. Cela se traduit par le fait que deux processus sont mutuellement atteignables seulement par intermittence. Dans le pire des cas, aucune exécution utile ne peut être terminée si deux processus p et q ne sont pas mutuellement atteignables aux moments où ils tentent d'échanger des messages. Pour éviter ces scénarios dégradés, nous supposons maintenant que les liens actifs par intermittence, incluant les liens ultimement actifs, sont équitables. Un lien équitable est défini comme suit.

Définition 12. Lien équitable. *Soient p et q deux processus. Le lien $p \rightsquigarrow q$ est dit équitable si, lorsque p diffuse à q un nombre non borné de messages et que q est correct, alors q reçoit un nombre non borné de ces messages. Formellement,*
$\Big(\forall t\ \exists t_1 \geqslant t : H(p,t_1) = \texttt{broadcast}_{nbg}(p,m)\Big) \wedge \Big(\forall t\ \exists t_2 \geqslant t \forall i : alive\_after(q,i) \wedge H(q,t_2) = \texttt{net\_receive}(q,m') \wedge\ \texttt{broadcast}_{nbg}(p,m') \in H(p,[t,+\infty[)\Big)$.

Nous notons une séquence de processus $(p_1 p_2 \dots p_n)$, dans laquelle les liens $p_1 \rightsquigarrow p_2$, $\dots$, $p_{n-1} \rightsquigarrow p_n$ sont équitables, comme un chemin équitable de p_1 à p_n, dénoté par $fair\_path(p_1 p_2 \dots p_n)$. Par définition, un chemin équitable peut perdre des messages.

Définition 13. Atteignabilité. *Soient deux processus p et q, q est atteignable par p si et seulement si le chemin de p à q est équitable. Nous dénotons cette relation d'atteignabilité de p à q par $\rightharpoonup (p,q)$. Ainsi, $\rightharpoonup (p,q) \overset{\text{déf}}{=} fair\_path(p \dots q)$.*

Si un processus q est atteignable par p, et vice-versa, nous écrivons $\leftrightharpoons (p,q)$. Dans ce cas, p et q sont dits mutuellement atteignables. Notons que, comme suggéré dans la section 1.4.5.3, la définition de chemin équitable ainsi que la définition d'atteignabilité sont indépendantes du temps comme suggéré et proposé par [Pleisch et al., 2008, Aguilera et al., 2000].

Avec uniquement des liens et des chemins équitables, l'application peut souffrir des pertes de messages et des délais de transmission de messages non bornés. Dans le pire des scénarios, il n'existe pas de « période stable » pendant laquelle les processus communiquent entre eux avec des garanties temporelles [11] afin de progresser et terminer une exécution utile. Aussi, nous introduisons le concept de lien SADDM (en anglais, *Simple Average Delayed/Dropped of a Message* [12]) stipulant que le délai de communication entre deux processus utilisant un tel lien est borné durant les périodes dites stables. La définition d'un lien SADDM est inspirée par la combinaison d'un lien équitable avec un lien à garantie temporelle (en anglais, *eventually timely*) [Aguilera et al., 2003]. Un lien à garantie temporelle impose qu'ultimement tous

11. La définition de garanties temporelles a été introduite dans [Dwork et al., 1988] pour modéliser les systèmes partiellement synchrones : « *il existe une borne supérieure de transfert d'un message, mais cette borne n'est pas connue par les processus* ». Comme montré dans [Schiper, 2004], cette définition est équivalente à celle qui suit : « *le délai de transfert d'un message est borné et la borne est connue. Cependant, la borne n'est pas appliquée dès le début de l'exécution de l'algorithme, mais seulement à partir d'un instant de stabilisation global GST non connu par les processus* ».

12. Ce concept est inspiré des travaux de [Sastry and Pike, 2007] via la notion de lien ADD.

les messages transportés sont reçus par les destinataires dans un délai borné. Autrement dit, ultimement, un lien à garantie temporelle ne perd aucun des messages qu'il transporte. Un lien SADDM est plus faible qu'un lien à garantie temporelle puisqu'il autorise que des messages soient perdus et que d'autres soient transmis avec des délais de transmission non bornés. Un lien SADDM est plus fort qu'un lien équitable puisqu'il assure qu'un sous-ensemble des messages transportés sont reçus ultimement par le destinataire et que les messages de cet ensemble ne sont pas trop dispersés dans le temps. Nous estimons que ce type de lien correspond mieux aux communications sans fil d'un système MANETs réel. Un lien SADDM est défini comme suit.

Définition 14. Lien SADDM. *Soient β et δ deux constantes, et soit $I = [t_1, t_2]$ un intervalle de temps fini durant lequel le processus p diffuse un message m au processus correct q au moins β fois. Le lien $p \rightsquigarrow q$ est un lien SADDM si q reçoit le message m au moins une fois à l'instant $t_1 + \delta$, avec $\delta > t_2 - t_1$. Formellement,*

$$saddm\_link(p,q) \stackrel{\text{déf}}{=} \exists I = [t_1, t_2] : \Big(\forall t_{i\in[1,\beta]} \in I : \; H(p, t_i) = \texttt{broadcast}_{nbg}(p, m) \\ \wedge \; \forall t \geqslant t_1 \forall i : H(q, t) = alive\_after(q, i) \\ \wedge \; \exists t' \leqslant t_1 + \delta : H(q, t') = \texttt{receive}(q, m) \wedge \delta > (t_2 - t_1)\Big)$$

Nous définissons un chemin SADDM comme une séquence de liens SADDM.

Définition 15. Chemin SADDM. *Une séquence de processus $(p_1 \dots p_n)$ est un chemin SADDM dans un intervalle de temps $\phi = [\tau_1, \tau_2]$ si $\forall i, j \in [1, n-1]$: $i \neq j \implies p_i \neq p_j$ et le lien $p_i \rightsquigarrow p_{i+1}$ est un lien SADDM dans ϕ.*

Cette définition exprime le fait qu'il existe une constante $c \in \mathbb{N}^*$ telle que, pour chaque intervalle de temps fini $\phi = [\tau_1, \tau_2]$ durant lequel p_1 diffuse de saut en saut un message m vers le processus p_n au moins β^c fois, p_n reçoit au moins un de ces messages via le chemin $saddm\_path(p_1 p_2 \dots p_n)$ avant l'instant $\tau_1 + \beta^c + c\delta$, avec $\beta^c + c\delta > \tau_2 - \tau_1$. Étant donné que l'accès simultané au système est borné, $\beta^c + c\delta$ est borné.

Un chemin SADDM du processus p vers le processus q est dénoté par $saddm\_path(p \dots q)_\phi$ relativement à l'intervalle de temps ϕ. L'intervalle ϕ sur le chemin est omis quand le contexte est clair et qu'il n'y a pas d'ambiguïté.

2.2.4 Exemple illustratif de chemins stables

La figure 2.1 est complétée afin d'illustrer les notions de liens et de chemins SADDM. Dans la figure 2.2, les cercles pointillés représentent les portées de transmission des nœuds. Les flèches solides correspondent à des liens SADDM. Les liens non SADDM sont représentés par des flèches pointillées. Les processus u et w peuvent communiquer l'un avec l'autre de manière régulière et avec garantie temporelle puisqu'il existe des liens SADDM de u vers w et de w vers u. C'est le cas aussi pour les processus x et y. En outre, les processus p, q, r, s et l peuvent communiquer entre eux de manière régulière et avec garantie temporelle car il existe des chemins SADDM entre chaque paire de processus. C'est aussi le cas pour les processus dans $\{a, b, c\}$. Les processus $\{p, q, r, s, l\}$ ne peuvent pas communiquer de manière régulière et avec garantie temporelle avec

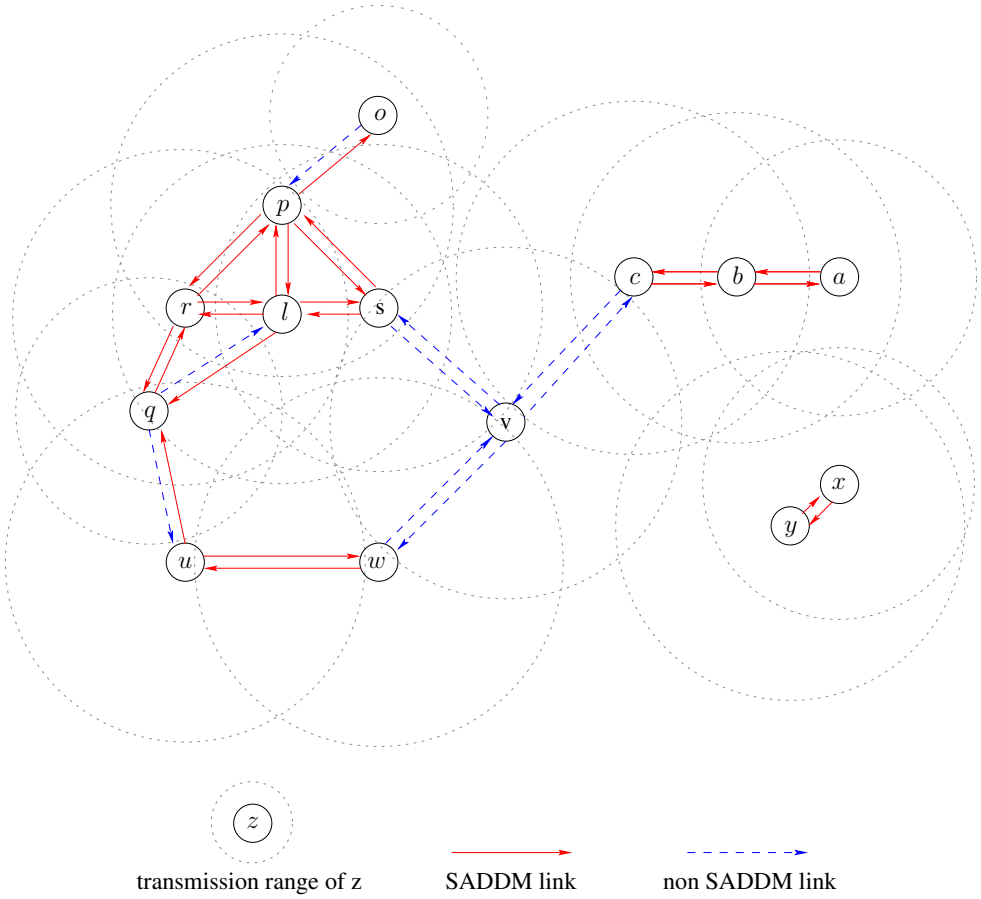

FIGURE 2.2 – Liens et chemins SADDM.

u puisqu'il n'existe pas de chemin SADDM d'un processus dans $\{p, q, r, s, l\}$ vers u. C'est aussi le cas entre les processus o et v.

2.2.5 Partition et concept de détecteur de participants d'une partition

Le graphe du réseau est partitionnable : plusieurs sous-ensembles de processus peuvent co-exister tels que les processus dans chaque sous-ensemble sont mutuellement atteignables. Les processus qui sont dans deux sous-ensembles différents ne sont pas mutuellement atteignables. La relation $\leftrightharpoons$ est une relation d'équivalence et les *partitions* sont définies par des classes d'équivalence de cette relation. La partition du processus p est dénotée par $PART_p$ et est définie comme suit.

Définition 16. Partition de p. *Soit un processus correct p. La partition de p est l'ensemble des processus qui sont mutuellement atteignables par p. Formellement,*

$$PART_p \stackrel{\text{déf}}{=} \{q \in \mathbb{P} | \leftrightharpoons (p, q)\}.$$

Afin de s'assurer qu'une exécution utile puisse progresser et terminer, une partition doit satisfaire une certaine forme de stabilité pendant une période de temps suffisamment longue. Les processus dans les partitions stables sont dits stables. Un processus est stable dans le contexte d'une partition. Un motif de partition est une fonction $\mathbf{P} : \mathbb{P} \times \mathcal{T} \rightarrow 2^{\mathbb{P}}$, où $\mathbf{P}(p, t)$ dénote l'ensemble des processus que p croit être dans sa partition. Un processus peut rejoindre ou quitter une partition de façon arbitraire. Ainsi, la fonction $\mathbf{P}$ n'est pas nécessairement monotone dans le temps. Par analogie avec les détecteurs de défaillances non fiables [Chandra and Toueg, 1996], les détecteurs de participants d'une partition sont des oracles répartis associés aux processus, un pour chaque processus. Les détecteurs de défaillances proposées dans [Chandra and Toueg, 1996] sont conçus pour les systèmes de partition primaire dans lesquels chaque paire de processus est connectée par un lien de communication non fiable. Le détecteur de défaillances est utilisé pour identifier les processus qui sont corrects. À la différence des détecteurs de défaillances, dans les systèmes partitionnables, et comme les processus peuvent rejoindre et quitter une partition, la spécification d'un détecteur de participants d'une partition doit être basée sur la capacité des processus à communiquer entre eux dans une partition plutôt que sur celle de détecter individuellement les processus qui sont corrects ou défaillants.

2.2.6 Hypothèse de stabilité

Dans un système de partition primaire composé de $|\Pi|$ processus, s'il existe une majorité de processus corrects qui peuvent communiquer entre eux durant une période de temps suffisamment longue, alors le système est considéré comme stable pendant cette période. Par analogie, dans un système partitionnable, une partition devient stable durant une période de temps quand les processus corrects dans cette partition peuvent communiquer entre eux durant cette période. Jusqu'à présent, nous n'avons pas défini ce qu'est une telle période stable. Pour ce faire, comme dans [Mostefaoui et al., 2005], nous définissons l'intervalle de temps $\Delta = [t_b, t_e]$. t_b et t_e sont définis par les processus participant à l'exécution de l'application répartie. t_b est l'instant de début de l'exécution de l'application et t_e son instant final. En pratique, l'exécution de l'application peut être divisée en phases. Les phases sont alors exécutées dans des périodes stables. Dans chaque période stable, les processus peuvent communiquer via des chemins SADDM. La partition stable associée à la période Δ est dénotée $\Delta PART_p$ et est définie comme suit.

Définition 17. Partition stable par période. *La partition stable de la période Δ du processus p, notée $\Delta PART_p$, est l'ensemble de tous les processus corrects tel qu'il existe au moins un chemin SADDM de p vers chaque processus $q \in \Delta PART_p$ et un chemin SADDM de q vers p. Formellement,*

$$\Delta PART_p \stackrel{\text{déf}}{=} \{q \in \Pi | saddm\_path(p \ldots q)_\Delta \wedge saddm\_path(q \ldots p)_\Delta\}.$$

Pour simplifier la présentation et sans perdre en généralité, nous considérons seulement une période dans le reste de ce manuscrit. Durant une période, la propriété de stabilité est une propriété qui reste satisfaite une fois qu'elle est satisfaite, c'est-à-dire qu'elle reste satisfaite

après un instant de stabilisation local à la partition notée lst, qui n'est pas connu des processus. Une partition stable associée au processus p est dénotée $\diamondsuit PART_p$ et est définie comme suit.

Définition 18. Partition stable. *La partition stable du processus p, notée $\diamondsuit PART_p$, est l'ensemble des processus corrects tel qu'il existe un instant lst_p après lequel il existe au moins un chemin SADDM de p vers chaque processus $q \in \diamondsuit PART_p$ et un chemin SADDM de q vers p. Formellement,*

$$\diamondsuit PART_p \stackrel{\text{déf}}{=} \Big\{ q \in \Pi |\, \exists lst_p\, \forall t' \geqslant lst_p \exists t'' > t' : \; saddm\_path(p \ldots q)_{[t',t'']} \wedge saddm\_path(q \ldots p)_{[t',t'']} \Big\}$$

Un processus q est stable dans une partition s'il existe des chemins SADDM de ce processus vers tous les autres processus de la partition et des chemins SADDM depuis chacun des processus de la partition vers q, et ce pendant une période suffisamment longue, afin de permettre aux processus de la partition de terminer l'exécution de l'algorithme (ou plus généralement d'exécuter une phase de l'algorithme). Nous définissons le concept de processus stable dans le contexte d'une partition stable comme suit.

Définition 19. Processus stable. *Soient p et q deux processus corrects. S'il existe un instant t après lequel $q \in \diamondsuit PART_p$, alors q est stable dans $\diamondsuit PART_p$. Nous dénotons le processus stable par $stable(p, q)$. Formellement,*

$$stable(p, q) \stackrel{\text{déf}}{=} q \in \diamondsuit PART_p.$$

Par la suite, par abus de langage, le processus q dans la définition précédente est dit stable sans mentionner le nom de la partition associée et nous écrirons $stable(q)$. En outre, par définition, un processus stable est correct. Dans la suite, nous considérons qu'un processus est stable sans préciser qu'il est correct.

Une partition peut ne jamais devenir complètement stable, c'est-à-dire $\diamondsuit PART_p$ peut ne jamais exister. Un tel scénario peut empêcher les processus de progresser dans une partition. Dans le pire des cas, cela peut bloquer l'application. Pour remédier à ce problème, la propriété de stabilité doit être affaiblie afin de permettre la progression et la terminaison des exécutions utiles même dans ces scénarios dégradés. Par ailleurs, puisque les processus stables et instables peuvent coexister dans le contexte d'une partition, il est souhaitable d'éviter que les processus instables empêchent la progression des processus stables. Par conséquent, les algorithmes utiles devraient être exécutés seulement par un ensemble constitué de α processus stables dans la partition. Intuitivement, α exprime le compromis entre l'accord et la progression parmi les processus d'une partition. Dans notre étude, α est un paramètre du module du détecteur de participants de partition. Il est de la responsabilité de l'application de fournir une valeur appropriée de α qui désigne le nombre de participants minimum. Ainsi, nous définissons la condition de stabilité associée au processus p comme suit.

Définition 20. Condition de stabilité. $|\diamondsuit PART_p| \geqslant \alpha_p$.

Les processus stables sont mutuellement atteignables pour toujours après l'instant de stabilisation minimal lst_p. Les processus ne connaissent pas lst_p. Étant donné que les patrons de

mouvement, les bandes passantes... des nœuds mobiles sont *a priori* différents, seul un sous-ensemble des participants sont sélectionnés parmi les nœuds qui sont mutuellement atteignables pour former une partition la plus stable possible et qui satisfait la condition de stabilité (au moins α processus stables). La sélection des processus stables peut être basée sur un ou plusieurs critères de stabilité. Nous définissons dans la section suivante un critère de stabilité pour sélectionner les nœuds propageant le mieux les messages de type battement de cœur.

2.2.7 Critère de stabilité

Un critère de stabilité est un paramètre qui peut être utilisé afin de déterminer quels sont les nœuds mutuellement atteignables les plus stables, c'est-à-dire les nœuds qui peuvent être considérés comme faisant partie d'une éventuelle partition stable. Les concepteurs d'applications peuvent choisir différents critères de stabilité pour leurs applications. Le choix des paramètres appropriés peut être influencé par les besoins applicatifs.

Dans notre étude, nous choisissons le critère de stabilité $HB(p,q) \geqslant \text{THRESHOLD}_p$, avec $HB : \mathbb{P}\times\mathbb{P} \to \mathbb{N}$ une fonction qui dépend du nombre de battements de cœur reçus par p de la part de q et $\text{THRESHOLD}_p \geqslant 1$ une valeur seuil. $HB(p,q)$ augmente si q est présent dans la partition de p et diminue sinon. q est considéré ou marqué comme stable par p si $HB(p,q) \geqslant \text{THRESHOLD}_p$, et retiré de la partition de p si $HB(p,q) = 0$, c'est-à-dire que p ne reçoit plus de battement de cœur de q. En utilisant ce critère de stabilité, nous pouvons éliminer un nœud lorsqu'il quitte cette partition tout en tolérant les déconnexions sporadiques.

2.2.8 Exemple illustratif de partitions stables

Nous complétons la figure 2.2 pour illustrer les concepts de processus stable, de condition de stabilité et de partition stable. Dans la figure 2.3, les disques noirs représentent les processus instables tandis que les disques blancs les processus stables. Chaque partition stable est entourée par un cercle en trait plein. Sont dessinées cinq partitions stables $\diamondsuit PART_o, \diamondsuit PART_p, \diamondsuit PART_w, \diamondsuit PART_a$ et $\diamondsuit PART_x$, avec leur valeur de α qui est égale respectivement à 1, 4, 2, 3 et 2. α de $\diamondsuit PART_o$ est égal à 1 par exemple pour exprimer le fait que l'application s'exécutant sur o accepte de progresser/fonctionner seule/sans autre participant. α de $\diamondsuit PART_p$ exprime quant à lui que les participants de cette partition requièrent au moins 4 membres pour progresser/fonctionner.

Les processus peuvent se déplacer dans la partition stable. Il est important de noter que les nœuds d'une partition stable ne sont pas nécessairement isolés des autres nœuds du réseau. En fonction de la connectivité du réseau, il est possible qu'un ou plusieurs nœuds d'une partition stable puissent diffuser des messages à d'autres nœuds, ou recevoir des messages de processus qui n'appartiennent pas à leur partition. Autrement dit, les liens d'un processus d'une partition stable vers un autre processus qui n'appartient pas à cette partition ne sont pas nécessairement inactifs. Par exemple, le processus u dans la partition stable $\diamondsuit PART_w$ peut recevoir des messages diffusés par les processus dans $\diamondsuit PART_p$ via des chemins SADDM. Mais, les processus dans $\diamondsuit PART_p$ ne peuvent pas recevoir avec garantie temporelle des messages diffusés par u car il

n'existe pas de chemin SADDM d'un processus dans $\diamondsuit PART_w$ vers p. u est considéré comme instable dans le contexte de $\diamondsuit PART_p$ et stable dans le contexte de $\diamondsuit PART_w$.

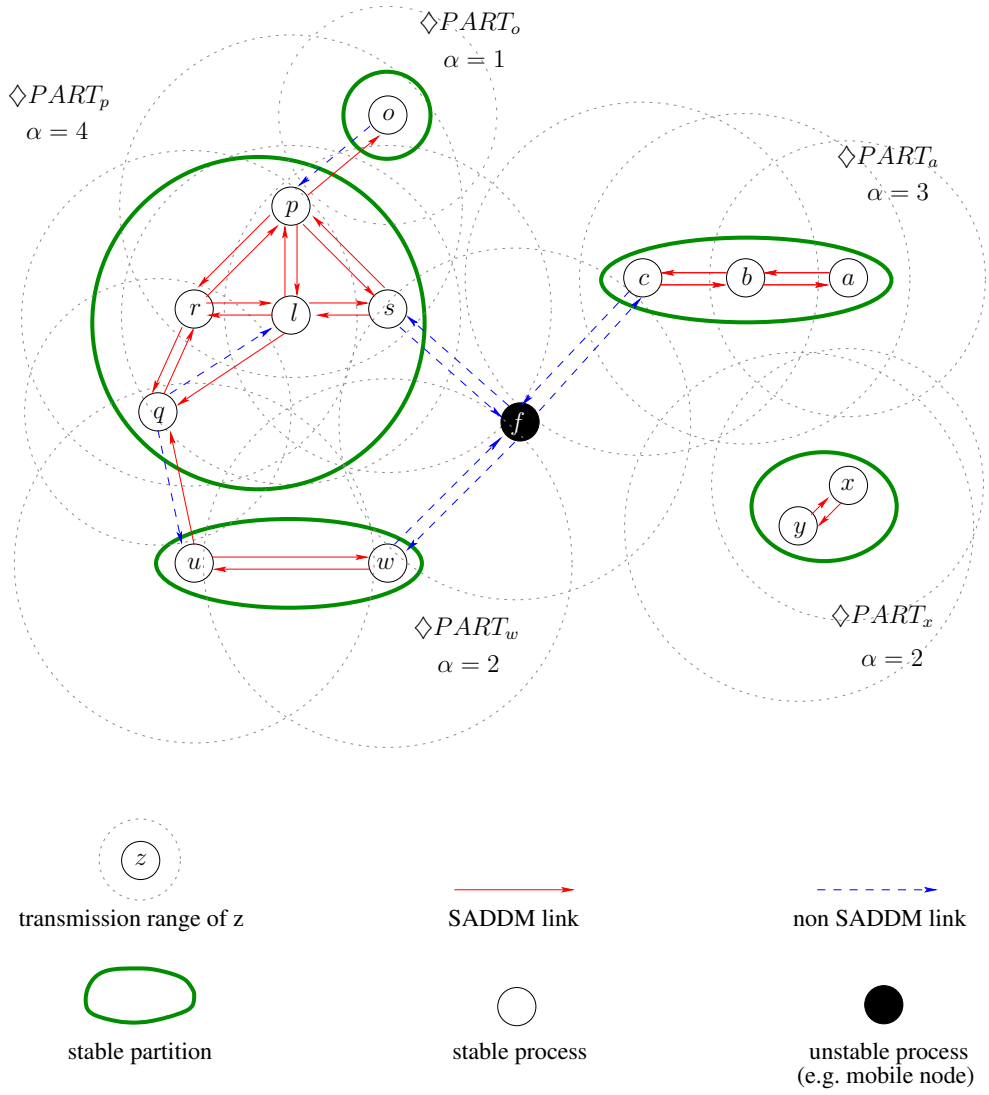

FIGURE 2.3 – Partitions stables et condition de stabilité.

Dans la section suivante, nous présentons les modèles de systèmes dynamiques avec les condition de stabilité considérés dans les travaux connexes.

2.3 Travaux connexes

[Hermant and Le Lann, 2002, Mostefaoui et al., 2005, Greve et al., 2011] présentent des modèles de systèmes dynamiques accompagnés d'une condition de stabilité. Dans [Hermant and Le Lann, 2002], α dénote le nombre minimum de processus qui exécutent l'algorithme et ne sont jamais suspectés d'être défaillants. Chaque processus défaillant est ultimement suspecté par tous les processus corrects. Le modèle est conçu pour les systèmes de partition primaire. Dans [Mostefaoui et al., 2005], la valeur de α joue le rôle de $(n - f)$ dans les systèmes statiques, où n est le nombre total de processus dans le système et f est le nombre maximal de processus qui peuvent être défaillants. Comme dans notre modèle, [Mostefaoui et al., 2005] considère qu'un système réparti dynamique doit posséder une certaine période de stabilité afin de garantir la progression et la terminaison des exécutions. Cependant, [Mostefaoui et al., 2005] cible les systèmes de partition primaire dans lesquels il existe un seul groupe dit « cœur » formé par les α processus stables. Ainsi, il n'est pas possible d'avoir plusieurs partitions stables simultanément comme dans notre approche. En outre, les liens de communication existent avant à la création du système et sont bidirectionnels. [Greve et al., 2011] étend le protocole de communication QUERY-RESPONSE de [Mostefaoui et al., 2005] en considérant en plus la mobilité des nœuds et propose un détecteur de défaillances non fiable $\diamondsuit S^M$ qui détecte ultimement l'ensemble des processus connus et stables. Un processus est connu s'il a rejoint le système et a été identifié par un processus stable. Un processus est stable si après avoir rejoint le système, il ne l'a jamais quitté par la suite. La valeur locale α d'un nœud p correspond à la densité au voisinage de p moins le nombre maximum de processus qui peuvent être défaillants dans le voisinage de p. Bien que le modèle proposé par [Greve et al., 2011] considère la mobilité des nœuds dans les MANETs, il est conçu pour les systèmes de partition primaire.

[Alekeish and Ezhilchelvan, 2011] considère les MANETs peu denses (en anglais, *sparse MANETs*) dans lesquels les déconnexions et les partitionnements du réseau sont des événements fréquents. Parmi l'ensemble des processus S du système, un petit groupe de nœuds G est formé dans le but d'exécuter un consensus uniforme. Tous les nœuds dans S ne participent pas à l'exécution du consensus, mais seulement les nœuds de G. Les nœuds qui sont dans $S \setminus G$ jouent le rôle de routeurs et coopèrent afin de découvrir et de maintenir la connectivité entre les nœuds de G. Parmi les n processus dans G, il est supposé qu'au plus $f < n/2$ d'entre eux peuvent défaillir durant le cycle de vie de G. Un processus défaillant ne se recouvre pas. En outre, chaque nœud connaît les nombres f et n ainsi que l'identité de tous les processus dans le système, mais ne connaît pas l'identité des processus de G qui défaillent et les instants auxquels les défaillances se produisent. Le consensus peut être résolu dans G si une majorité des nœuds corrects de G restent connectés pendant une période de temps suffisamment longue. Comme dans notre modèle, les communications via des chemins sans fil entre deux nœuds pendant une période stable sont assurées avec garanties temporelles. Cependant, aucune perte de message n'est autorisée pendant les périodes stables. Enfin, le modèle est proposé pour les systèmes de partition primaire.

[Tucci-Piergiovanni and Baldoni, 2010] a pour objectif de caractériser les système répartis dynamiques et considère le modèle d'arrivée infinie avec accès simultané borné par la constante b (en anglais, *infinite arrival model with b-bounded concurrency*) [Aguilera, 2004]. Comme dans

notre modèle, les processus peuvent rejoindre ou quitter le système et le nombre total de processus dans une exécution peut croître à l'infini au fil du temps. Cependant, à la différence de notre modèle, le nombre d'accès simultanés ne dépasse pas une borne supérieure connue b. [Tucci-Piergiovanni and Baldoni, 2010] propose un oracle appelé HB^* qui implante le détecteur de défaillances non fiable Ω, qui identifie ultimement le leader unique du système. HB^* fournit une liste des processus stables [13] contenant des processus stables et dont la taille est b. HB^* satisfait la propriété de stabilité suivante : les processus stables possèdent ultimement des positions fixes dans la liste des processus stables. Même si le problème de partitionnement du réseau est considéré durant des périodes instables, le modèle cible le problème de couverture de connectivité ultime (en anglais, *eventual connectivity overlay*), c'est-à-dire qu'ultimement il n'y a pas de partitionnement du réseau. Le modèle capture le comportement des processus qui sont déployés dans des réseaux WANs tandis que notre modèle capture les processus qui sont déployés dans des MANETs. De plus, le graphe du réseau dans [Tucci-Piergiovanni and Baldoni, 2010] est considéré comme étant fortement connexe. C'est une propriété essentiellement considérée pour les réseaux qui ne se partitionnent pas de façon permanente. Au contraire, dans notre modèle, le graphe du réseau n'est pas nécessairement complètement connexe et le réseau peut se partitionner de façon permanente. De plus, les chemins entre les nœuds mobiles sont construits dynamiquement au cours du temps.

Le concept de détecteur de défaillances à base de battement de cœur (en anglais, *heartbeat failure detector*) $\mathcal{HB}$ a été généralisé dans [Aguilera et al., 1999] pour les réseaux partitionnables. Le module $\mathcal{HB}$ fournit une liste dont chaque élément est associé à un processus. Le nombre de battements de cœur des processus qui n'appartiennent pas à la même partition est borné. Notre concept de détecteur de participants d'une partition est inspiré de [Aguilera et al., 1999]. Cependant, dans [Aguilera et al., 1999], le système est statique, l'ensemble des nœuds dans le système est connu et fixe, et les nœuds ne se déplacent pas. De plus, les événements `add` et `remove` ne sont pas considérés.

Par analogie avec le concept de détecteur de participants de partition, la notion de détecteur de participants (en anglais, *participant detector*) a été introduite dans [Cavin et al., 2004] pour résoudre le problème d'auto-amorçage dans un MANET. Le détecteur de participants capture l'information minimale dont un processus a besoin afin de résoudre un consensus sans connaître à l'avance les participants. Ce problème est connu sous le nom de CUP pour *Consensus with Unknown Participants*. Comme dans notre approche, le nombre et l'identité des processus sont initialement inconnus. [Greve and Tixeuil, 2007] étend [Cavin et al., 2004] et identifie une hypothèse de synchronie dite minimale pour résoudre le consensus et le consensus uniforme dans des scénarios autorisant les défaillances des processus. Cependant, les modèles dans ces deux travaux ne considèrent pas le cas du partitionnement permanent du réseau. Ces modèles sont conçus pour les systèmes de partition primaire dans lesquels le nombre total de processus dans le système est fini et le graphe du réseau est toujours connexe. Dans notre modèle, le système est dynamique et partitionnable, et le réseau n'est pas toujours connexe.

[Bui-Xuan et al., 2003] utilise la notion de graphe évolutif afin de modéliser la dépendance

13. Appelés *good* dans [Tucci-Piergiovanni and Baldoni, 2010].

temporelle des chemins entre les nœuds dans les systèmes dynamiques tels que les réseaux sans fil. Un graphe évolutif est une séquence indexée de sous-graphes dans laquelle le sous-graphe associé à un indice i donné correspond à la connectivité du réseau à l'intervalle de temps indiqué par l'indice i. Pour ce faire, chaque nœud ou lien possède une « planification de présence » qui indique la période durant laquelle le nœud participe au système. Le modèle est adapté pour les systèmes de réseau dynamiques dans lesquels la trajectoire des nœuds mobiles est supposée déterministe tels que les systèmes de satellites en orbite basse (en anglais, *Low Orbit Satellite*). Plus généralement, le modèle cible les systèmes dynamiques de planification fixe (en anglais, *fixed schedule dynamic networks*). Comme dans notre approche, les graphes évolutifs capturent la notion de chemins au cours du temps. Cependant, comme la trajectoire des nœuds n'est pas déterministe dans les MANETs, les chemins ne peuvent pas être planifiés, mais sont crées dynamiquement. En outre, les graphes évolutifs ne supportent pas un nombre infini de nœuds.

Dans notre travail précédent [Arantes et al., 2010], nous avons proposé un modèle de système dynamique pour les MANETs. Comme dans [Chockler et al., 2001], les propriétés de vivacité du système sont garanties seulement dans les partitions complètement stables. Une implantation d'un détecteur de participants de partition utilisant des chemins dynamiques est également proposé afin de capturer cette propriété de vivacité. Nous avons étendu le concept de chemin dynamique afin de proposer la notion de chemin SADDM qui permet de garantir les propriétés de vivacité dans les partitions non complètement stables. L'idée de la combinaison des chemins équitables avec des chemins à garantie temporelle provient de [Sastry and Pike, 2007] via la notion de lien ADD (en anglais, *Average Delayed/Dropped*).

2.4 Conclusion et perspectives

La tâche de définir un modèle de système dynamique pour les MANETs est complexe à cause de la forte volatilité des nœuds : les nœuds peuvent rejoindre et quitter le système aussi rapidement qu'arbitrairement. Les nœuds qui restent mutuellement atteignables pendant une période de temps suffisamment longue constituent des partitions stables. Cependant, les partitions ne sont pas nécessairement isolées les unes des autres et peuvent ne jamais être complètement stables. Il est possible que des groupes de nœuds ne puissent pas progresser et terminer des exécutions utiles telles que le consensus. Pour cela, nous proposons un modèle de système dynamique avec une condition de stabilité faible qui permet de mieux capturer les comportements très dynamiques des MANETs et rendre notre spécification implantable. La stabilité d'une partition est basée sur la notion de chemin SADDM qui combine des propriétés des liens équitables et des liens avec garantie temporelle. Intuitivement, une partition est stable s'il existe un chemin SADDM dynamiquement créé entre chaque paire de processus dans cette partition. Un chemin SADDM est plus faible qu'un chemin avec garantie temporelle car il autorise les pertes de messages ou les délais de transmission de messages non bornés. D'un autre côté, il est plus fort qu'un chemin équitable car il impose qu'une partie des messages soient transmis avec des garanties temporelles. Cette notion de chemin SADDM permet de modéliser les communications sans fil dynamiques et spontanées dans les MANETs.

Pour compléter notre solution pour la modélisation des MANETs, nous proposons le concept

de détecteur de participants d'une partition. La spécification d'un détecteur de participants d'une partition est basée sur la capacité des processus à communiquer entre eux dans une partition via des chemins SADDM plutôt que de détecter individuellement les processus qui sont corrects ou défaillants. Enfin, le choix des processus dans une partition stable est déterminé par un critère de stabilité construit par comptage des messages battements de cœur.

Ensuite, nous avons défini une condition de stabilité faible pour chaque partition. Elle est basée sur le paramètre α qui représente le nombre de processus stables minimum requis dans une partition afin de permettre la progression et la terminaison des exécutions utiles.

Les perspectives de la modélisation des MANETS sont doubles. Plusieurs travaux proposent différents critères de stabilité. En particulier, [García et al., 2009] propose des critères de stabilité basés sur le temps de présence du processus dans la partition, sur la distance entre nœuds mobiles, ou encore sur le niveau d'énergie des nœuds. [Boulkenafed et al., 2005] propose d'autres critères de stabilité basés sur la proximité géographique, sur le nombre de sauts de communication ainsi que sur la qualité de service (incluant la fiabilité, la sécurité, la performance, la bande passante, la charge de calcul, et la mémoire). La première perspective est donc d'étudier ces critères de stabilité et de sélectionner ceux qui sont adaptés à notre modèle.

Dans une deuxième perspective, il nous semble intéressant d'étudier d'autres propriétés et conditions de stabilité quand la valeur de α devient dynamique.

Dans ce chapitre, nous avons décrit notre modèle de système dynamique partitionnable avec recouvrement. Nous sommes maintenant en mesure de présenter notre approche de résolution de la gestion de groupe partitionnable à base de consensus abandonnables dans le prochain chapitre.

Chapitre 3

Gestion de groupe basée sur Paxos

Sommaire

3.1 Introduction

Un problème d'algorithmique répartie proche de la gestion de groupe qui requiert aussi l'accord entre les processus du système est le consensus. L'objectif du consensus est de permettre à un ensemble de processus, chacun possédant sa propre valeur initiale, de décider ultimement une valeur unique. L'action de décider est irrévocable. De manière plus précise, le problème du consensus est défini par les quatre propriétés suivantes [Chandra and Toueg, 1996] :

- *terminaison* : chaque processus correct décide ultimement une valeur ;
- *intégrité uniforme* : chaque processus décide au plus une fois ;
- *accord uniforme*[14] : deux processus ne décident pas différemment ;

14. Nous considérons le consensus *uniforme*. Il existe un autre type de consensus, le *consensus régulier*, dans

– *validité uniforme* : si un processus décide une valeur *val*, alors *val* a été proposée par un processus.

Les propriétés de validité uniforme, d'accord uniforme et d'intégrité uniforme correspondent aux propriétés de sûreté tandis que la propriété de terminaison est une propriété de vivacité. Cependant, le problème du consensus ne peut pas être résolu dans les systèmes répartis purement asynchrones [Fischer et al., 1985]. Plus précisément, le résultat d'impossibilité de [Fischer et al., 1985] stipule qu'il n'existe pas d'algorithme déterministe pour résoudre le problème du consensus dans un système purement asynchrone sujet à des défaillances par arrêt franc. Dans les systèmes statiques de partition primaire, le problème du consensus est typiquement spécifié pour un ensemble fixe de n processus avec au plus $f < n$ processus pouvant subir des défaillances et chaque paire de processus est reliée par un lien de communication asynchrone. Le consensus peut être résolu en ajoutant des hypothèses de synchronie au système afin d'exclure les scénarios défavorables. [Dolev et al., 1987] caractérise l'effet de l'ajout des hypothèses de synchronie au système. En particulier, les auteurs montrent que la résolution du consensus devient possible quand au plus f processus peuvent défaillir et quand certaines contraintes sont satisfaites afin de permettre aux processus de progresser et aux messages échangés entre processus d'être reçus. [Dwork et al., 1988] introduit le modèle de système partiellement synchrone permettant de résoudre le consensus dans lequel il existe une borne supérieure de transfert d'un message, cette borne n'étant pas connue des processus.

[Chandra and Toueg, 1996] propose une autre approche qui permet d'échapper au résultat d'impossibilité de [Fischer et al., 1985]. Celle-ci repose sur l'utilisation du concept de détecteur de défaillances non fiable. En particulier, il a été montré que le détecteur de défaillances non fiable Ω est le plus faible détecteur qui peut être utilisé pour résoudre le consensus dans un système avec la présence d'une majorité de processus corrects ($n-f > f$) [Chandra et al., 1996a]. [Aguilera et al., 2004] montre que dans un système avec $n - f > f$ processus corrects, il est possible d'implanter Ω et de résoudre le consensus s'il existe un processus correct (non connu à l'avance) qui possède f liens sortants transmettant ultimement les messages de manière fiable et avec une durée de transmission bornée.

La gestion de groupe diffère du consensus sur au moins deux points [Chandra et al., 1996b] : *1)* dans la gestion de groupe, un processus qui est suspecté d'être défaillant peut être retiré du groupe, même si la suspicion est fausse ; *2)* le consensus requiert la progression dans toutes les exécutions tandis que la gestion de groupe autorise les exécutions qui « ne font rien » (par exemple, « ne rien faire » peut correspondre à un scénario dans lequel aucun processus ne souhaite rejoindre ou quitter le groupe, et aucun processus n'est détecté comme défaillant). Résoudre le consensus permet de résoudre la gestion de groupe dans les systèmes de partition primaire. Intuitivement, cela est réalisé en transformant la gestion de groupe de partition primaire en une séquence de consensus, où chaque consensus est exécuté par les processus de la vue courante et la décision retournée par le consensus est l'ensemble des processus de la vue suivante [Guerraoui and Schiper, 2001, Schiper, 2006, Schiper, 2004]. Ainsi, notre intuition de départ est que, puisque le consensus peut être utilisé pour résoudre le problème de la gestion

lequel la propriété d'accord requiert que seuls les processus corrects ne décident pas différemment. L'algorithme proposé par [Chandra and Toueg, 1996] permet de résoudre le problème du consensus uniforme.

de groupe de partition primaire, un autre type de consensus pourrait permettre de trouver une solution pour la gestion de groupe partitionnable.

L'un des algorithmes de consensus les plus connus dans les systèmes de partition primaire est l'algorithme du consensus de Paxos [Lamport, 1998, Lamport, 2001]. Notons que Paxos met justement en place une séquence de consensus. Il existe plusieurs variantes de Paxos, chaque variante étant configurée pour un environnement spécifique [Lamport, 2001, Gafni and Lamport, 2003, Lamport and Massa, 2004, Lamport, 2006]. À notre connaissance, aucune de ces variantes ne considère les spécificités des systèmes dynamiques avec partitionnement. Notre approche pour la résolution de la gestion de groupe partitionnable est d'utiliser une adaptation de Paxos pour les systèmes partitionnables. Afin de réaliser cette adaptation, nous utilisons les méthodes proposées par [Boichat et al., 2003a, Boichat et al., 2003b] qui déconstruisent et reconstruisent Paxos. Nous proposons alors la spécification d'un consensus abandonnable en adaptant Paxos. Le consensus abandonnable que nous mettons en œuvre requiert l'accord seulement pour les exécutions dans lesquelles la condition de stabilité (au moins α processus stables) est satisfaite. Ainsi, contrairement aux consensus uniforme et régulier, le consensus abandonnable ne requiert pas l'accord dans toutes les exécutions. Notons que le terme « consensus abandonnable » (en anglais, *abortable consensus*) a été proposé par [Chen, 2006, Guerraoui and Rodrigues, 2006] dans le contexte des systèmes statiques de partition primaire.

Jusqu'à présent, nous n'avons pas présenté précisément le problème de la gestion de groupe de partition primaire ainsi que la mise en œuvre d'une solution basée sur une séquence de consensus. Pour mieux comprendre notre intuition de départ, dans la suite de ce chapitre, nous exposons le problème de la gestion de groupe de partition primaire ainsi qu'une solution basée sur Paxos. Notre solution pour la gestion de groupe partitionnable basée sur l'adaptation de Paxos est présentée dans les chapitres 4 et 5.

Dans la section 3.2, nous présentons les principes de l'algorithme du consensus de Paxos. Puis, dans la section 3.3, nous présentons la déconstruction et la reconstruction de Paxos. Ensuite, dans la section 3.4, nous décrivons les principes d'utilisation de Paxos pour la gestion de groupe de partition primaire. Enfin, dans la section 3.5, nous concluons ce chapitre.

3.2 Principes de l'algorithme du consensus de Paxos

Dans la section 3.2.1, nous présentons les principes de l'algorithme du consensus de Paxos sous plusieurs angles différents : une explication intuitive, une description algorithmique et un scénario d'exécution de l'algorithme. Ensuite, dans la section 3.2.2, nous mettons en évidence les relations entre une machine à états de Paxos et la gestion de groupe de partition primaire de Paxos.

3.2.1 Consensus Synod de Paxos

L'algorithme du consensus Synod est le cœur de Paxos [Lamport, 1998, Lamport, 2001]. Pour mettre en place une séquence de consensus, Paxos utilise plusieurs instances de l'algorithme Synod. L'algorithme Synod permet de résoudre le problème du consensus dans un système

réparti non byzantin composé de $|\Pi|$ processus. Dans le système considéré, les messages peuvent être dupliqués, perdus, mais ceux qui sont reçus sont supposés non corrompus.

Le consensus est exprimé en termes de trois ensembles d'agents : « proposeurs » (anglicisme construit à partir du mot anglais « *proposers* »), accepteurs (en anglais, *acceptors*) et apprenants (en anglais, *learners*). Un agent représente un rôle qu'un processus peut jouer et un processus peut tenir plusieurs rôles. Un proposeur est un processus qui croît être le leader de Π. Seuls les proposeurs peuvent proposer des valeurs. Le rôle des accepteurs est de choisir une valeur unique parmi les valeurs proposées. Les apprenants prennent connaissance de la valeur choisie en demandant à une majorité d'accepteurs la valeur qu'ils ont choisie. Dans la suite de notre présentation, comme nous étudions la gestion de groupe de partition primaire, tous les processus de Π tiennent les trois rôles simultanément : ils sont tous potentiellement des processus qui peuvent être des leaders et proposer de retirer des processus défaillants de Π ou d'ajouter de nouveaux participants à Π, des accepteurs qui votent pour le choix de la prochaine vue, et des apprenants qui installent la prochaine vue.

L'algorithme du consensus Synod met en œuvre une séquence de scrutins (en anglais, *ballots*) totalement ordonnés. À chaque fois qu'un proposeur propose une valeur, il associe sa valeur à un numéro de scrutin. Les proposeurs choisissent les numéros de scrutins de façon strictement croissante. Le but de l'algorithme est de trouver le premier scrutin associé à une majorité d'accepteurs qui choisissent la valeur proposée dans ce scrutin comme le résultat du consensus. Avant que le système ne devienne stable [15], plusieurs proposeurs peuvent *1)* croire qu'ils sont leader de Π, *2)* proposer des valeurs, et donc *3)* créer des scrutins concurrents. Un processus qui estime être le leader refuse d'accepter les scrutins des autres processus. Ainsi, tant qu'il existe des scrutins concurrents, aucun proposeur ne réussit à faire choisir sa valeur par une majorité d'accepteurs, et les apprenants n'obtiennent pas la même valeur d'une majorité d'accepteurs. Le consensus n'est donc pas atteint.

Pour garantir la vivacité du consensus, c'est-à-dire qu'ultimement une valeur est choisie dans un scrutin, l'algorithme Synod requiert l'existence d'un proposeur distingué appelé le leader ultime. En parallèle de la génération des scrutins et des votes, les processus élisent le leader de Π. Quand le système est stable, une majorité de processus peuvent communiquer entre eux pour élire un leader ultime. Ainsi, le leader ultime devient à terme le seul proposeur qui peut continuer à proposer des valeurs. Et, ultimement, il réussit à faire choisir sa valeur par une majorité d'accepteurs dans un scrutin donné.

Quant aux propriétés de sûreté du consensus, c'est-à-dire la validité uniforme, l'intégrité uniforme et l'accord uniforme, elles sont préservées indépendamment du succès ou de l'échec de l'élection du leader. Intuitivement, cela provient du fait qu'une valeur est choisie si elle est choisie par une majorité d'accepteurs. Or, l'intersection de deux majorités d'accepteurs associées à deux scrutins différents est non vide. Par conséquent, si une valeur est choisie alors elle est unique et irrévocable.

La figure 3.1 représente le modèle d'exécution de l'algorithme Synod le plus simple. Dans cette exécution, plusieurs proposeurs proposent des valeurs. Il existe une période de temps

15. Pendant une période de temps suffisamment longue.

suffisamment longue dans laquelle le leader ultime réussit à convaincre une majorité d'accepteurs de choisir sa valeur. Cette valeur choisie est unique et irrévocable. Les apprenants prennent ensuite connaissance de la valeur choisie en demandant à une majorité d'accepteurs la valeur qu'ils ont choisie.

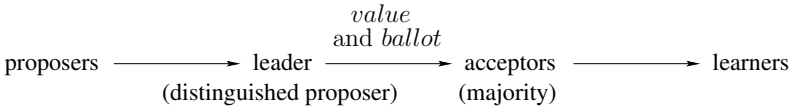

FIGURE 3.1 – Modèle d'exécution de l'algorithme du consensus de Paxos.

Nous présentons les principes de l'algorithme Paxos plus précisément dans ce qui suit. L'algorithme est divisé en deux phases :

– phase 1 (ou phase de lecture) :
  1. un proposeur prépare une proposition sous la forme d'une requête prepare et la soumet à une majorité d'accepteurs. La requête est associée à un numéro de scrutin *sc*. Le proposeur choisit de façon strictement croissante les numéros de scrutins qu'il associe à ses requêtes prepare. De plus, les numéros de scrutins choisis par tous les proposeurs sont totalement ordonnés ;
  2. quand un accepteur reçoit une requête `prepare` avec un numéro de scrutin *sc*, si *sc* est strictement plus grand que tous les numéros de scrutins associés à des requêtes `prepare` déjà rencontrées, alors l'accepteur répond à l'expéditeur avec un message d'acquittement, incluant la dernière valeur acceptée par l'accepteur. Il promet aussi de ne pas accepter de proposition avec un numéro de scrutin plus petit que *sc*[16] ;

– phase 2 (ou phase d'écriture) :
  1. si un proposeur reçoit avec succès un acquittement de sa requête prepare (associée au numéro de scrutin *sc*) d'une majorité d'accepteurs, alors il envoie à tous les accepteurs une proposition avec le numéro de scrutin *sc* sous forme d'une requête `accept` incluant une valeur *val*. *val* est la valeur de la proposition associée au plus grand numéro de scrutin parmi les messages d'acquittement reçus ou une valeur choisie par le proposeur si les acquittements ne s'accompagnent d'aucune valeur ;
  2. si un accepteur reçoit une requête `accept` incluant une proposition avec un numéro de scrutin *sc*, il accepte la valeur incluse dans la proposition à condition qu'il n'ait pas déjà répondu à un proposeur suite à la réception d'une requête `prepare` d'une proposition avec un numéro de scrutin strictement plus grand que *sc*[17].

La figure 3.2 détaille un scénario d'exécution de l'algorithme du consensus Synod dans laquelle il y a plusieurs scrutins concurrents. Les phases de lecture et d'écriture sont représentées

16. Dans le cas où l'accepteur a déjà accepté une proposition avec un numéro de scrutin plus grand que *sc*, l'accepteur peut notifier le proposeur pour que ce dernier abandonne sa proposition. Il s'agit d'une optimisation de l'algorithme. Comme indiqué dans [Lamport, 2001], cette optimisation n'altère pas la correction de l'algorithme.

17. Sinon, comme dans la phase 1, l'accepteur peut notifier le proposeur pour que ce dernier abandonne sa proposition.

respectivement par les lettres R et W. Chaque phase de lecture ou d'écriture y est représentée par un rectangle. La longueur de ce rectangle représente la durée d'exécution de cette phase. Si la phase est terminée, le rectangle est plein. Si elle a été abandonnée, un zigzag en fin de rectangle le signale. Dans ce scénario, un ensemble de trois processus p_2, p_3 et p_4 exécutent le consensus pour choisir une valeur unique. Nous considérons que les processus p_2 et p_4 sont les seuls qui croient être leader et qui proposent des valeurs. Pour garantir que l'ordre des scrutins choisis par les proposeurs soit total, nous considérons que chaque proposeur qui propose une valeur pour la première fois doit choisir le numéro de scrutin qui est son identifiant. Ensuite, les proposeurs incrémentent leur numéro de scrutin avec un pas [18] de 3, qui correspond au nombre de processus. Nous notons par $B_{i,j}$ le numéro de scrutin associé à la i^e (avec $i \geqslant 1$) proposition de p_j (avec $j \in [2,4]$), qui est égal à $3(i-1)+j$. À l'instant t_1, le proposeur p_2 envoie sa requête prepare avec un numéro de scrutin 2 à une majorité d'accepteurs. Lors de la réception de la requête prepare de p_2, l'accepteur p_3 accepte cette requête car p_3 n'a encore vu aucun scrutin avec un numéro supérieur à 2 et envoie un message d'acquittement à p_2 incluant sa valeur $\perp$ dernièrement acceptée. $\perp$ représente le fait que p_3 n'a proposé aucune valeur et n'a accepté aucune valeur (c'est-à-dire qu'il n'a reçu aucun message accept avec un numéro de scrutin plus grand que ceux déjà rencontrés). Cela n'est pas le cas pour le processus p_4. Lors de la réception du message prepare de p_2, comme le proposeur p_4 vient de démarrer une proposition avec un numéro de scrutin $4 > 2$, p_4 refuse la requête de p_2 et notifie ce dernier que, dans le scrutin 2, aucune valeur ne peut être choisie. Ainsi, p_2 abandonne son scrutin dès qu'il reçoit le refus de p_4. De la même manière que p_2, p_4 abandonne son scrutin numéro 4 car p_2 continue à proposer une valeur de façon concurrente.

Pour son scrutin numéro 5, p_2 réussit à terminer avec succès la phase de lecture car il a reçu un acquittement d'une majorité d'accepteurs (incluant p_3 et p_2 qui reçoivent son propre message). p_2 commence ensuite la phase d'écriture du scrutin 5 en envoyant un message accept incluant sa valeur val_2 à tous les accepteurs. Lors de la réception du message accept, p_3 accepte la valeur val_2 de p_2, en remplaçant sa valeur $\perp$ dernièrement acceptée par val_2. Cependant, p_2 ne réussit pas à terminer la phase d'écriture de ce scrutin. En effet, lors de la réception du message accept, p_4 refuse la requête de p_2 car il vient de démarrer une nouvelle proposition associée au scrutin numéro $7 > 5$. C'est le même cas pour le scrutin numéro 7 de p_4. À partir de l'instant t_5 et avant l'instant t_i (avec $i > 5$), les proposeurs p_2 et p_4 peuvent continuer à proposer des valeurs de façon concurrente, et aucun d'entre eux ne réussit à convaincre une majorité d'accepteurs. Il doit exister un instant t_i après lequel le système devient stable [19], p_2 devient le leader ultime. p_2 est le seul proposeur qui continue à proposer des valeurs. Il doit exister un instant $t_s \geqslant t_i$ auquel p_2 propose une valeur et réussit ultimement à convaincre une majorité d'accepteurs de choisir sa valeur. p_2 décide de manière irrévocable val_2 et ne propose plus de valeur. Lors de la réception du message accept avec un numéro de scrutin $B_{i,2} > 7$ de p_2 incluant val_2, p_3 accepte cette valeur, en remplaçant sa valeur val_4 dernièrement acceptée par val_2. Dans une perspective

18. Notons que le choix du pas avec une valeur 3 n'est pas obligatoire. Les proposeurs peuvent incrémenter leur numéro de scrutin avec d'autres valeurs à condition que tous les scrutins soient totalement ordonnés.

19. Pour une période de temps suffisamment longue durant laquelle une majorité des processus peuvent communiquer entre eux.

d'implantation de l'algorithme, une fois qu'un leader ultime sait que sa valeur est choisie, il peut continuer à diffuser de manière fiable sa valeur à tous les processus qui exécutent le consensus. Ainsi, lors de la réception d'un tel message incluant la valeur choisie, les processus qui exécutent le consensus prennent connaissance de la valeur choisie.

	p, val, b : p proposes value val with ballot number b
R : read phase	p, PR, b : p promises not to accept prepare request with ballot number less than b
W : write phase	p, PW, b, val : p promises not to accept prepare request with ballot number less than b and val is the last accepted value by p

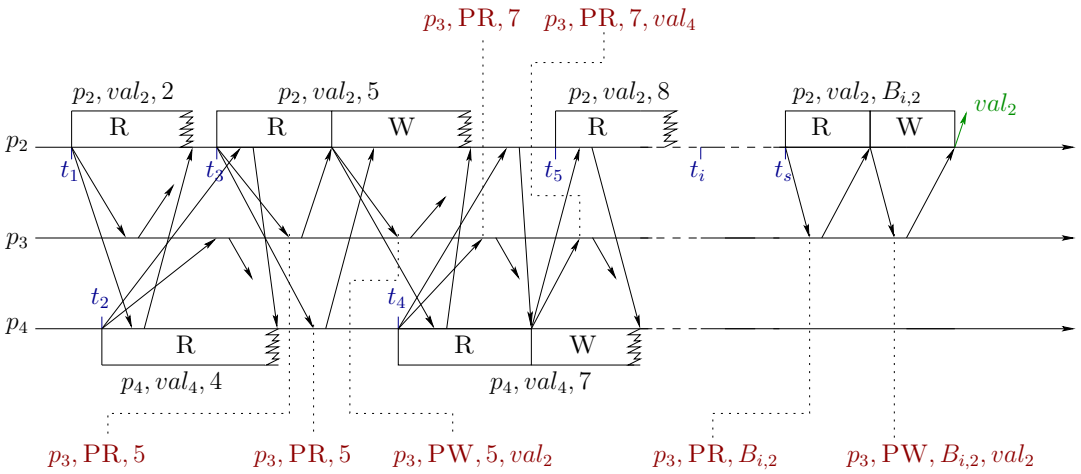

FIGURE 3.2 – Scénario d'exécution du consensus de Paxos : plusieurs scrutins concurrents.

Dans cette section, nous avons présenté les principes de l'algorithme du consensus Synod qui est le cœur de Paxos. Paxos est constitué d'une séquence d'instances du consensus Synod. Nous sommes maintenant en mesure de présenter les relations entre Paxos et la gestion de groupe de partition primaire. Dans la section 3.2.2, nous présentons une solution de la gestion de groupe de partition primaire basée sur Paxos

3.2.2 Machine à états Paxos et gestion de groupe dans Paxos

[Lamport, 2001] présente l'implantation d'une machine à états [Schneider, 1990] en utilisant Paxos. Nous nous servons de cet exemple pour montrer que Paxos peut être utilisé pour implanter la gestion de groupe de partition primaire.

La machine à états est constituée d'un groupe de processus serveurs qui exécutent des commandes soumises par des processus clients. Les processus serveurs doivent exécuter la même séquence de commandes. La machine à états est déterministe : deux serveurs qui exécutent la même séquence de commandes génèrent la même séquence de valeurs de sortie.

Pour garantir que tous les processus serveurs exécutent la même séquence de commandes,

chaque commande soumise est associée à une instance de consensus parmi les processus serveurs, et chaque processus serveur joue les trois rôles de proposeur, accepteur et apprenant. La valeur choisie dans la i^e instance de consensus correspond à la i^e commande, dénotée c_i. Un serveur est autorisé à recevoir plusieurs commandes. Dès lors, il est possible que les commandes $c_1 \dots c_{i-1}$ et $c_{i+1} \dots c_m$ soient prêtes à être exécutées avant que la commande c_i ne soit exécutée. Un tel scénario est possible car les instances de consensus de la 1^{re} à la $(i-1)^e$ et de la $(i+1)^e$ à la m^e sont terminées avant que la i^e instance de consensus ne soit terminée. Le serveur peut exécuter les commandes $c_1 \dots c_{i-1}$, mais pas les commandes $c_{i+1} \dots c_m$ car la commande c_i n'a pas été exécutée.

Pour gérer la dynamicité du groupe de processus serveurs, [Lamport, 2001] propose d'associer les différentes instances de consensus de Paxos aux différents groupes de serveurs, c'est-à-dire de déterminer quels sont les serveurs qui exécutent telles et telles instances de consensus. Pour cela, le groupe de processus serveurs courant est intégré à l'état courant de la machine. Cet ensemble peut être changé par exemple suite à des commandes de changement de groupe. Par exemple, si le serveur est autorisé à prendre θ commandes, alors le groupe de processus serveurs qui exécutent la $(i+\theta)^e$ instance de consensus est spécifié par l'état de la machine après l'exécution de la commande c_i.

Dans l'utilisation de Paxos pour la gestion de groupe de partition primaire, une tentative d'installation d'une nouvelle vue correspond à une commande soumise pour changer la valeur de l'ensemble des serveurs. Plusieurs commandes de changement de vue peuvent être soumises, mais une seule commande est exécutée à la fois. Le groupe qui exécute la i^e instance de consensus doit être spécifié par l'état de la machine après l'exécution de la commande c_{i-1}. Nous avons alors une correspondance entre les processus serveurs qui exécutent la même séquence de commandes et les processus du groupe qui installent la même séquence de vues.

Le tableau 3.1 met en évidence la correspondance entre la machine à état Paxos et la gestion de groupe de partition primaire. La première colonne représente les éléments de la machine à états tandis que la troisième colonne représente les éléments de la gestion de groupe de partition primaire. La deuxième colonne indique les relations entre les éléments de la machine à états et ceux de la gestion de groupe. Chaque ligne précise la correspondance entre un élément de la machine à états et un élément de la gestion de groupe. La première ligne montre qu'une exécution d'une commande peut correspondre à une installation d'une vue, c'est-à-dire l'exécution de la i^e commande peut correspondre à la i^e installation de la vue du groupe. La deuxième ligne indique que la majorité des processus serveurs associés à l'état courant de la machine à états correspond à la majorité des membres de la vue courante du groupe. La troisième ligne souligne le fait que le serveur est autorisé à prendre plusieurs commandes ($\theta > 1$) et à les exécuter les unes après les autres immédiatement. Cela n'est pas le cas pour la gestion de groupe. En effet, $\theta = 1$ signifie que la vue successeur immédiate du groupe doit être spécifiée par les membres de la vue courante. Par conséquent, l'implantation de la machine à états Paxos peut être utilisée pour l'implantation de la gestion de groupe de partition primaire.

La sûreté du consensus Synod de Paxos est préservée dans toutes les exécutions. Quant à la vivacité du système, elle est conditionnelle et n'est garantie que quand le système devient stable pour une période assez longue. Cette hypothèse de stabilité n'est pas définie

Machine à état	Correspondance	Gestion de groupe
Exécution d'une commande (i^e commande)	Correspond possiblement à	Installation d'une vue (i^e vue du groupe)
Majorité des processus serveurs de l'état courant	Correspond à	Majorité des membres de la vue courante
Plusieurs commandes ($\theta \geqslant 1$)	*Versus*	Une vue ($\theta = 1$)

TABLE 3.1 – Paxos et gestion de groupe de partition primaire.

dans [Lamport, 1998, Lamport, 2001]. Définir la vivacité du système est d'autant plus intéressant que nous considérons les systèmes partitionnables dans lesquels l'accord sur la vue finale parmi les processus dépend de la stabilité du système. En outre, comme indiqué dans [Boichat et al., 2003a] : « *Paxos implique plusieurs détails épineux et il est difficile d'identifier toutes ses abstractions comprenant l'algorithme.* » Les auteurs fournissent des méthodes qui permettent de déconstruire et reconstruire Paxos et ses variantes avec deux abstractions qui sont le registre ultime et le leader ultime. Le leader ultime permet de garantir la vivacité du consensus de Paxos tandis que le registre ultime préserve la sûreté du consensus de Paxos. Par conséquent, nous utilisons les méthodes proposées par [Boichat et al., 2003a, Boichat et al., 2003b] pour adapter Paxos à la gestion de groupe partitionnable dans les chapitres 4 et 5. Auparavant, la section 3.3 explique comment [Boichat et al., 2003a, Boichat et al., 2003b] déconstruisent le consensus de Paxos et le reconstruisent dans le contexte des systèmes statiques de partition primaire.

3.3 Déconstruction et reconstruction du consensus de Paxos

Dans cette section, nous présentons la déconstruction et la reconstruction de Paxos. Tout d'abord, dans la section 3.3.1, nous décrivons les principes de construction et reconstruction. Ensuite, dans la section 3.3.2, nous détaillons l'architecture du consensus Synod de Paxos reconstruit.

3.3.1 Principes de déconstruction et reconstruction

La déconstruction de l'algorithme du consensus Synod de Paxos correspond à la factorisation de ses principes algorithmiques en deux modules [Boichat et al., 2003a] : le registre ultime (en anglais, *eventual register* ou $\diamondsuit Register$) et l'élection du leader ultime (en anglais, *eventual leader* ou $\diamondsuit Leader$).

Dans les systèmes de partition primaire, chaque instance du consensus Synod de Paxos est une combinaison d'une instance du module $\diamondsuit Register$ et d'une instance du module $\diamondsuit Leader$. Ainsi, [Boichat et al., 2003a] considère Paxos comme une séquence de combinaison des deux modules $\diamondsuit Register$ et $\diamondsuit Leader$. La reconstruction de Paxos consiste à utiliser les abstractions des modules $\diamondsuit Register$ et $\diamondsuit Leader$. [Boichat et al., 2003b] montre comment utiliser $\diamondsuit Register$

et $\diamondsuit$*Leader* pour construire différentes variantes de Paxos. Chaque variante peut être implantée en modifiant seulement l'implantation de l'un des deux modules $\diamondsuit$*Register* et $\diamondsuit$*Leader*, ou les deux.

L'ensemble $\mathbb{E}$ du chapitre 2 est étendu pour inclure l'ensemble des événements suivants :
- $\{R\text{-}\texttt{propose}(p, val) | p \in \mathbb{P}, val \in 2^{\mathbb{P}}\}$;
- $\{R\text{-}\texttt{return}(p, val) | p \in \mathbb{P}, val \in 2^{\mathbb{P}}\}$;
- $\{L\text{-}\texttt{leader}(p) | p \in \mathbb{P}\}$;
- $\{L\text{-}\texttt{return}(p, l) | p \in \mathbb{P}, l \in \mathbb{P}\}$.
- $\{s\text{-}\texttt{send}(p, m) | p \in \mathbb{P}, m \in \mathbb{M}\}$;
- $\{s\text{-}\texttt{receive}(p, m) | p \in \mathbb{P}, m \in \mathbb{M}\}$.

3.3.2 Architecture du consensus

La figure 3.3 représente l'architecture du consensus. La couche la plus basse est le module réseau. [Boichat et al., 2003a] considère que chaque paire de processus dans le réseau est reliée par un lien bidirectionnel et équitable (définition 10 [20]). Le rôle du module Retransmission module est de permettre aux processus de tolérer les pertes de messages échangés à travers les liens équitables. Les modules $\diamondsuit$*Register* et $\diamondsuit$*Leader* sont construits au dessus du module de retransmission. Le composant consensus $\mathcal{C}$ utilise $\diamondsuit$*Register* pour proposer et décider une valeur respectivement via les primitives *R-propose* et *R-return*. Le composant $\diamondsuit$*Leader* est utilisé par le composant consensus pour connaître le leader courant du processus via les primitives *L-leader* et *L-return*. Au dessus du composant consensus, le composant gestion de groupe $\mathcal{GM}$ peut utiliser le consensus pour installer des nouvelles vues via les primitives $\mathcal{C}$*-propose* et $\mathcal{C}$*-return*. Afin de tolérer les pertes de messages, le module $\mathcal{GM}$ utilise le module Retransmission module via ses primitives *s-send* et *s-receive* afin de permettre au processus d'envoyer chaque vue installée aux membres de cette vue de manière fiable.

3.3.2.1 Module de retransmission

Le module de retransmission permet aux processus de communiquer entre eux malgré les pertes de messages dans les liens équitables. Il est défini par deux primitives *s-send* et *s-receive*. Pour envoyer (respectivement recevoir) le message m, le processus p utilise le module de retransmission et exécute la primitive *s-send*(p, m) (respectivement *s-receive*(p, m)). Le module de retransmission satisfait la propriété suivante, que nous nommons quasi-fiabilité-∞ :

Propriété 9. Quasi-fiabilité-∞ *[Boichat et al., 2003a]. Soit deux processus p et q. Si p envoie (via la primitive s-send) un message m au processus q, alors q reçoit m de p (via la primitive s-receive) une infinité de fois.*

La propriété de lien quasi-fiable-∞ est plus forte que celle de lien quasi-fiable (« *pour tout $k \geqslant 1$, si deux processus p et q sont corrects, et si p envoie un message m à q k fois, alors q reçoit ultimement m de p exactement k fois.* » [Aguilera and Toueg, 1997]). Néanmoins, les

20. Appelé, en anglais, *fair loss* dans [Boichat et al., 2003a].

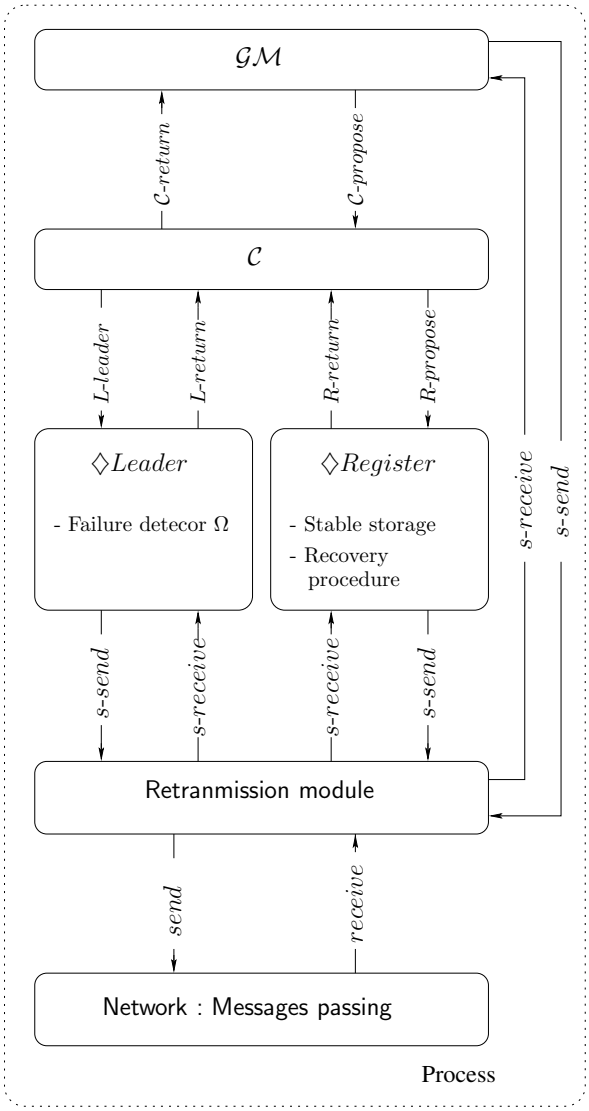

FIGURE 3.3 – Architecture du consensus Paxos reconstruite.

liens quasi-fiables-∞ et quasi-fiables peuvent être implantés au dessus des liens équitables en retransmettant les messages envoyés. [Boichat et al., 2003a] fournit une implantation du module de retransmission dans laquelle chaque message envoyé par le processus expéditeur est retransmis infiniment. Cependant, notons dès maintenant que les propriétés des liens quasi-fiables-∞ et quasi-fiables ne peuvent pas être garanties quand le système se partitionne de façon permanente, c'est-à-dire lorsque deux processus p et q qui ne défaillent pas ne peuvent plus communiquer entre eux.

Dans les sections 3.3.2.2 et 3.3.2.3, nous présentons respectivement les modules $\diamondsuit$*Register* et $\diamondsuit$*Leader* qui sont implantés au dessus du module de retransmission.

3.3.2.2 Registre ultime

Le module $\diamondsuit$*Register* matérialise une mémoire (ou registre) stable partagée par les processus courant du système avec la sémantique « écrire une fois » (en anglais, *write-once*). L'écriture dans ce registre exprime l'acte de choisir une valeur unique et irrévocable. Pour garantir l'accord entre les processus, l'implantation de $\diamondsuit$*Register* impose qu'une majorité de processus dans le système soient corrects. Le processus p peut accéder au module $\diamondsuit$*Register* en invoquant la primitive *R-propose*(p, val) pour tenter d'écrire sa valeur val dans le registre. Nous considérons que $val \in 2^{\mathbb{P}}$ est un ensemble d'identifiants de processus. L'invocation de *R-propose*(p, val) par le processus p peut soit retourner une valeur (c'est-à-dire, le processus p décide), soit retourner $\perp$ s'il existe un autre processus qui a proposé une valeur concurremment (c'est-à-dire, le processus abandonne). $\diamondsuit$*Register* satisfait les trois propriétés suivantes.

Propriété 10. R-Terminaison *[Boichat et al., 2003a]. 1) si un processus propose, soit il défaille, soit l'invocation se termine ; 2) si un processus unique propose une infinité de fois, alors il décide ultimement.*

Propriété 11. R-Accord *[Boichat et al., 2003a]. Deux processus ne décident pas différemment.*

Propriété 12. R-Validité *[Boichat et al., 2003a]. Si un processus décide une valeur val, alors val a été proposée par un processus.*

Les propriétés R-Validité et R-Accord correspondent respectivement aux propriétés de validité et d'accord du consensus. Quant à la propriété R-Terminaison, elle est plus faible que la propriété de terminaison du consensus dans le sens où s'il existe plusieurs processus qui proposent des valeurs, alors aucun d'entre eux ne réussit à décider. La propriété R-Terminaison exprime le fait qu'une valeur est choisie (dans le sens du consensus Synod de Paxos) uniquement s'il existe un leader ultime. L'existence du leader ultime est capturée par le module $\diamondsuit$*Leader* décrit dans la section suivante.

3.3.2.3 Leader ultime

Le module $\diamondsuit$*Leader* est un oracle réparti qui élit ultimement un leader ultime parmi les processus du système. Il modélise les hypothèses de synchronie ajoutées au système afin de

garantir la progression et la terminaison de l'exécution de l'algorithme du consensus. En présence d'au moins un processus correct dans le système durant l'exécution, $\diamondsuit Leader$ correspond au détecteur de défaillances non fiable Ω défini dans [Chandra et al., 1996a].

Le processus p peut accéder au module $\diamondsuit Leader$ en invoquant la primitive $L\text{-}leader(p)$. L'invocation se termine quand la primitive $L\text{-}return(l)$ est exécutée. l est l'identifiant du processus (possiblement p) que p croit être le leader courant. $\diamondsuit Leader$ satisfait les trois propriétés suivantes.

Propriété 13. L-Terminaison *[Boichat et al., 2003a]. Après avoir invoqué la primitive $L\text{-}leader(p)$, soit le processus p défaille, soit ultimement la primitive $L\text{-}return(p, l)$ est exécutée.*

Propriété 14. L-Accord *[Boichat et al., 2003a]. Il existe un instant après lequel deux processus ne choisissent pas différents leaders.*

Propriété 15. L-Validité *[Boichat et al., 2003a]. Il existe un instant après lequel le leader est correct.*

3.3.2.4 Implantation du consensus

L'algorithme 1 correspond à une implantation du consensus de Paxos. Il satisfait les quatre propriétés du consensus. La décision du consensus correspond à la décision retournée par $\diamondsuit Register$. Les propriétés de sûreté du consensus, c'est-à-dire la validité, l'intégrité et l'accord, sont assurées par l'existence d'une majorité de processus corrects dans le système et proviennent directement des propriétés de $\diamondsuit Register$. La propriété de vivacité du consensus, c'est-à-dire la terminaison, est garantie par $\diamondsuit Leader$ qui retourne ultimement le même processus correct pour tous les processus corrects dans le système.

La section 3.4 présente une solution de la gestion de groupe de partition primaire basée sur l'utilisation de Paxos. C'est la même approche qui est utilisée dans le chapitre 5 pour construire le consensus abandonnable pour les MANETs, puis la gestion de groupe partitionnable.

3.4 Utilisation de Paxos pour la gestion de groupe de partition primaire

Dans la section 3.4.1, nous décrivons les principes de l'utilisation du consensus Synod de Paxos pour une solution de la gestion de groupe de partition primaire en utilisant les modules $\diamondsuit Register$ et $\diamondsuit Leader$. Ensuite, dans les sections 3.4.2 et 3.4.3, nous présentons respectivement la spécification de la gestion de groupe de partition primaire et une implantation. Enfin, dans la section 3.4.4, nous fournissons une idée de preuve de la correction de l'implantation.

3.4.1 Principes de l'utilisation

L'utilisation d'une instance du module $\diamondsuit Register$ en conjonction avec une instance du module $\diamondsuit Leader$ permet de résoudre une instance du consensus de Paxos. $\diamondsuit Register$ peut être

Algorithm 1 Implantation du consensus pour le processus p

```
1  init():
2  Begin
3      register ← create ♢Register;                    {Instance of ♢Register }
4      leader ← create ♢Leader;                        {Instance of ♢Legister}
5      decision ← (p, ⊥);
6  End
7
8  Procedure C-propose(p, val)
9  Begin
10     While decision = (p, ⊥) do
11         If leader.L-leader(p) = p then
12             decision ← register.R-propose(p, val);
13         generate C-return(decision);
14 End
```

implantée avec une majorité de processus corrects parmi un groupe de processus fixes (l'ensemble Π des processus qui constituent le système est fixe). Dans les systèmes dynamiques, la notion de majorité de processus corrects est spécifique à la composition courante du groupe, c'est-à-dire les membres de la vue courante du groupe. Ainsi, par la suite, l'expression « majorité des processus corrects » est utilisée dans le contexte d'une vue v donnée. Nous supposons qu'il existe au moins une majorité de processus corrects parmi les membres de la vue courante du groupe.

Les implantations de $\diamondsuit Register$ et $\diamondsuit Leader$ peuvent être utilisées pour obtenir un consensus dans le contexte d'une vue v donnée. Pour ce faire, l'ensemble des processus Π doit être remplacé par l'ensemble des membres de la vue courante du groupe. Étant donné que les identifiants des vues sont totalement ordonnés, ils peuvent être utilisés pour estampiller les messages échangés entre les processus dans le contexte d'une vue donnée. Ainsi, les messages qui sont associés aux anciennes vues sont ignorés. La valeur décidée par le consensus dans une vue v correspond à la vue successeur de v.

3.4.2 Spécification

Nous considérons seulement les propriétés qui sont directement liées à la gestion de groupe de partition primaire. Les propriétés de diffusion des messages ne sont pas considérées. Nous avons vu dans la section 1.4.1 que la gestion de groupe de partition primaire satisfait les propriétés de sûreté de base qui sont l'auto-inclusion (si le processus p installe la vue v alors p est un membre de v) et la monotonicité locale (si le processus p installe la vue v après avoir installé la vue v' alors l'identifiant de v est plus grand que celui de v').

En plus des deux propriétés précédentes, la gestion de groupe de partition primaire impose un ordre total sur les identifiants des vues installées par les processus. Pour cela, [Chockler et al., 2001] suppose qu'il existe une fonction $\Psi : \mathbb{V}^* \times \mathbb{N}$, où $\mathbb{V}^*$ est l'ensemble des vues installées, telle que Ψ satisfait la propriété partition primaire qui est définie comme suit.

Propriété 16. Partition primaire *[Chockler et al., 2001]. Il existe une fonction* $\Psi : \mathbb{V}^* \to \mathbb{N}$*, où* $\mathbb{V}^* \subset \mathbb{V}$ *est l'ensemble des vues installées dans une exécution, telle que* Ψ *satisfait la propriété suivante : pour chaque vue* v *avec* $\Psi(v) > 1$ *il existe la vue* v'*, telle que* $\Psi(v) = \Psi(v') + 1$ *et un processus* p *membre de* v *qui installe* v *dans* v' *(c'est-à-dire,* v *est la vue successeur de* v'*). Formellement,*

$$\begin{aligned}\exists \Psi : \{v | \exists p : install(p, v)\} \to \mathbb{N} : \\ \Psi(v) = \Psi(v') \implies v = v' \\ \wedge\ \forall v : \Psi(v) > 1 \implies \exists v' : \Big(\Psi(v) = \Psi(v') + 1 \\ \wedge\ \exists p \in v.members : install\_in(p, v, v')\Big)\end{aligned}$$

La propriété partition primaire impose que les identifiants des vues installées par les processus dans une exécution forment une séquence totalement ordonnée. Cette séquence est représentée par la fonction f qui prend en entrée une vue installée et fournit en sortie l'identifiant de cette vue.

La fonction Ψ implique que l'intersection des membres de deux vues consécutives dans la séquence est non vide. Plus précisément, soient v' et v deux vues consécutives installées par les processus, il existe au moins un processus correct qui survit entre la première vue et la deuxième vue ($v'.members \cap v.members \neq \emptyset$). En revanche, la fonction Ψ ne spécifie pas les contraintes qui doivent être satisfaites afin de garantir l'accord entre les processus. En outre, elle ne définit pas les événements qui génèrent l'exclusion (l'inclusion) d'un processus dans (à) une vue. Enfin, [Chockler et al., 2001] ne fournit aucune implantation de la gestion de groupe de partition primaire.

En suivant l'approche de [Chandra et al., 1996b, Guerraoui and Schiper, 2001], nous décomposons la propriété partition primaire en précisant les contraintes du système ainsi que le contexte d'exclusion ou d'inclusion d'un processus. Les trois propriétés ci-dessous sont inspirées de [Guerraoui and Schiper, 2001] que nous présentons de manière personnelle comme suit :

Propriété 17. Terminaison. *Si le processus* p *membre de* v' *1) souhaite quitter* v'*, 2) défaille ou 3) souhaite ajouter le processus* q *(*$q \notin v'.members$*) alors chaque processus correct* $r \in v'.members$ *ne souhaitant pas quitter* v' [21] *installe ultimement* v*, avec* $v.id = v'.id + 1$*. Formellement,*

21. Dans [Guerraoui and Schiper, 2001], le fait que le processus r ne souhaite pas quitter v' n'est pas défini.

$$\exists v' \exists t' \exists p \in v'.members\ \exists q \notin v'.members : \Big(H(p,t') = \texttt{remove}(p,p,v') \vee H(p,t') = \texttt{add}(p,q,v') \vee H(p,t') = \texttt{crashed\_in}(p,v')\Big)$$
$$\Rightarrow \Big(\forall r \in v'.members\ \exists t > t' : H(r,t) = \neg\texttt{crashed\_in}(r,v') \wedge \neg\texttt{remove}(r,r,v') \wedge \texttt{install\_in}(r,v',v)\Big)$$

Propriété 18. Accord. *Si le processus p installe deux vues consécutives v' et v telles que $v.id = v'.id + 1$ et si le processus q membre de v' installe v'', alors $v = v''$. Formellement,* $\Big(install\_in(p,v') \wedge v.id = v'.id + 1 \wedge install\_in(p,v',v'')\Big) \Rightarrow v = v''$.

Propriété 19. Validité faible. *Si le processus p installe deux vues consécutives v' et v telles que $v.id = v'.id + 1$, alors : 1) $v.members \neq v'.members$; 2) $v.members$ contient une majorité des processus de $v'.members$; 3) pour chaque processus $q \in v'.members \setminus v.members$, il existe au moins un processus r qui installe v dans v' qui a souhaité retirer q de v' (soit q est suspecté d'être défaillant par r, soit q souhaite quitter v') ; et 4) pour chaque processus $q \in v.members \setminus v'.members$, il existe au moins un processus r qui installe v dans v' et a souhaité ajouter q dans v. Formellement,*

$$\exists v' \exists v \exists p \exists t : H(p,t) = \texttt{install\_in}(p,v,v') \wedge v.id = v'.id + 1$$
$$\Rightarrow \Big(v'.members \neq v.members\Big)$$
$$\wedge \Big(|v.members \cap v'.members| \geqslant \lceil \tfrac{|v'.members|+1}{2} \rceil\Big)$$
$$\wedge \Big(\forall q \in v'.members \setminus v.members\ \exists t' < t\ \exists t'' > t' \exists r : H(p,t'') = \texttt{install\_in}(r,v,v') \wedge H(p,t') = \texttt{remove}(r,q,v')\Big)$$
$$\wedge \Big(\forall q \in v.members \setminus v'.members\ \exists t' < t\ \exists t'' > t' \exists r : H(r,t'') = \texttt{install\_in}(r,v,v') \wedge H(p,t') = \texttt{add}(r,q,v')\Big)$$

3.4.3 Implantation

Dans cette section, nous présentons une implantation de la gestion de groupe de partition primaire. Pour simplifier la présentation, nous supposons que chaque paire de processus est reliée par un lien de communication logique et équitable. Un processus p peut envoyer (recevoir) des messages à (d') un autre processus q dans le système en exécutant la primitive *s-send* (*s-receive*).

Par la suite, nous considérons que les messages sont typés et peuvent contenir des informations telles que l'identité du processus expéditeur et l'identité du (des) processus destinataire(s). Chaque message m est dénoté par $m \stackrel{\text{déf}}{=} (originalSender, \langle \text{TYPE} \mid attribute_1, attribute_2 \ldots \rangle, destinatorsSet)$, où TYPE est le type du message, et $originalSender, destinatorsSet, attribute_1, attribute_2 \ldots$ sont des informations du message. $originalSender$ correspond à l'expéditeur originel de m et $dest$ est l'ensemble des

destinataires. Quand les destinataires du messages ne sont pas précisés, nous écrivons « * » à la place de *destinatorsSet* qui signifie que n'importe quel processus dans le système est un destinataire de m.

L'algorithme 2 implante la gestion de groupe de partition primaire en utilisant le consensus comme service de base. Les identifiants sont utilisés pour estampiller de manière unique les messages des différentes instances de consensus. Ainsi, l'identifiant de l'instance du consensus courant correspond à l'identifiant de la vue courante. Les processus ne traitent que les messages associés à la vue courante.

À l'initialisation (lignes 2–7), tous les processus dans le groupe installent la même vue initiale contenant tous les membres $members_{init} \neq \emptyset$ du groupe. Les variables v et id correspondent respectivement à la vue courante du groupe et à son identifiant. Les membres de la nouvelle vue potentielle sont stockés dans la variable $members_{new}$. $members_{new}$ est initialisée à $v.members$.

Dans la tâche 1 (lignes 11–18), le service du consensus est utilisé par le processus afin de proposer la nouvelle vue détectée par le processus. Lorsque le processus p souhaite installer une nouvelle vue, il propose cette nouvelle vue potentielle aux membres de sa vue courante v en invoquant la primitive $\mathcal{C}\text{-}propose(p, v, members_{new})$ du consensus. La nouvelle vue potentielle peut être proposée si elle inclut au moins la majorité des membres de v. La décision retournée par le consensus correspond à la vue successeur de v. Cette décision est envoyée par p à tous les processus dans $v.members$.

Notons que plusieurs processus de la vue courante peuvent proposer différentes valeurs (vues), mais seul le leader ultime de la vue courante réussit ultimement à convaincre les autres processus d'accepter sa proposition. Les différentes vues proposées par les processus reflètent les différentes détections non-fiables de la composition du groupe, mais traduisent aussi les scénarios dans lesquels les processus souhaitent quitter ou rejoindre le groupe volontairement.

Dans la tâche 2, lors de la réception du message $(q, \langle \text{DECISION} \mid decision \rangle, dest)$, p vérifie si la vue décidée peut être la vue successeur de sa vue courante v ($decision_q.id \geqslant v.id \wedge p \in decision_q.members \wedge decision_q.members \neq v.members$). Si c'est le cas, p installe la nouvelle vue et prépare l'installation d'une vue successeur en affectant $decision_q.members$ à la variable $members_{new}$. Ensuite, la couche applicative est notifiée du changement de vue par un événement de type *view_change*. Sinon, p ignore le message reçu car il s'agit d'un message associé à une ancienne vue.

Remarquons qu'un processus n'installe une nouvelle vue que s'il est membre de cette vue. En revanche, un processus correct dans v' peut installer la nouvelle vue v sans avoir la garantie que tous les processus corrects de v installent v, mais seulement la majorité des membres corrects de v'. En effet, un processus peut être exclu du groupe même s'il est faussement suspecté d'être défaillant.

3.4.4 Idée de la preuve de correction

Dans cette section, nous présentons une idée de la preuve de correction montrant que l'algorithme 2 satisfait les propriétés de l'auto-inclusion, de la monotonicité locale, de la validité faible, de l'accord et de la terminaison.

Algorithm 2 Implantation de la gestion de groupe de partition primaire pour le processus p

init():
Begin
  $members_{new} \leftarrow members_{init} \subseteq \mathbb{P}$; {Members of a new view}
  $v \leftarrow (0, members_{new})$; {Current view of the group}
  $c \leftarrow$ *create* $\mathcal{C}$; {Consensus instance)}
  $decision_p \leftarrow (p, v.id, \perp)$; {Decision, a 3-tuple $(process, identifier, members)$}
End

Task 1: **upon** *1)* [p **suspects** $q, r... \in v.members \vee q, r... \in v.members$ **wish to leave** v]
  $\vee$ *2)* [p **wishes to add** $q, r... \notin v.members$]
Begin
  If 1) then
    $members_{new} \leftarrow members_{new} \setminus \{q, r...\}$;
  If 2) then
    $members_{new} \leftarrow members_{new} \cup \{q, r...\}$;
  If $(|members_{new} \cap v.members| \geqslant \lceil \frac{|v.members|+1}{2} \rceil)$ then
    $decision_p \leftarrow c.propose(p, v, members_{new})$; {Consensus decision}
    s-send$(p, \langle \text{DECISION} \mid decision_p \rangle, v.members)$ {To all the processes in $v.members$}
End

Task 2: upon reception of $(q, \langle \text{DECISION} \mid decision_q \rangle, dest)$
Begin
  If $decision_q.id \geqslant v.id$ then
    If $p \in decision_q.members \wedge decision_q.members \neq v.members)$ then
      $members \leftarrow decision_q.members$;
      $v \leftarrow (decision_q.id, decision_q.members)$; {Install a new view}
      $members_{new} \leftarrow decision_q.members$; {Prepare for a new view installation}
      generate *view_change*(p, v); {Notify application layer}
End

Auto-inclusion. Soit v la nouvelle vue installée par le processus p (ligne 26) alors $p \in v.members$ (ligne 24). De plus, $p \in members_{init}$ (ligne 3). Par conséquent, toute vue v installée par le processus p inclut nécessairement p ($p \in v.members$).

Monotonicité locale. Considérons que le processus p installe la vue v dans v'. Nous avons alors $v.id > v'.id + 1$ car les identifiants des instances de consensus suivent un ordre strictement croissant (lignes 23 et 26).

Validité faible. Considérons que le processus p installe deux vues consécutives v' et v, c'est-à-dire que $\texttt{install\_in}(p, v', v) \wedge v.id = v'.id + 1$. Nous avons alors :

1. d'après la ligne 26, le processus p installe la vue v dans la vue v' si $v.members \neq v'.members$;
2. $v.members$ contient une majorité de processus dans $v'.members$. Par hypothèse, il existe une majorité de processus corrects dans chaque vue installée. Une majorité des membres de la (nouvelle) vue v est acceptée au moins par une majorité des processus de la vue (courante) v', c'est-à-dire $|members_{new} \cap v.members| \geqslant \lceil \frac{|v.members|+1}{2} \rceil$;
3. pour chaque processus $q \in v'.members \setminus v.members$, soit q est suspecté d'être défaillant, soit q a souhaité quitter v'. Il peut exister plusieurs processus membres de la vue v' qui tentent d'installer une nouvelle vue. Soit p le premier d'entre eux. p invoque la primitive $\mathcal{C}$-$propose(v, members_{new})$, avec $members_{new} \neq v'.members$, *i)* s'il souhaite quitter le groupe ou *ii)* s'il suspecte un processus q d'être défaillant. Dans le cas *i)*, aucun processus ne propose $members_{new}$ qui inclut p. Ainsi, la nouvelle vue v installée par les processus du groupe ne contient pas p. Nous prouvons le cas *ii)* pas contraposition en supposant qu'il n'existe aucun processus correct dans $v'.members$ qui ait suspecté q. Cela veut dire qu'aucun processus correct dans $v'.members$ n'a invoqué la primitive $\mathcal{C}$-$propose(v, members_{new})$, avec $q \notin members_{new}$. Par conséquent, la nouvelle vue v installée par les processus du groupe contient q ;
4. pour chaque processus $q \in v.members \setminus v'.members$, il existe au moins un processus p qui a souhaité ajouter q. Comme dans le cas *3)-ii)*, il existe au moins un processus correct dans $v'.members$ qui a souhaité ajouter q dans le groupe.

Accord. La preuve peut être obtenue à partir de la propriété *R-Accord uniforme*, l'unicité des identifiants des instances de consensus et du fait que l'identifiant de toute nouvelle vue installée est incrémenté de 1 par rapport à la vue courante.

Terminaison. La preuve peut être obtenue à partir de la propriété *R-Terminaison*, l'unicité des identifiants des instances de consensus et du fait que l'identifiant de toute nouvelle vue installée est incrémenté de 1 par rapport à la vue courante.

3.5 Conclusion

Dans ce chapitre, nous avons présenté une solution de la gestion de groupe de partition primaire en la transformant en une séquence de consensus Synod de Paxos. Paxos a été choisi car il met en place une séquence de consensus de façon native. Les principes de l'algorithme du consensus Synod de Paxos intègrent plusieurs facettes implicites qui peuvent être optimisées. Plusieurs variantes de Paxos sont proposées dans la littérature pour mettre en place ces optimisations, chaque variante étant configurée pour un environnement spécifique. À notre connaissance, aucune de ces variantes ne considère les spécificités des systèmes dynamiques/avec recouvrement/avec partitionnement. La sûreté du consensus Synod est garantie dans toutes les exécutions grâce à une majorité des processus dans le système qui sont corrects. Quant à la vivacité du consensus, elle est conditionnelle et n'est garantie que quand le système devient stable pour une période assez longue. Cette hypothèse de stabilité n'est pas définie. Définir la vivacité du système est d'autant plus intéressant que nous considérons les systèmes partitionnables dans lesquels l'accord sur la vue finale parmi les processus dépend de la stabilité du système. En outre, comme indiqué dans [Boichat et al., 2003a] : « *Paxos implique plusieurs détails épineux et il est difficile d'identifier toutes ses abstractions comprenant l'algorithme.* » Les auteurs fournissent des méthodes qui déconstruisent le consensus de Paxos et le reconstruisent avec deux modules qui sont le registre ultime et le leader ultime. Le leader ultime définit la vivacité du consensus de Paxos tandis que le registre ultime préserve la sûreté du consensus de Paxos. Ensuite, la gestion de groupe de partition primaire est proposée comme une séquence de consensus, eux-mêmes construits au dessus des modules $\diamondsuit Register$ et $\diamondsuit Leader$.

Dans le prochain chapitre, nous présentons une spécification de la gestion de groupe partitionnable basée sur une séquence de consensus abandonnables. Le consensus abandonnable est construit au dessus de versions adaptées des modules $\diamondsuit Register$ et $\diamondsuit Leader$ pour les systèmes dynamiques partitionnables.

Chapitre 4

Spécification de la gestion de groupe partitionnable

Sommaire

4.1 Introduction

Dans le chapitre 3, nous avons présenté une solution de la gestion de groupe de partition primaire basée sur Paxos et implantée comme une séquence de consensus Synod de Paxos. Dans cette implantation, l'algorithme du consensus Synod est déconstruit puis reconstruit avec les modules du registre ultime $\diamondsuit$*Register* et du leader ultime $\diamondsuit$*Leader* proposés par [Boichat et al., 2003a, Boichat et al., 2003b]. Dans ce chapitre, par analogie, nous proposons une adaptation de Paxos pour la gestion de groupe partitionnable. Le résultat de l'adaptation est

une spécification du consensus abandonnable construit au-dessus de versions adaptées des modules $\diamondsuit Register$ et $\diamondsuit Leader$ pour les systèmes dynamiques partitionnables. Ensuite, la gestion de groupe partitionnable est construite comme une séquence de consensus abandonnables.

Le chapitre est organisé comme suit. Dans la section 4.2, nous détaillons les principes d'adaptation de Paxos pour une solution de la gestion de groupe partitionnable. Dans la section 4.3, nous spécifions le consensus abandonnable comme une combinaison de deux nouveaux modules : le registre ultime par partition $\diamondsuit\mathcal{RPP}$ et le détecteur ultime des α participants dans une partition $\diamondsuit\mathcal{PPD}$. Puis, dans la section 4.4, nous présentons l'architecture et la spécification de la gestion de groupe partitionnable utilisant le consensus abandonnable. Enfin, dans la section 4.5, nous concluons ce chapitre et présentons quelques perspectives. Le chapitre 5 sera l'objet de l'implantation de chacun des modules $\diamondsuit\mathcal{PPD}$, $\diamondsuit\mathcal{RPP}$, consensus abandonnable et gestion de groupe partitionnable.

4.2 Principes d'adaptation de Paxos pour la gestion de groupe partitionnable

Dans notre adaptation, comme dans Paxos, les participants au consensus sont répartis en trois ensembles d'agents : les proposeurs, les accepteurs et les apprenants. Pour la gestion de groupe partitionnable, durant une exécution, les processus tiennent les trois rôles simultanément. Dans notre modèle de système dynamique avec partitionnement, α constitue le nombre minimum d'accepteurs dans un groupe. Par « minimum », nous entendons que le consensus peut être atteint aussitôt que la condition de stabilité est satisfaite : $|\diamondsuit PART_p| \geqslant \alpha_p$.

Un proposeur est un processus qui croît être le leader de sa partition constituée d'au moins α processus stables (noté αSet). Les accepteurs constituent un sous-ensemble S des processus de αSet tel que $\alpha \leqslant |S| \leqslant |\alpha Set|$; ils représentent des serveurs au sens Paxos du terme. La particularité de notre approche consiste en la manière avec laquelle les processus détectent cet ensemble αSet. Intuitivement, un processus qui croît être le leader de son αSet prend le rôle de proposeur et propose une valeur (une nouvelle vue potentielle dont l'ensemble des membres est inclus dans ou égal à l'ensemble αSet). Le proposeur peut continuer à proposer des valeurs tant qu'il croit être le leader de αSet et tant qu'il n'a pas décidé. Avant que la condition de stabilité ne soit satisfaite, il peut y avoir plusieurs proposeurs qui tentent de convaincre les autres accepteurs de se mettre d'accord sur leur valeur, mais aucun d'entre eux ne réussit. Quand la condition de stabilité est satisfaite, selon la définition 18 d'une partition stable, les processus stables de αSet peuvent communiquer entre eux pour élire le leader ultime de αSet. Ainsi, le leader ultime devient à terme le seul proposeur de αSet qui peut proposer des valeurs. L'ensemble des accepteurs stables inclus dans αSet reçoivent ultimement la proposition du leader ultime. Chaque accepteur accepte la proposition reçue uniquement si la vue successeur proposée par le leader ultime est égale ou incluse dans l'ensemble des processus stables αSet connu par l'accepteur. Si ce n'est pas le cas, le proposeur est informé que sa proposition est abandonnée.

L'adaptation donne lieu à une spécification d'un autre type de consensus strictement plus faible que les consensus régulier et uniforme dans le sens où l'accord n'est pas garanti dans

toutes les exécutions, mais uniquement dans les exécutions dans lesquelles il y a au moins α processus stables dans le groupe. Ce consensus, que nous appelons consensus abandonnable, est spécifique à la gestion de groupe partitionnable dont la spécification autorise le désaccord entre processus tant que la condition de stabilité n'est pas satisfaite. Dans la suite, le consensus abandonnable est une combinaison des modules $\Diamond\mathcal{PPD}$ et $\Diamond\mathcal{RPP}$ qui sont des versions adaptées de $\Diamond Leader$ et $\Diamond Register$ pour les systèmes partitionnables. $\Diamond\mathcal{PPD}$ spécifie la vivacité du consensus abandonnable en capturant l'existence de l'ensemble αSet ainsi que le leader ultime de cet ensemble tandis que $\Diamond\mathcal{RPP}$ préserve la sûreté du consensus abandonnable. Ensuite, la gestion de groupe partitionnable est proposée comme une séquence de consensus abandonnables.

La figure 4.1 représente un modèle d'exécution du consensus abandonnable pour les systèmes partitionnables. Avant que la condition de stabilité ne soit satisfaite, il peut y avoir plusieurs proposeurs qui proposent des valeurs et tentent de convaincre les autres accepteurs. De manière optimiste, selon notre modèle, il existe un instant après lequel la condition de stabilité est satisfaite et le leader ultime de αSet réussit à convaincre les accepteurs dans αSet de choisir sa valeur. Les apprenants prennent alors connaissance de la valeur choisie en demandant à l'ensemble des accepteurs de la partition la valeur qu'ils ont choisie. Rappelons que dans le cas où il existe moins de α processus stables dans la partition, le consensus est abandonné ; les processus doivent alors déterminer un nouvel ensemble αSet et démarrer une nouvelle instance de consensus abandonnable. Ainsi, la progression et la terminaison de l'exécution de l'algorithme du consensus abandonnable sont garanties par l'existence d'un ensemble de processus qui deviennent ultimement stables (pour une période de temps suffisamment longue).

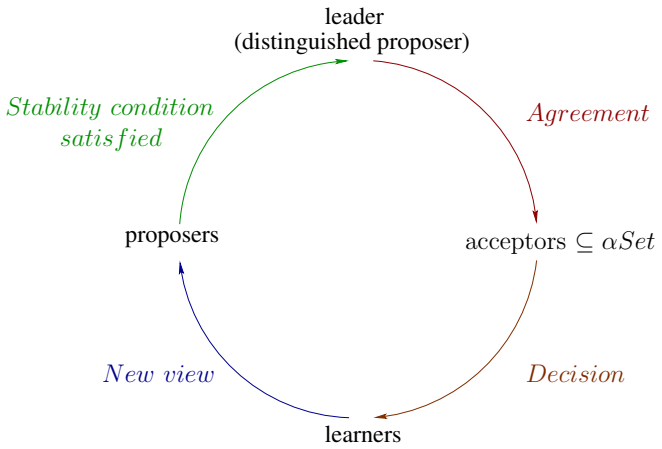

FIGURE 4.1 – Modèle d'exécution de l'algorithme du consensus abandonnable.

4.3 Consensus abandonnable

Cette section présente le consensus abandonnable. Tout d'abord, dans la section 4.3.1, nous exposons l'architecture du consensus abandonnable. Ensuite, dans la section 4.3.2, nous présen-

tons la spécification du consensus abandonnable. Les sections qui suivent spécifient les modules composant le consensus abandonnable : le module de retransmission de messages dans la section 4.3.3, le module $\diamondsuit\mathcal{PPD}$ dans la section 4.3.4, et le module $\diamondsuit\mathcal{RPP}$ dans la section 4.3.5.

4.3.1 Architecture

La figure 4.2 présente l'architecture du consensus abandonnable (en anglais, *abortable consensus* ou $\mathcal{AC}$). $\mathcal{AC}$ est une combinaison de deux abstractions : un détecteur ultime des α participants d'une partition (en anglais, *eventual α partition-participant detector* ou $\diamondsuit\mathcal{PPD}$) et un registre ultime par partition (en anglais, *eventual register per partition* ou $\diamondsuit\mathcal{RPP}$). Le module consensus abandonnable $\mathcal{AC}$ utilise $\diamondsuit\mathcal{PPD}$ via ses primitives $\mathcal{PPD}$*-participants* et $\mathcal{PPD}$*-return* pour construire l'ensemble αSet et élire le leader ultime de αSet. Quant au module $\diamondsuit\mathcal{RPP}$, il est utilisé par le module consensus abandonnable $\mathcal{AC}$ via ses primitives $\mathcal{RPP}$*-propose* et $\mathcal{RPP}$*-return* pour proposer et décider une valeur. $\diamondsuit\mathcal{RPP}$ est implanté au dessus d'un module de retransmission de messages pour les réseaux mobiles spontanés. Le rôle ce module de retransmission est de permettre aux processus de tolérer les pertes de messages échangés entre processus à travers les liens SADDM. Le module $\diamondsuit\mathcal{RPP}$ utilise également le module $\diamondsuit\mathcal{PPD}$ pour décider si une valeur proposée doit être abandonnée quand un ou plusieurs participants disparaissent de la partition. Au dessus du module consensus abandonnable, le module gestion de groupe partitionnable $\mathcal{PGM}$ utilise le consensus abandonnable comme service de base via les primitives $\mathcal{AC}$*-propose* et $\mathcal{AC}$*-return* pour installer les nouvelles vues. Comme le module $\diamondsuit\mathcal{RPP}$, $\mathcal{PGM}$ utilise les primitives du module de retransmission de messages *xbroadcast* et *xreceive* pour envoyer et recevoir les vues installées. Par la suite, nous considérons les événements suivants qui sont ajoutés à l'ensemble $\mathbb{E}$ du chapitre 2 :

- $\{\mathcal{PPD}\texttt{-participants}(p)|p \in \mathbb{P}\}$;
- $\{\mathcal{PPD}\texttt{-return}(p, l, \alpha Set)|p \in \mathbb{P}, l \in \mathbb{P}, \alpha Set \in 2^{\mathbb{P}}\}$;
- $\{\mathcal{RPP}\texttt{-propose}(p, vset, vid)|p \in \mathbb{P}, vset \in 2^{\mathbb{P}}, vid \in \mathbb{N}\}$;
- $\{\mathcal{RPP}\texttt{-return}(p, vset, vid)|p \in \mathbb{P}, val \in 2^{\mathbb{P}}, vid \in \mathbb{N}\}$;
- $\{\mathcal{AC}\texttt{-propose}(p, vset, vid)|p \in \mathbb{P}, val \in 2^{\mathbb{P}}, vid \in \mathbb{N}\}$;
- $\{\mathcal{AC}\texttt{-return}(p, vset, vid)|p \in \mathbb{P}, val \in 2^{\mathbb{P}}, vid \in \mathbb{N}\}$;
- $\{\texttt{xbroadcast}(p, m)|p \in \mathbb{P}, m \in \mathbb{M}\}$;
- $\{\texttt{xreceive}(p, m)|p \in \mathbb{P}, m \in \mathbb{M}\}$.

4.3.2 Spécification du consensus abandonnable

À la différence des consensus régulier et uniforme, comme dans Synod, chaque processus dans le système n'est pas supposé proposer une valeur : seulement les processus qui détectent l'existence d'un ensemble αSet constitué d'au moins α processus et dont ils sont le leader, proposent une valeur. α est un paramètre de l'application répartie et correspond au nombre minimum de processus stables [22] dans une partition. Pour le processus p, le consensus abandonnable est

22. Nous rappelons que, dans une exécution, les processus stables et instables peuvent cohabiter dans une partition. Si l'ensemble αSet_p associé au processus p est stable, nous avons alors $\alpha Set \subseteq \diamondsuit PART_p \subseteq PART_p$ (d'après

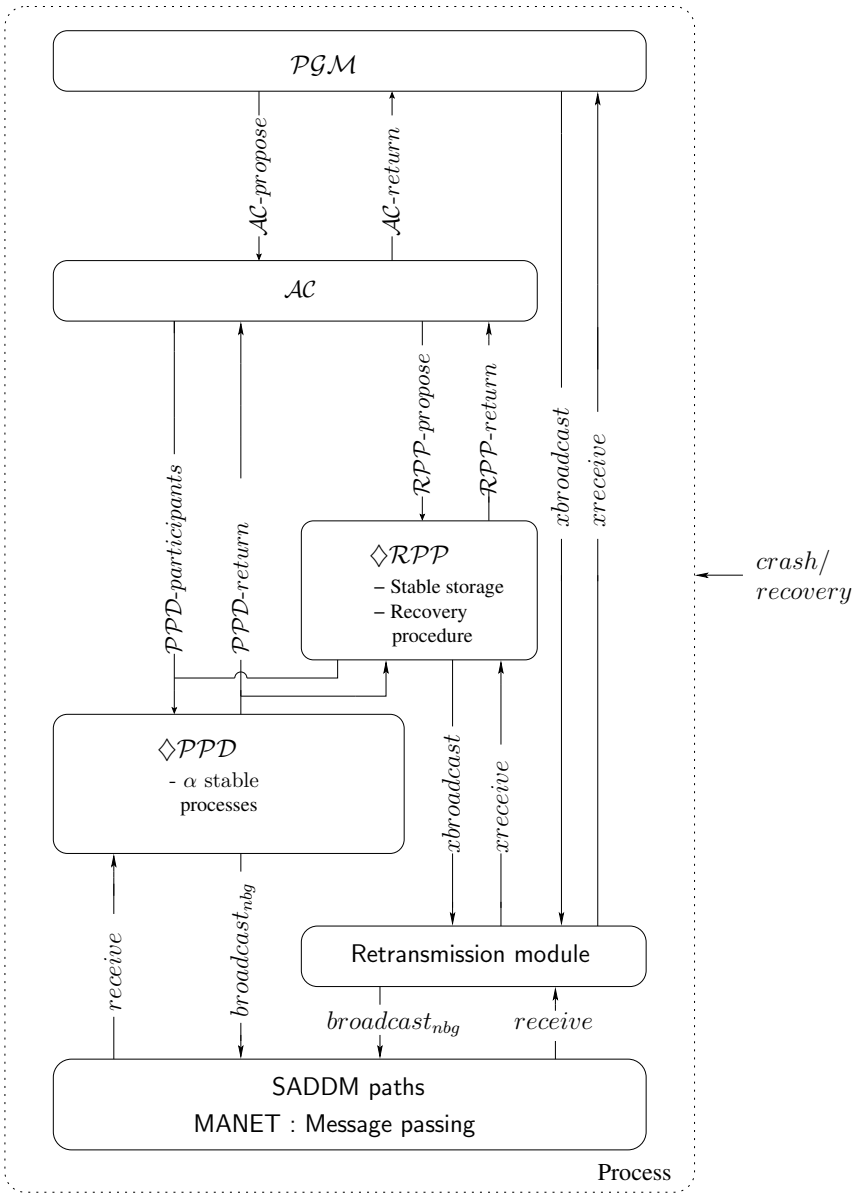

FIGURE 4.2 – Architecture du consensus abandonnable.

défini par les deux primitives $\mathcal{AC}$-*propose* et $\mathcal{AC}$-*return*. Dans le rôle du proposeur, p propose un ensemble $vset$ et un identifiant vid qui correspondent aux membres et à l'identifiant de la vue potentiellement successeur de la vue courante. Cela est effectué en invoquant la primitive $\mathcal{AC}$-*propose*$(vset, vid)$. L'invocation se termine avec le retour d'une valeur, c'est-à-dire l'appel de la primitive $\mathcal{AC}$-*return*$(vset, vid)$. Les valeurs $vset$ et vid retournées peuvent être $\perp$, signifiant alors que le consensus a été abandonné. Si $vset \neq \perp \ \wedge vid \neq \perp$ alors le consensus est dit atteint. Le consensus abandonnable satisfait les trois propriétés suivantes.

Propriété 20. $\mathcal{AC}$-Terminaison. *Dans une partition, soit un processus p qui propose une valeur. S'il existe un ensemble αSet constitué d'au moins α processus stables incluant p, alors p décide ultimement. Sinon, p abandonne. Formellement,*

$\exists p\, \exists vset\, \exists vid\, \exists t' :$

$$H(p,t') = \mathcal{AC}\text{-}\texttt{propose}(p, vset, vid)$$
$$\wedge \Big[\Big(\forall t'' \geqslant t' : (\exists \alpha Set \subseteq \Diamond PART_p : |\alpha Set| > \alpha) \Rightarrow \exists vset'\, \exists vid'\, \exists t > t' :$$
$$H(p,t) = \mathcal{AC}\text{-}\texttt{return}(p, vset', vid') \wedge vset' \neq \perp \wedge vid' \geqslant vid\Big)$$
$$\vee \Big(\forall t'' \geqslant t' : (\nexists \alpha Set \in \Diamond PART_p : |\alpha Set| > \alpha) \Rightarrow \exists t > t' : H(p,t) = \mathcal{AC}\text{-}\texttt{return}(p, \perp, \perp)\Big)\Big]$$

Dans la propriété précédente, remarquons que la vue retournée possède le numéro vid' et non vid, avec $vid' \geqslant vid$. La raison est la suivante. Le processus p propose une valeur en indiquant un ensemble de processus et un identifiant. Avant d'atteindre le consensus, p peut être amené à proposer plusieurs fois sa valeur, par exemple en attendant que la partition se stabilise. Or, pour ne pas confondre les différentes propositions, à chaque fois que p propose, il fournit le même ensemble $vset$ mais avec un identifiant strictement supérieur à l'identifiant de la proposition précédente. Par conséquent, l'identifiant vid' retourné à terme par le module $\mathcal{AC}$ est supérieur ou égal à vid. Ce raisonnement s'applique aussi aux propriétés qui suivent.

Propriété 21. $\mathcal{AC}$-Accord par partition. *Dans une partition, deux processus dans l'ensemble αSet des processus stables ne décident pas ultimement différentes valeurs. Formellement,*

$\exists vset\, \exists vset'\, \exists vid\, \exists vid'\, \forall p, q \in \alpha Set \subseteq \Diamond PART_p\, \exists t'\ \forall t \geqslant t'\ \exists t_1 \geqslant t\ \exists t_2 \geqslant t :$

$$\Big(H(p,t_1) = \mathcal{AC}\text{-}\texttt{return}(p, vset', vid')$$
$$\wedge\ H(q,t_2) = \mathcal{AC}\text{-}\texttt{return}(p, vset, vid)\Big) \Rightarrow (vset = vset \wedge vid = vid')$$

Propriété 22. $\mathcal{AC}$-Validité par partition. *Dans une partition, si un processus p décide la valeur $val = (vset, vid)$, alors $val' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans vset. Formellement,*

$\exists vset\, \exists vid\, \exists p \exists t :$

$$H(p,t) = \mathcal{RPP}\text{-}\texttt{return}(p, vset, vid)$$
$$\Rightarrow \Big(\exists q \in vset\, \exists t' < t : H(q,t') = \mathcal{RPP}\text{-}\texttt{propose}(q, vset, vid') \wedge vid' \leqslant vid\Big)$$

les définitions 16 et 18). En outre, les processus stables sont corrects, mais l'inverse n'est pas nécessairement vrai.

4.3.3 Spécification du module de retransmission pour les réseaux mobiles spontanés

Nous avons vu dans la section 3.3.2.1 que dans le contexte des systèmes de partition primaire l'utilisation d'un module de retransmission est nécessaire afin de tolérer les pertes de messages échangés entre processus corrects à travers des liens équitables. Le module de retransmission est implanté en retransmettant infiniment les messages échangés. Ainsi, tout message envoyé est garanti d'être reçu ultimement par le destinataire. Cependant, dans les systèmes dynamiques avec partitionnement tels que les MANETs, il est possible que deux processus qui ne défaillent pas ne peuvent pas communiquer entre eux. Autrement dit, bien que deux processus corrects diffusent infiniment souvent leurs messages, il est possible qu'un message diffusé infiniment ne soit pas reçu par le destinataire car le réseau peut se partitionner de façon permanente.

Dans notre modèle de système dynamique avec partitionnement, deux processus stables peuvent communiquer entre eux s'il existe un chemin SADDM de l'un vers l'autre. Le module de retransmission est utilisé pour tolérer les pertes de messages acheminés à travers les liens SADDM. Ce module est défini par deux primitives *xbroadcast* et *xreceive*. Pour envoyer (respectivement recevoir) le message m, le processus p utilise le module de retransmission en exécutant la primitive $xbroadcast(m)$ (respectivement $xreceive(m)$). Le module de retransmission pour les MANETs satisfait la propriété suivante :

Propriété 23. Diffusion fiable à un ensemble de destinataires. *Soit p un processus stable et $dest$ un ensemble de processus stables inclus dans $\Diamond PART_p$. Si p diffuse un message m β^n fois et si $dest$ est l'ensemble des processus destinataires de m, alors le message m originellement diffusé par p arrive à chaque processus $q \in dest$ au plus en $\beta^n\eta + n\delta$ secondes, avec η la période de temps maximale qui sépare deux diffusions consécutives du message m et n la longueur du chemin SADDM le plus long entre p et un processus $r \in dest$. Formellement,*
$\exists I = [t_1, t_2]\ \exists dest\, \forall q \in dest\, \exists r \in dest\, \exists m\ \exists n :$

$$\Big(m = (p, \langle \text{TYPE} \mid p, \dots \rangle, dest)$$
$$\wedge\ n = |saddm\_path(p \dots r)| \geqslant |saddm\_path(p \dots q)|$$
$$\wedge\ \forall t_{i \in [1,\beta^n]} \in I\ \exists \eta = (max(t_{i+1} - t_i) : \forall i \in [1, \beta^n]) :$$
$$H(p, t_i) = \texttt{broadcast}_{nbg}(m)\Big) \Rightarrow\ \Big(\forall q \in dest\ \exists t \leqslant t_1 + \beta^n\eta + n\delta : H(q, t) = \texttt{receive}(q, m)\Big)$$

Pour satisfaire la propriété de diffusion fiable à un ensemble de destinataires, les messages diffusés doivent être retransmis de telle sorte que la période de temps maximale η qui sépare deux diffusions consécutives de m ne devienne pas trop grande. Rappelons que β et δ sont des constantes utilisées dans la définition d'un lien SADDM (définition 16).

4.3.4 Spécification du détecteur ultime des α participants d'une partition

Un *détecteur ultime des α participants d'une partition* $\Diamond\mathcal{PPD}$ est un oracle réparti qui détecte ultimement l'ensemble αSet des processus stables dans une partition. Les processus dans αSet sont sélectionnés selon le critère de stabilité $HB(p, q) \geqslant \text{THRESHOLD}_p$ (défini dans la section 2.2.7). Ensuite, trivialement, $\Diamond\mathcal{PPD}$ détecte ultimement un leader stable unique parmi

les processus de αSet. En outre, ultimement, la composition du groupe de processus dans αSet s'arrête de changer, mais les processus ne savent pas quand cela se produit. Ainsi, plusieurs processus peuvent croire pendant un moment qu'ils sont leaders. Cependant, quand la condition de stabilité est satisfaite ($|\diamondsuit PART_p| \geqslant \alpha_p$) après l'instant de stabilisation local à la partition (ou pour une période de temps suffisamment longue), un leader unique est élu. $\diamondsuit\mathcal{PPD}$ satisfait les deux propriétés suivantes.

Propriété 24. $\mathcal{PPD}$-Stabilité de αSet. *Il existe un instant après lequel deux processus p et q d'un ensemble αSet de processus stables possèdent la même connaissance du même ensemble αSet. Formellement,*
$\forall p, q \in \alpha Set \subseteq \diamondsuit PART_p \; \exists t' \; \forall t \geqslant t' \; \exists t_1 \geqslant t \; \exists t_2 \geqslant t :$
$\Big(H(p, t_1) = \mathcal{PPD}\texttt{-return}(p, l, \alpha Set) \;\wedge\; H(q, t_2) = \mathcal{PPD}\texttt{-return}(q, l', \alpha Set')\Big) \Rightarrow \alpha Set = \alpha Set'$

Propriété 25. $\mathcal{PPD}$-Accord sur l'élection du leader. *Il existe un instant après lequel deux processus p et q d'un ensemble αSet de processus stables élisent le même processus $l \in \alpha Set$ comme le leader. Formellement,*
$\forall p, q \in \alpha Set \subseteq \diamondsuit PART_p \; \forall t \geqslant t' \; \exists t_1 \geqslant t \; \exists t_2 \geqslant t :$
$\Big(H(p, t_1) = \mathcal{PPD}\texttt{-return}(p, l, \alpha Set) \;\wedge\; H(q, t_2) = \mathcal{PPD}\texttt{-return}(q, l', \alpha Set')\Big) \Rightarrow l = l'$

4.3.5 Spécification du registre ultime par partition

Comme dans [Boichat et al., 2003a], un registre par partition matérialise la mémoire stable (le registre) partagée. Cependant, à la différence de [Boichat et al., 2003a], le registre n'est pas nécessairement partagé par tous les processus du système (incluant une majorité des processus corrects), mais par un ensemble constitué d'au moins α processus stables d'une partition. En outre, la tentative de stocker une valeur dans le registre peut ne pas aboutir dans deux cas : *1)* dans le cas de contention ou *2)* dans le cas où la condition de stabilité n'est pas satisfaite. Le premier cas est le même que celui du module $\diamondsuit\mathcal{R}egister$: un proposeur peut abandonner une proposition s'il existe un autre proposeur dans la même partition qui a commencé à proposer une valeur de façon concurrente. Notons que dans ce cas, le consensus n'est pas abandonné. Dans le second cas, le proposeur n'abandonne pas seulement sa proposition, mais aussi l'instance du consensus associée car il n'existe pas α processus stables dans sa partition. Le modèle d'exécution pendant une période stable est le suivant : quand la condition de stabilité est satisfaite et si un seul processus continue à proposer une valeur alors la valeur est ultimement stockée dans le registre.

Le processus p tente d'écrire une valeur $val = (vset, vid)$ dans le registre de façon persistante en invoquant la primitive $\mathcal{RPP}$-*propose*$(p, vset, vid)$. Une valeur est retournée suite à l'invocation si la primitive $\mathcal{RPP}$-*return*$(p, vset, vid)$ est exécutée. L'invocation de la primitive $\mathcal{RPP}$-*propose* peut retourner $(p, vset, vid)$ égale à $(p, \perp, \perp)$. Dans ce cas, le consensus est dit abandonné. Sinon, le consensus est dit atteint. $\diamondsuit\mathcal{RPP}$ satisfait les propriétés suivantes.

Propriété 26. $\mathcal{RPP}$-Non trivialité d'abandon. *Dans une partition, s'il existe un ensemble αSet de processus stables constitué d'au moins α processus stables et si un processus p de cet ensemble*

propose une infinité de fois une valeur $val = (vset, vid)$ *avec* $vset \subseteq \alpha Set \wedge |vset| \geqslant \alpha$*, alors* p *décide ultimement* $val' = (vset, vid')$*. S'il n'existe pas un ensemble* αSet *constitué d'au moins* α *processus stables, alors* p *abandonne ; Formellement,*

$$\begin{aligned}
&\exists p\, \exists vset\, \exists t' :\ H(p,t') = \mathcal{AC}\text{-}\texttt{propose}(p, vset, vid) \\
&\wedge \Big[\Big(\forall t'' \geqslant t'\, \exists t_1 \geqslant t'' : \\
&\quad (vset \subseteq \alpha Set \subseteq \Diamond PART_p : \exists vid\ : |vset| \geqslant \alpha\ \wedge\ \mathcal{RPP}\text{-}\texttt{propose}(p, vset, vid) \in H(p, [t'', \infty])) \\
&\quad \Rightarrow\ \exists t_2 \geqslant t_1 : H(p,t_2) = \mathcal{AC}\text{-}\texttt{return}(p, vset, vid') \wedge vset \neq \perp \wedge vid' \geqslant vid\Big) \\
&\quad \vee \Big(\forall t'' \geqslant t' : (\nexists \alpha Set \in \Diamond PART_p : |\alpha Set| > \alpha) \Rightarrow \exists t > t' : H(p,t) = \mathcal{AC}\text{-}\texttt{return}(p, \perp, \perp)\Big)\Big]
\end{aligned}$$

Propriété 27. $\mathcal{RPP}$-Accord par partition. *Idem* $\mathcal{AC}$*-Accord par partition.*

Propriété 28. $\mathcal{RPP}$-Validité par partition. *Idem* $\mathcal{AC}$*-Validité par partition.*

4.4 Gestion de groupe partitionnable

Cette section présente la gestion de groupe partitionnable. Dans la section 4.4.1, nous présentons l'architecture de la gestion de groupe partitionnable. Ensuite, dans la section 4.4.2, nous spécifions la gestion de groupe partitionnable.

Par la suite, nous considérons que les événements suivants sont inclus dans E :

- $\texttt{propose}(p, v) | p \in \mathbb{P}, v \in \mathbb{V}$;
- $\texttt{nack\_view\_change}(p, v) | p \in \mathbb{P}, v \in \mathbb{V}\}$;
- $\texttt{view\_change}(p, v) | p \in \mathbb{P}, v \in \mathbb{V}\}$;
- $\alpha\texttt{\_set}(p, \alpha Set)(p, \alpha Set) | p \in \mathbb{P}, \alpha Set \in 2^{\mathbb{P}}\}$.

4.4.1 Architecture

La figure 4.2 est complétée afin de présenter la gestion de groupe partitionnable. Dans la figure 4.3, la gestion de groupe partitionnable est construite au dessus du consensus abandonnable. Le consensus abandonnable est utilisé comme service de base par le composant gestion de groupe partitionnable pour installer les nouvelles vues. La couche applicative peut notifier les exclusions et les inclusions de processus en proposant d'installer une nouvelle vue via la primitive *propose* du module de gestion de groupe partitionnable. En outre, l'ensemble des membres potentiels de la vue successeur doit être inclus dans l'ensemble αSet. Pour cela, à chaque fois que $\Diamond\mathcal{PPD}$ détecte un changement dans la composition de αSet, il notifie la couche applicative en générant un événement $\alpha\texttt{\_set}(p, \alpha Set)$. Il est alors de la responsabilité de l'application de proposer si besoin une nouvelle vue en appelant la primitive *propose*. Une vue est installée lorsqu'un événement de type *view_change* est retournée par la gestion de groupe partitionnable. Le module de gestion de groupe partitionnable utilise le module de retransmission pour diffuser la nouvelle vue de manière fiable. Lorsque la proposition d'installation d'une vue échoue, c'est-à-dire que le consensus a été abandonnée, le module de gestion de groupe partitionnable notifie la couche applicative avec l'événement `nack_viewchange` que sa requête ne peut pas être satisfaite.

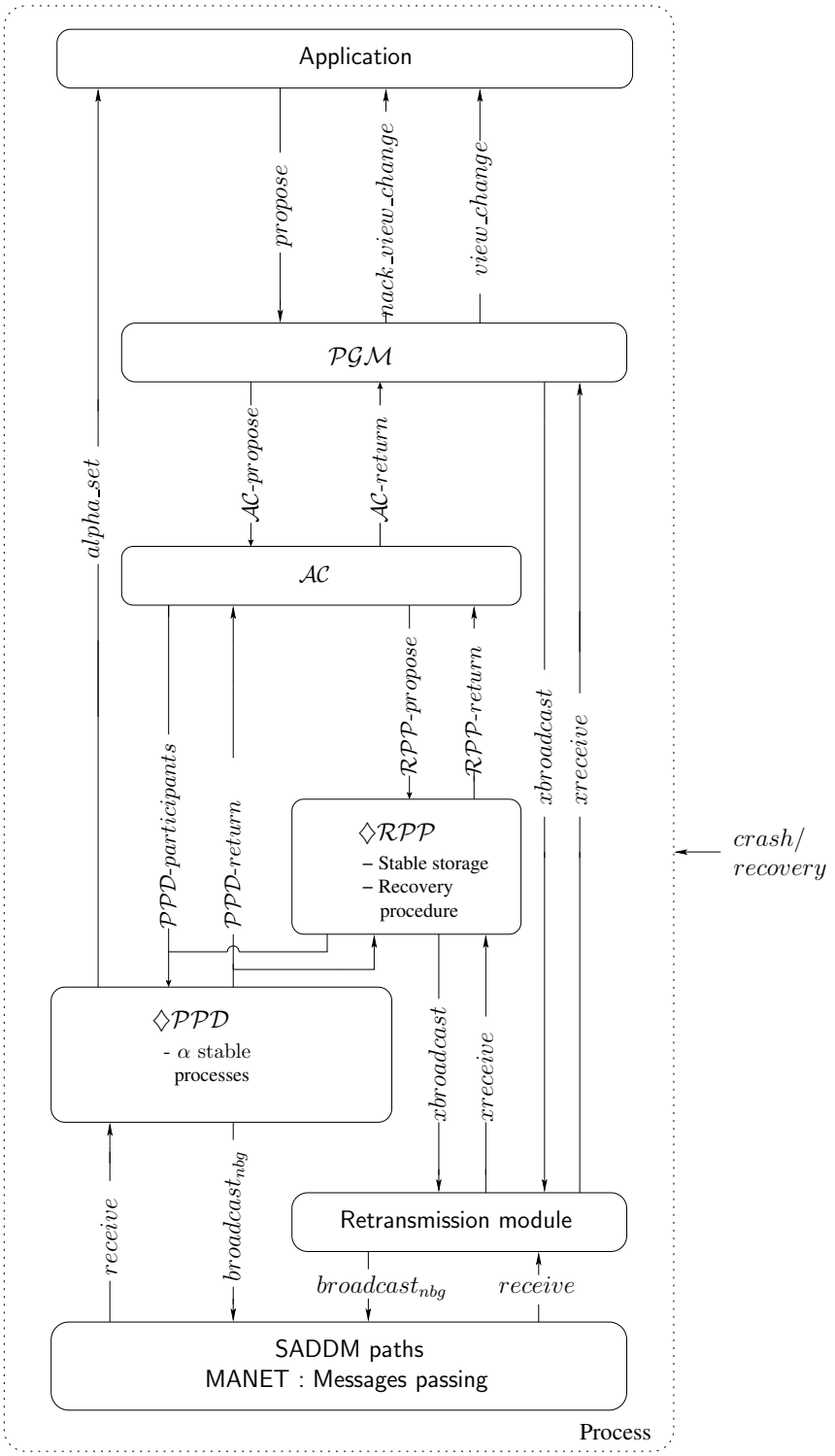

FIGURE 4.3 – Architecture de la gestion de groupe partitionnable.

4.4.2 Spécification

Le service de gestion de groupe partitionnable notifie la couche applicative des changements de vues en générant les événements `view_change`. Plus précisément, l'exécution de l'événement `view_change`(p, v) correspond à l'installation de la nouvelle vue v par p. Le service est partitionnable dans le sens où les processus qui sont dans des groupes différents reçoivent des vues concurrentes qui reflètent la composition de leur partition.

La gestion de groupe partitionnable satisfait les propriétés de sûreté de base que sont l'auto-inclusion (si le processus p installe la vue v alors p est un membre de v) et la monotonicité locale (si le processus p installe la vue v après avoir installé la vue v' alors l'identifiant de v est plus grand que celui de v').

Rappelons que la propriété de monotonicité locale peut être violée dans le cas où un processus défaille et se recouvre en installant sa vue initiale, qui est différente de sa dernière vue installée avant la défaillance. Pour remédier à ce problème, nous imposons que chaque processus qui se recouvre installe la dernière vue qu'il a stocké dans sa mémoire stable.

En outre, observons que, si l'identifiant est un simple entier naturel, le même identifiant peut être utilisé dans deux vues différentes, c'est-à-dire qu'il peut être associé à différents ensembles de processus. Par exemple, il peut exister deux vues v' et v tels que $v' = (3, \{a, b, c\})$ et $v = (3, \{p, q, r\})$, c'est-à-dire $(v'.id = v.id \wedge v'.members \neq v.members)$. Certaines applications requièrent des vues avec des identifiants uniques. En particulier, l'ensemble des identifiants des vues installées doit correspondre à un sous-ensemble d'un ensemble ordonné. Pour garantir l'unicité des identifiants des vues, nous considérons que chaque identifiant de vue est une paire (p, c) où p est l'identifiant d'un processus et c est la valeur d'un compteur local à p. Ainsi, l'unicité des identifiants des processus est assurée. Par la suite, nous considérons que $\mathbb{VID}$ est l'ensemble des identifiants possibles des vues. Chaque $vid \in \mathbb{VID}$ est composé d'un identifiant d'un processus et d'un entier qui représente la valeur d'un compteur local au processus. Nous définissons les relations de comparaison, d'addition et de soustraction sur les identifiants de vues comme suit. Soient $vid' = (p, i)$ et $vid = (q, j)$ deux identifiants dans $\mathbb{VID}$ et k un entier dans $\mathbb{N}$:

- $vid' = vid \stackrel{\text{déf}}{=} (p = q \ \wedge \ i = j)$;
- $vid' > vid \stackrel{\text{déf}}{=} (p > q \vee p = q \ \wedge \ i > j)$;
- $vid' < vid \stackrel{\text{déf}}{=} (p < q \vee p = q \ \wedge \ i < j)$;
- $vid' \geqslant vid \stackrel{\text{déf}}{=} (p > q \wedge p = q \ \wedge \ i \geqslant j)$;
- $vid' \leqslant vid \stackrel{\text{déf}}{=} (p < q \wedge p = q \ \wedge \ i \leqslant j)$;
- $vid + k \stackrel{\text{déf}}{=} vid = (p, i + k)$;
- $vid - k \stackrel{\text{déf}}{=} vid = (p, i - k)$.

Concernant l'installation d'une vue, elle doit être justifiée et refléter le résultat des événements qui se sont produits dans l'environnement. Ces événements doivent être exposés à la couche applicative sensible au contexte, c'est-à-dire aux événements de l'environnement. Ainsi, la gestion de groupe partitionnable satisfait la propriété $\mathcal{PGM}$-Validité.

Propriété 29. $\mathcal{PGM}$-Validité. *Si le processus p installe la vue $v = (vset, vid)$ alors $v' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans $v.members$ et tel que*

$|v.members| \geqslant \alpha_p$. *Formellement,*
$\exists t \exists p : H(t,p) = view\_change(p,v))$
$\Rightarrow \exists t' < t \ \exists q \in v.members : H(q,t') = propose(q,v') \wedge |v.vset| \geqslant \alpha_p \wedge v.vid \geqslant v'.vid$

Pour satisfaire l'accord sur les vues installées parmi un ensemble S de processus dans une partition, le service de gestion de groupe partitionnable doit fournir ultimement la même vue à tous les processus de S lorsque la partition se stabilise. Dans notre spécification, les membres de chaque nouvelle vue constituent un ensemble inclus dans ou égal à l'ensemble αSet courant fourni par le module $\Diamond\mathcal{PPD}$. La vue finale est obtenue lorsque la partition se stabilise et qu'ultimement aucun processus dans S ne souhaite modifier l'ensemble S. La propriété suivante définit l'accord sur la vue finale installée par un groupe de processus stables dans la partition.

Propriété 30. $\mathcal{PGM}$-Accord sur la vue finale. *Dans une partition, s'il existe un ensemble* $S \subseteq \alpha Set \subseteq \Diamond PART_p$ *de processus stables qui souhaitent installer la vue* v_f *avec* $v_f.members = S$*, alors tous les processus dans* S *installent la même vue finale* v_f*. Formellement,*
$\exists S \subseteq \alpha Set \subseteq \Diamond PART_p \ \exists p \ \forall q \in S \ \exists t' \ \forall t \geqslant t' \ \exists v_f : H(p,t') = propose(p,v_f) \Rightarrow last\_view(q,v_f)$

La propriété d'accord sur la vue finale permet de garantir la vivacité d'une partition même si la partition n'est pas complètement stable. Rappelons que dans notre modèle de système dynamique avec partitionnement, les nœuds d'une partition stable ne sont pas nécessairement isolés des autres nœuds du réseau. En effet, en fonction de la connectivité du réseau, il est possible que des nœuds stables et instables cohabitent dans une partition réseau. La particularité de notre approche consiste en la manière avec laquelle chaque processus p détecte un ensemble constitué d'au moins α processus stables. En outre, le fait que les membres de chaque nouvelle vue doivent constituer un sous-ensemble de S qui est inclus dans ou égal à l'ensemble αSet interdit toute installation de vue capricieuse par les processus de S après l'instant de stabilisation locale à la partition, c'est-à-dire après que la condition de stabilité soit satisfaite.

4.5 Conclusion

Dans ce chapitre, nous proposons une spécification de la gestion de groupe partitionnable en adaptant Paxos. Le résultat obtenu est la spécification d'un consensus abandonnable pour réseau partitionnable. Le consensus abandonnable est spécifié à partir des modules $\Diamond\mathcal{PPD}$ et $\Diamond\mathcal{RPP}$ qui, respectivement, abstrait la vivacité dans une partition et encapsule la sûreté dans la même partition. Le rôle de $\Diamond\mathcal{PPD}$ est de capturer le compromis entre l'accord et la progression de l'exécution en détectant ultimement au moins α processus stables dans la partition ainsi que le leader ultime parmi ces processus. Le rôle de $\Diamond\mathcal{RPP}$ est de matérialiser une mémoire partagée tolérante aux défaillances et aux partitionnements. Ensuite, la gestion de groupe partitionnable est résolue en la transformant en une séquence de consensus abandonnables.

Dans notre spécification de la gestion de groupe partitionnable, l'installation de vues capricieuse n'est pas autorisée pendant les périodes stables. En outre, la vivacité d'une partition est assurée même si elle n'est pas complètement stable. Seule l'existence d'un ensemble composé d'au moins α processus stables dans la partition est requise. Cela est garanti par le module

$\diamondsuit\mathcal{PPD}$. L'accord sur la vue finale installée par les processus pendant des périodes stables est assurée par le module $\diamondsuit\mathcal{RPP}$.

Dans le prochain chapitre, nous présentons une implantation de chacun des modules spécifiés dans ce chapitre.

Chapitre 5

Implantation de la gestion de groupe partitionnable

Sommaire

5.1 Introduction

Dans le chapitre précédent, nous avons fourni une spécification de la gestion de groupe partitionnable à base de consensus abandonnables. Dans ce chapitre, nous mettons en œuvre une implantation de notre approche. Le consensus abandonnable repose sur un détecteur ultime des α participants d'une partition $\Diamond\mathcal{PPD}$ et sur un registre ultime par partition $\Diamond\mathcal{RPP}$.

Ce chapitre est organisé comme suit. D'abord, dans la section 5.2, nous présentons une implantation du module $\Diamond\mathcal{PPD}$. Puis, dans la section 5.3, nous implantons le module de retransmission des messages qui est utilisé par le module $\Diamond\mathcal{RPP}$ et la gestion de groupe partitionnable. Dans la section 5.4, nous présentons une implantation du module $\Diamond\mathcal{RPP}$. Dans la section 5.6, nous présentons une implantation de la gestion de groupe partitionnable. Enfin, dans la section 5.7, nous parcourons les travaux de la littérature liés au détecteur de participants dans une partition, à la notion de registre dans les systèmes dynamiques tels que les MANETs et à la gestion de groupe dans les MANETs, avant de conclure et de présenter quelques perspectives dans la section 6.5.

5.2 Détecteur ultime des α participants d'une partition

Dans cette section, nous présentons une implantation du détecteur ultime des α participants. Intuitivement, chaque processus p diffuse périodiquement un message de battement de cœur (en anglais, *heartbeat* [23]) afin de détecter les processus stables qui se trouvent dans la même partition. De manière similaire au détecteur de défaillances [Chandra and Toueg, 1996], le détecteur ultime des α participants d'une partition fournit des informations non fiables quand la condition de stabilité n'est pas satisfaite. Par exemple, si la valeur de la période de détection est trop faible, un processus risque d'être détecté comme non stable alors qu'il l'est. Le choix de la valeur de la période de détection est un paramètre important et doit dépendre des scénarios applicatifs. Notre modèle présenté dans le chapitre 3 suppose qu'il existe un instant après lequel la partition devient stable, c'est-à-dire qu'il existe une période de temps suffisamment longue durant laquelle les processus dans une même partition stable peuvent communiquer entre eux à travers des chemins SADDM. Cette durée est cependant inconnue des processus. Le processus p peut alors utiliser le critère de stabilité basé sur le comptage des battements de cœur afin de sélectionner les processus qui sont les plus stables durant une période donnée, c'est-à-dire les nœuds qui peuvent être considérés comme faisant partie d'une éventuelle partition stable.

5.2.1 Implantation

L'algorithme 3 implante le module $\Diamond\mathcal{PPD}$ pour le processus p. L'algorithme utilise deux types de messages : HEARTBEAT et ALPHASET. Nous rappelons que chaque message m est dénoté par $m \stackrel{\text{déf}}{=} (originalSender, \langle \text{TYPE} \mid attribute_1, attribute_2 \ldots \rangle, destinatorsSet)$. TYPE est le type du message. $originalSender$ correspond à l'expéditeur originel de m et $destinatorsSet$

23. Le concept de message de battement de cœur est repris de [Aguilera et al., 2000].

est l'ensemble des destinataires. L'ensemble *destinatorsSet* est remplacé par « * » quand les destinataires du message ne sont pas précisés. Cela signifie que n'importe quel processus dans le système est un destinataire de m. Un message de type HEARTBEAT ou de type ALPHASET contient un chemin, c'est-à-dire une séquence de processus qui ont déjà vu (reçu) le message. Dans l'algorithme, le symbole « $\circ$ » est utilisé comme opérateur de concaténation de deux chemins. Soient $path_1 = (p, q, r)$ et $path_2 = (s, t, u)$ deux chemins, $path_1 \circ path_2 \overset{\text{déf}}{=} (pqr) \circ (stu) = (pqrstu)$. En outre, par un léger abus de notation, les variables *path* utilisées dans les messages HEARTBEAT contiennent des n-uplets $(processus, \alpha)$, au lieu de simples singletons $(processus)$.

L'algorithme est basé sur l'échange périodique de messages HEARTBEAT afin d'identifier les processus qui sont actuellement mutuellement atteignables. Chaque processus utilise son critère de stabilité local pour déterminer les processus qui sont les plus stables, c'est-à-dire ceux qui ont échangé les plus grands nombres de battements de cœur.

Les messages ALPHASET sont diffusés par les processus qui croient être leader de leur partition. Ultimement, seul le leader ultime avec la plus grande valeur de α continue à diffuser originellement des messages ALPHASET. Nous rappelons que l'expression « diffuser originellement un message m » signifie que le premier appel à la primitive $broadcast_{nbg}$ est effectué par p, et donc $originalSender = p$.

Les variables locales de l'algorithme 3 sont initialisées dans la phase **Init** (lignes 2–14) :

- α_p est le nombre de processus stables minimum requis par l'application. Chaque processus $q \neq p$ peut choisir une valeur α_q différente de celle de α_p, c'est-à-dire $\alpha_q \neq \alpha_p$;
- THRESHOLD_p est la valeur seuil du critère de stabilité décrit dans la section 2.2.7. Le critère de stabilité sert à sélectionner les processus stables parmi les processus actuellement mutuellement atteignables. Le critère proposé est basé sur le comptage des battements de cœur. hb_p^q représente le compteur de battements de cœur que p utilise pour compter les battements de cœur de q ;
- αSet_p est l'ensemble des processus qui sont considérés comme stables par p tel que $\forall q \in \alpha Set_p : hb_p^q \geqslant \text{THRESHOLD}_p \wedge \alpha_p \geqslant \alpha_q$. Il contient seulement p à l'initialisation ;
- MAXHB_p est la valeur maximale qu'un compteur de battements de cœur peut atteindre. Ainsi, un compteur de battements de cœur n'augmente pas indéfiniment et la durée de détection de la disparition d'un processus n'est pas proportionnelle à la durée de présence du processus dans la partition. Par convention, p est considéré recevoir ses propres messages de battements de cœur ;
- ${}_{part}\mathsf{timer}$ et ${}_{proc}\mathsf{timer}_p^q$ sont des temporisations qui délimitent deux types de périodes. ${}_{part}\mathsf{timer}$ est utilisée pour vérifier la condition de stabilité : à la fin d'une période d'une durée de ${}_{part}timeout$ secondes, le processus p vérifie si tous les processus dans αSet_p peuvent être encore considérés comme stables et vérifie aussi la condition de stabilité $|\alpha Set_p| \geqslant \alpha_p$ (ligne 24). Durant une période de ${}_{part}timeout$ secondes, ${}_{proc}\mathsf{timer}_p^q$ délimite une période d'une durée de ${}_{proc}timeout_p^q$ secondes. Ce second type de temporisation est utilisé pour compter les battements de cœurs, estimer si p et q sont mutuellement atteignables, et estimer aussi si q est stable (lignes 47–58) ;

- $mreachable_p$ est un ensemble de n-uplets (q, α_q), où q est un processus que p croît être dans sa partition, c'est-à-dire que $p \leftrightharpoons q$;
- $previous_p$ est la valeur précédente de l'ensemble $mreachable_p$;
- $tentative_p \supseteq \alpha Set_p$ est un ensemble de n-uplets $(process, \alpha, heartbeat\_nb)$ contenant les processus estimés par p comme les plus stables parmi les processus mutuellement atteignables, c'est-à-dire que ces nœuds peuvent être considérés comme faisant partie d'une éventuelle partition stable. p sélectionne seulement les processus qui possèdent un nombre de battements de cœur dépassant le seuil THRESHOLD_p. αSet_p est un sous-ensemble des processus de $tentative_p$ tel que $\forall q \in \alpha Set_p \; \exists l \in \alpha Set_p : \alpha_q \leqslant \alpha_l \wedge h_p^q \geqslant \text{THRESHOLD}_p$. Par convention, lorsque nous manipulons des n-uplets, nous utilisons le caractère « * » pour indiquer que la valeur correspondante est quelconque. Par exemple, à la ligne 66, l'écriture $(q, *, 1)$ correspond au triplet pour le processus q, avec $hb_p^q = 1$ et une valeur quelconque pour α.

L'objectif de l'algorithme est triple : *1)* tous les processus dans αSet_p possèdent ultimement le même ensemble αSet_p, *2)* la valeur de αSet_p est ultimement égale à celle de αSet_l avec l étant le leader parmi les processus de αSet_p, et ainsi *3)* tous les processus dans αSet_p élisent ultimement le même leader $l \in \alpha Set_p$. L'ensemble αSet_p est utilisé comme valeur de sortie de l'algorithme (Ligne 76). Un processus q participe à la construction de l'ensemble αSet_p si $|\alpha Set_p| \geqslant \alpha_q$. L'idée est qu'ultimement, seul le processus stable avec la plus grande valeur de α devient le leader ultime et impose sa valeur de αSet. Ainsi, le processus stable ayant la contrainte la plus forte (la valeur de α la plus élevée) impose sa contrainte aux autres processus de sa partition.

Après avoir initialisé ses variables, l'algorithme du processus p est divisé en 5 tâches. La tâche 1 sert à diffuser périodiquement les messages HEARTBEAT. La tâche 2 est utilisée pour vérifier la condition de stabilité à la fin d'une période de ${}_{part}timeout$ secondes. Les tâche 3 et 4 traitent respectivement les messages ALPHASET et HEARTBEAT reçus. Enfin, la tâche 5 sert à déterminer le nombre de battements de cœur de chaque processus q durant la période de vérification de la condition de stabilité. Nous détaillons maintenant ces cinq tâches.

Dans la tâche 1, le processus p diffuse périodiquement un message HEARTBEAT contenant le chemin d'amorçage (p, α_p), c'est-à-dire $(p,\langle \text{HEARTBEAT} \mid (p, \alpha_p)\rangle,*)$, pour annoncer qu'il est correct et présent.

Dans la tâche 2, à l'expiration de ${}_{part}\mathsf{timer}$, p vérifie l'ensemble des processus que p croît être stables dans sa partition. Si un ou plusieurs processus dans αSet_p deviennent non stables ou si un ou plusieurs processus sont devenus stables ($\alpha Set_p \not\subseteq tentative_p$, ligne 24), ou si la condition de stabilité n'est pas satisfaite ($|\alpha Set_p| < \alpha_p$, ligne 24), alors p recompose son αSet_p (ligne 25). Si la condition de stabilité est potentiellement satisfaite ($|\alpha Set_p| \geqslant \alpha_p$, ligne 26) et que p croit être le leader (ligne 27), alors p tente de convaincre les autres processus dans son ensemble αSet_p de se mettre d'accord sur sa valeur en diffusant un message $(p,\langle \text{ALPHASET} \mid (p), \alpha Set_p\rangle,*)$ (ligne 28). Sinon, p incrémente la période de détection ${}_{part}timeout$. Cela signifie que si p est stable, soit la durée de stabilisation locale à la partition n'est pas atteinte, soit cette durée est atteinte mais la valeur de la période de vérification de la condition de stabilité ${}_{part}timeout$ n'est pas assez grande pour que p détecte chaque processus stable q comme tel à la fin de cette période. Finalement, p

Algorithm 3 Implantation pour le processus p du module $\Diamond\mathcal{PPD}$: phase initiale (1/3)

```
1  init():
2  Begin
3      α_p ← n with n ⩾ 1;                          {Minimum number of stable processes required by the application}
4      αSet_p ← {(p, α_p)};                          {Stable processes with their value of α}
5      THRESHOLD_p ← c;                              {Minimal value for being stable according to the stability criterion}
6      MAXHB_p ← hb with hb ⩾ THRESHOLD_p;            {Maximum value that a heartbeat counter can have}
7      part timeout ← t ⩾ 1;
8      set part timer to part timeout;               {Timer used for checking the stability condition}
9      proc timeout ← {}                             {Sets of pairs (process, timeout)}
10     proc timer ← {};                              {Sets of pairs (process, timer)}
11     mreachable_p ← {(p, α_p)};                    {Mutually reachable processes with their respective value of α}
12     previous_p ← {(p, α_p)};                      {Previous value of mreachable_p}
13     tentative_p ← {(p, α_p, MAXHB_p)};            {Set of tuples (process, α, heartbeat_nb) for constituting αSet_p}
14 End
15
16 Tasks T1–T3 and T4–T5 are described at pages 87 and 88, respectively.
```

se prépare pour une nouvelle exécution de la tâche 2 (lignes 31–32).

Dans la tâche 3, à la réception du message $(q,\langle\text{ALPHASET} \mid path, \alpha Set_p\rangle,*)$ (ligne 35), p vérifie si le premier élément du chemin $path$ est p. Alors, p sait que son message $(p,\langle\text{ALPHASET} \mid (p), \alpha Set_p\rangle,*)$ a été transporté à travers un cycle. Si c'est le cas, p ignore le message reçu car p est l'expéditeur originel du message. Dans le cas où le premier élément du chemin $path$ n'est pas p (ligne 37), p vérifie s'il n'apparaît pas dans $path$ (ligne 38). Ensuite, p vérifie si p et q sont mutuellement atteignables, et si q est le leader de p (ligne 39). Si c'est le cas, p remplace la valeur de son αSet_p par celle de αSet_q (ligne 40). Par conséquent, p et q croient qu'il existe au moins $\alpha_q \geqslant \alpha_p$ processus stables dans leur partition. Dans le cas où p apparaît déjà dans $path$, p ne vérifie pas la valeur de αSet_q car p avait déjà reçu et traité le message reçu. Enfin, si le premier élément de $path$ n'est pas p et si p apparaît au plus une fois dans $path$, alors p concatène p à $path$ et diffuse le message $(q,\langle\text{ALPHASET} \mid path \circ p, \alpha Set_p\rangle,*)$ afin de permettre d'acheminer le message aux autres processus destinataires (ligne 42). Observons que, comme décrit dans [Aguilera et al., 1999], p doit retransmettre le message même s'il apparaît déjà une fois dans $path$ puisqu'il peut exister un cycle de q à r où p appartient à la fois au chemin de q à r et au chemin de r à q.

Dans la tâche 4, à la réception d'un message $(r,\langle\text{HEARTBEAT} \mid path\rangle,*)$, si le premier élément du chemin $path$ est (p, α_p), alors p sait qu'un de ses messages $(p,\langle\text{HEARTBEAT} \mid (p, \alpha_p)\rangle,*)$ a été transporté à travers un cycle, c'est-à-dire que chaque nœud q dans les n-uplets qui apparaissent après (p, α_p) dans $path$ sont mutuellement atteignables par p (lignes 47). Quand p voit q pour la première fois (ligne 48), p crée dynamiquement une temporisation $_{proc}\mathsf{timer}_p^q$ associée à la

période ${}_{proc}timeout_p^q$ et démarre la temporisation (Line 58). La temporisation ${}_{proc}\mathsf{timer}_p^q$ est redémarrée à chaque fois qu'un message de battements de cœur est transporté à travers un cycle (ligne 58). Si q était précédemment atteignable par p (ligne 53) et s'il ne participait pas déjà à la construction de αSet_p ($(q, \alpha_q) \notin tentative_p$, ligne 54), alors p commence à considérer q comme un processus potentiellement stable et ajoute q à son ensemble $tentative_p$ avec un compteur de battements de cœur initialisé à 1 (ligne 55). Si q a déjà participé à la construction de αSet_p ($(q, \alpha_q, hb_p^q) \in tentative_p$, ligne 56), p incrémente le compteur de battements de cœur de q car q est « plus stable » (ligne 57). Si le chemin $path$ ne commence pas par le premier élément (p, α_p) et si p n'apparaît pas dans $path$ ou apparaît juste une fois, alors p concatène (p, α_p) à $path$ et diffuse à ses voisins un battement de cœur avec le nouveau chemin (ligne 61).

Dans la tâche 5, à l'expiration de la temporisation ${}_{proc}\mathsf{timer}_p^q$, p décrémente le compteur de battements de cœur de q car q devient « moins stable » (ligne 69). Quand le compteur de battements de cœur de q atteint 0 (ligne 66), q est retiré de $tentative_p$ pour signifier que q n'est plus considéré comme stable par p et qu'il ne doit plus participer à la construction de αSet_p (ligne 66). L'expiration de ${}_{proc}\mathsf{timer}_p^q$ traduit le fait que la valeur de la période ${}_{proc}timeout_p^q$ n'est pas assez grande pour qu'un message de battement de cœur soit transporté à travers un cycle partant de p et incluant q. C'est pourquoi ${}_{proc}timeout_p^q$ est incrémenté. Cependant, la valeur de la temporisation ${}_{proc}timeout_p^q$ ne doit pas dépasser celle de ${}_{part}timeout_p$ (ligne 70).

Enfin, un client applicatif appelle la procédure $\mathcal{PPD}$-$participants()$ pour invoquer son module de détection des participants dans sa partition (lignes 74–77). Il obtient alors l'identifiant du leader l de la partition ainsi que l'ensemble des processus stables αSet_p, avec $\alpha Set_p = \alpha Set_l$.

5.2.2 Preuve de correction

Nous montrons que l'algorithme 3 implante le détecteur ultime des α participants d'une partition $\Diamond\mathcal{PPD}$ spécifié dans la section 4.3.4 car il respecte les propriétés suivantes :

- $\mathcal{PPD}$-Stabilité de αSet : il existe un instant après lequel deux processus stables p et q d'un ensemble αSet de processus stables possèdent la même connaissance du même ensemble αSet ;
- $\mathcal{PPD}$-Accord sur l'élection du leader : il existe un instant après lequel deux processus stables p et q d'un ensemble αSet élisent le même processus stable dans αSet comme le leader.

Lemme 1. *Soit p_1 un processus stable et p_n un processus dans $\Diamond PART_{p_1}$ tel qu'il existe un chemin SADDM de p_1 à p_n. Ultimement, un des β^{n-1} messages (p_1,⟨HEARTBEAT | (p_1, α_{p_1})⟩,$*$) diffusés originellement par p_1 arrive à p_n au plus en $\beta^{n-1}\eta + (n-1)\delta$ secondes.*

Démonstration. Soit p_1 un processus stable et p_n un processus dans $\Diamond PART_{p_1}$ tel qu'il existe un chemin SADDM de p_1 à p_n. Soit $\Sigma = saddm\_path(p_1 p_2 \dots p_n)$. Pour simplifier la présentation de la preuve sans perdre en généralité, seule la partie incluant le type et les attributs du message est considérée. De plus, le chemin inclus dans le message de type HEARTBEAT est considéré seulement comme une séquence de processus. Autrement dit, nous ne considérons pas la valeur de α associée à chaque processus. Par définition d'un chemin SADDM, chaque processus p_i, pour

Algorithm 3 Implantation pour le processus p du module $\Diamond\mathcal{PPD}$: suite (2/3)

```
17  Task T1: every η seconds                                   {Broadcast heartbeats}
18  Begin
19      broadcast_nbg(p,⟨HEARTBEAT | (p, α_p)⟩,∗);
20  End
21
22  Task T2: upon expiration of _part timer                    {Check the stability of an αSet}
23  Begin
24      If (αSet_p ⊈ tentative_p ∨ |αSet_p| < α_p) then
25          αSet_p ← {(q, α_q)|(q, α_q, hb_p^q) ∈ tentative_p ∧ hb_p^q ⩾ THRESHOLD_p};
26      If |αSet_p| ⩾ α_p then
27          If p = r : (r, α_r) ∈ αSet_p ∧ [∀(s, α_s) ∈ αSet_p : α_r > α_s ∨ (α_r = α_s ∧ r > s)] then
28              broadcast_nbg(p,⟨ALPHASET | (p), αSet_p⟩,∗);
29      Else
30          _part timeout ← _part timeout + 1;   {Stability condition not satisfied, increase detection period}
31      previous_p ← mreachable_p;
32      set _part timer to _part timeout;
33  End
34
35  Task T3: upon reception of (q,⟨ALPHASET | path, αSet_q⟩,∗)   {Verify the leader's αSet}
36  Begin
37      If (first process in path is not p) then
38          If p does not appear in path then
39              If αSet_p ⊆ αSet_q then
40                  αSet_p ← αSet_q;
41          If p appears at most once in path then
42              broadcast_nbg(q,⟨ALPHASET | path ∘ (p)⟩,∗);
43  End
```

Algorithm 3 Implantation pour le processus p du module $\Diamond\mathcal{PPD}$: suite (3/3)

```
44  Task T4: upon reception of (r,⟨HEARTBEAT | path⟩,∗)        {Detect mutually reachable processes}
45  Begin
46      If first tuple in path is (p, α_p) then
47          For all (q, α_q) : (q, α_q) appears after the first tuple in path ∧ q ≠ p do
48              If (q, α_q) ∉ mreachable_p then
49                  mreachable_p ← mreachable_p ∪ {(q, α_q)};
50                  _proc timer ← _proc timer ∪ {(q,_proc timer_p^q)}; {Dynamic creation of timer for new process q}
51                  _proc timeout_p^q ← 1; _proc timeout ← _proc timeout ∪ {(q,_proc timeout_p^q)};
52              Else
53                  If (q, α_q) ∈ previous_p then
54                      If (q, α_q, hb_p^q) ∉ tentative_p then
55                          tentative_p ← tentative_p ∪ {(q, α_q, 1)};
56                      Else
57                          tentative_p ← tentative_p \ {(q, ∗, ∗)} ∪ {(q, α_q, max(hb_p^q+1, MAXHB_p))};
58              set _proc timer_p^q to _proc timeout_p^q;        {Reset timer when receiving HEARTBEAT}
59      Else
60          If (p, α_p) appears at most once in path then
61              broadcast_nbg(r,⟨HEARTBEAT | path ∘ (p, α_p)⟩,∗);
62  End
63
64  Task T5: upon expiration of _proc timer_p^q                {Compute heartbeats per period}
65  Begin
66      If hb_q^p = 1 : (q, ∗, 1) ∈ tentative_p then
67          tentative_p ← tentative_p \ {(q, ∗, 1)};
68      Else
69          tentative_p ← tentative_p \ {(q, ∗, ∗)} ∪ {(q, ∗, hb_p^q−1)};
70      _proc timeout_p^q ← min(_proc timeout_p^q + 1,_part timeout);
71      _proc timeout ← _proc timeout \ {(q, ∗)} ∪ {(q,_proc timeout_p^q)};
72  End
73
74  Procedure ΠΠΔ-participants()
75  Begin
76      generate ΠΠΔ-return(l, αSet_p) : (l, α_l) ∈ αSet_p ∧ [∀(r, α_r) ∈ αSet_p : α_l > α_r ∨ (α_l = α_r ∧ l > r)];
77  End
```

$i \in [1,n]$, apparaît au plus une fois dans Σ. Pour tout $j \in [1,n]$, soit $P_j = saddm\_path(p_i)_{i\in[1,j]}$. Pour démontrer le lemme, nous montrons par induction que $\forall j \in [1,n-1]$, au moins un des β^{j-1} messages $\langle \text{HEARTBEAT} \mid P_{j-1}\rangle$ originellement diffusés par p_1 arrive à p_j au plus en $\beta^{j-1}\eta+(j-1)\delta$ secondes.

Pour le cas de base ($j = 1$), dans la tâche T1, p_1 diffuse périodiquement le message $\langle \text{HEARTBEAT} \mid P_1\rangle$ tous les η secondes à tous ses voisins, incluant p_2. Comme le chemin (p_1p_2) est un chemin SADDM, au moins un des β messages diffusés par p_1 est reçu par p_2 au plus en $\beta\eta+\delta$ secondes. Ceci montre que le lemme est vrai pour le cas de base. Pour le cas d'induction, soit $j \leqslant n-1$ et supposons par induction qu'au moins un des β^{j-1} messages $\langle \text{HEARTBEAT} \mid P_{j-1}\rangle$ originellement diffusés par p_1 est reçu par p_j au plus en $\beta^{j-1}\eta + (j-1)\delta$ secondes. Puisque le chemin (p_jp_{j+1}) est un chemin SADDM, p_{j+1} reçoit au moins un de ces $\beta\beta^{j-1} = \beta^j$ messages $\langle \text{HEARTBEAT} \mid P_1\rangle$ au plus en $\beta^j\eta+j\delta$ secondes. De plus, p_{j+2} apparaît au plus une fois dans P_{j+1} et p_{j+2} est un voisin de p_{j+1}. Donc, à chaque fois que p_{j+1} reçoit $\langle \text{HEARTBEAT} \mid P_j\rangle$, il rediffuse $\langle \text{HEARTBEAT} \mid P_{j+1}\rangle$ à p_{j+2} en s'ajoutant au chemin P_j (ligne 61). En outre, puisque $(p_{j+1}p_{j+2})$ est un chemin SADDM, p_{j+2} reçoit l'un des $\beta\beta^j = \beta^{j+1}$ messages $\langle \text{HEARTBEAT} \mid P_{j+1}\rangle$ originellement diffusés par p_1 au plus en $\beta^{j+1}\eta + (j+1)\delta$ secondes. Ceci montre le cas d'induction. Par conséquent, nous concluons qu'ultimement un des β^{n-1} messages $\langle \text{HEARTBEAT} \mid P_1\rangle$ diffusés par p_1 arrive à p_n au plus en $\beta^{n-1}\eta + (n-1)\delta$ secondes. □

Lemme 2. *Soit p un processus stable. Il existe un instant après lequel $hb_p^q = \text{MAXHB}_p$ et $hb_q^p = \text{MAXHB}_q$ restent vrais, $\forall q \in \Diamond PART_p$.*

Démonstration. Soient p un processus stable et q un processus dans $\Diamond PART_p$. Soient $\Sigma 1 = saddm\_path(p_i)_{i\in[1,k]}$ un chemin SADDM de p à q et $\Sigma 2 = saddm\_path(p_i)_{i\in[k,n]}$ un chemin SADMM de q à p. Nous considérons le chemin $\Sigma = saddm\_path(p_i)_{i\in[1,n]}$ qui correspond à la concaténation de $\Sigma 1$ avec $\Sigma 2$, c'est-à-dire $\Sigma = \Sigma 1 \circ \Sigma 2$.

Selon la définition d'un chemin SADDM, $\forall i \in [1,k]$ et $\forall i \in [k,n]$, chaque processus p_i apparaît au plus une fois dans $\Sigma 1$ et $\Sigma 2$, respectivement, et au plus deux fois dans Σ. Par construction, $p_1 = p_n = p$, et $p_k = q$. Pour tout $j \in [1,n]$, soit $P_j = saddm\_path(p_i)_{i\in[1,j]}$. La preuve de $hb_q^p = \text{MAXHB}_q$ peut être obtenue de la même façon en considérant le chemin SADDM Σ' qui est égal au chemin $\Sigma 1'$ concaténé avec $\Sigma 2'$, où $\Sigma 1' = saddm\_path(p_i)_{i\in[k,n]}$ et $\Sigma 2' = saddm\_path(p_i)_{i\in[n,k]}$, de q à p. Puisque les deux preuves sont similaires, seule l'une d'entre elles ($hb_p^q = \text{MAXHB}_p$) est présentée.

La preuve de $hb_p^q = \text{MAXHB}_p$ est présentée comme suit. Selon le lemme 1, au moins un des β^{n-1} messages $\langle \text{HEARTBEAT} \mid P_{n-1}\rangle$ originellement diffusés par p_1 arrive ultimement à p_n au plus en $\beta^{n-1}\eta+(n-1)\delta$ secondes. Quand p_n reçoit un message HEARTBEAT, $\forall i \in [2,n-1]$, ${}_{proc}\mathsf{timer}_{p_n}^{p_i}$ est réinitialisé à ${}_{proc}timeout_{p_n}^{p_i}$ (ligne 58) tel que ${}_{proc}\mathsf{timer}_{p_n}^{p_i}$ n'expire pas. Sinon, le compteur ${}_{proc}timeout_{p_n}^{p_i}$ est incrémenté ultimement jusqu'à atteindre la valeur $\beta^{n-1}\eta + (n-1)\delta$. Ainsi, p reçoit ultimement pour toujours les messages de battements de cœur $\langle \text{HEARTBEAT} \mid P_{n-1}\rangle$ avec p comme premier élément du chemin P_{n-1} et tel que ${}_{proc}\mathsf{timer}_p^q$ n'expire pas. Puisque q apparaît après p dans P_{n-1}, $(q,*) \in previous_p$ (ligne 53) et $(q,*,*) \in tentative_p$ (lignes 54–57) (rappelons que par construction, $p_1 = p_n$ et $p_k = q$), p incrémente le compteur hb_p^q jusqu'à ce qu'il atteigne la valeur MAXHB_p. □

Lemme 3. *Soit p un processus stable. Il existe un instant après lequel si $q \in \alpha Set_p$ de façon permanente alors $q \in \Diamond PART_p$.*

Démonstration. Soit p un processus stable. Nous montrons par contraposée que pour chaque processus $q \notin \Diamond PART_p$, il n'existe pas d'instant après lequel $q \in \alpha Set_p$ de façon permanente. Selon le lemme 2, pour chaque processus $q \notin \Diamond PART_p$, il n'existe pas d'instant après lequel $hb_p^q = \text{MAXHB}_p$ et $hb_q^p = \text{MAXHB}_q$ restent vrais. Donc, les lignes 66 et 69 de l'algorithme 3 ne sont plus exécutées. De plus, chaque processus $q \notin \Diamond PART_p$ qui est ajouté à l'ensemble $tentative_p$ peut être retiré de cet ensemble par la suite. αSet_p est un sous-ensemble de processus dans $tentative_p$ (ligne 25). Ainsi, il n'existe pas d'instant après lequel $q \in \alpha Set_p$ de façon permanente. □

Lemme 4. *Soient p un processus stable et q un processus dans $\Diamond PART_p$. Il existe un instant après lequel $\alpha Set_p \subseteq tentative_p$ reste vrai.*

Démonstration. Soient p un processus stable et q un processus dans $\Diamond PART_p$. Selon le lemme 3, le fait que q appartienne à αSet_p est traduit par $q \in \Diamond PART_p$. Considérons qu'il existe un instant après lequel $hb_p^q = \text{MAXHB}_p$. Selon le lemme 2, q est alors ajouté à l'ensemble $tentative_p$ et q n'est jamais retiré de cet ensemble par la suite. Puisque l'ensemble αSet_p correspond à un sous-ensemble de $tentative_p$ (lignes 24–25), $\alpha Set_p \subseteq tentative_p$ reste vrai ultimement. □

Lemme 5. *Soit p un processus stable tel que $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q), \forall q \in \Diamond PART_p$. Il existe un instant après lequel tous les messages* ALPHASET *diffusés contiennent αSet_p et p est l'expéditeur originel de ces messages.*

Démonstration. Soit p un processus stable tel que $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q), \forall q \in \Diamond PART_p$. Par définition de la condition de stabilité, c'est-à-dire $|\alpha Set_p| \geqslant \alpha$, et par le lemme 4, la condition de la ligne 26 est ultimement toujours vraie. Ainsi, ultimement, p est le seul processus expéditeur qui déclenche périodiquement la diffusion des messages $(p,\langle \text{ALPHASET} \mid (p)\rangle,*)$. Nous pouvons montrer par contradiction qu'ultimement aucun processus $(q \neq p) \in \alpha Set_p$ ne génère de message ALPHASET. Supposons qu'il existe un processus $q \in \alpha Set_p$ tel que $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q)$, et qu'il n'existe pas d'instant après lequel q arrête de diffuser originellement des messages ALPHASET. Dans la tâche T2, q diffuse les messages ALPHASET quand les conditions dans les lignes 24, 26 et 27 sont vraies. Ceci implique que $\alpha Set_q \not\subseteq tentative_q$ (avant que q mette à jour la valeur de αSet_q à la ligne 25). Puisque $p \in tentative_q$ et $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q)$, la condition $\alpha Set_q \not\subseteq tentative_q$ ne peut être vraie que si $hb_q^p < \text{THRESHOLD}_q \leqslant \text{MAXHB}_q$, ce qui conduit à une contradiction avec le lemme 2. □

Lemme 6. *Soit p un processus stable tel que $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q), \forall q \in \Diamond PART_p$. Il existe un instant après lequel $\alpha Set_q = \alpha Set_p$ reste vrai.*

Démonstration. Soit p un processus stable tel que $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q), \forall q \in \Diamond PART_p$. Selon le lemme 5, il existe un instant après lequel p est le seul processus expéditeur qui diffuse originellement les messages ALPHASET. Dans la tâche T3, à chaque fois que q reçoit un message $(p,\langle \text{ALPHASET} \mid path\rangle,*)$, q adopte la valeur de αSet_p pour sa variable locale αSet_q car $\alpha Set_q \subseteq$

αSet_p (selon le lemme 2 et par le fait que $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q), \forall q \in \Diamond PART_p$). De plus, d'après le lemme 4, $\alpha Set_q \subseteq tentative_q$ est ultimement toujours vrai. En outre, q garde ultimement son ensemble αSet_q inchangé. Ainsi, il existe un instant après lequel la condition $\alpha Set_q = \alpha Set_p$ reste vraie. □

Lemme 7. *Soit p un processus stable et q un processus dans $\Diamond PART_p$. Il existe un instant après lequel le module $\Diamond\mathcal{PPD}_q$ du processus $q \in \Diamond PART_p$ fournit toujours comme valeur de sortie $(l, \alpha Set_l)$ avec $(l, \alpha_l) \in \alpha Set_p \subseteq \Diamond PART_p \wedge [\forall (r, \alpha_r) \in \alpha Set_p : \alpha_l > \alpha_r \vee (\alpha_l = \alpha_r \wedge l > r)]$.*

Démonstration. Soit p un processus stable. Selon le lemme 6, il existe un instant après lequel $\alpha Set_q = \alpha Set_p$ reste vrai. Puisque la fonction d'élection du leader est la même pour tous les processus, il existe un instant après lequel $\Diamond\mathcal{PPD}_q$ du processus $q \in \alpha Set_p$ fournit $(l, \alpha Set_l)$ tel que $(l, \alpha_l) \in \alpha Set_p \subseteq \Diamond PART_p \wedge [\forall (r, \alpha_r) \in \alpha Set_p : \alpha_l > \alpha_r \vee (\alpha_l = \alpha_r \wedge l > r)]$ (ligne 76). □

Théorème 1. *$\Diamond\mathcal{PPD}$ satisfait les propriétés $\mathcal{PPD}$-Stabilité de αSet et $\mathcal{PPD}$-Accord sur l'élection du leader.*

Démonstration. Considérons un processus stable p tel que $\alpha_p > \alpha_q \vee (\alpha_p = \alpha_q \wedge p > q), \forall q \in \alpha Set_p \subseteq \Diamond PART_p$. Selon le lemme 6, il existe un instant après lequel $\alpha Set_q = \alpha Set_p$ reste vrai. Ainsi, tous les processus dans αSet_p possèdent ultimement le même ensemble $\alpha Set = \alpha Set_p$. Ceci satisfait la propriété $\mathcal{PPD}$-Stabilité de αSet. Selon le lemme 7, pour chaque processus stable q dans αSet, le module $\Diamond\mathcal{PPD}_q$ fournit le même leader l. Ceci satisfait la propriété $\mathcal{PPD}$-Accord sur l'élection du leader. □

5.3 Module de retransmission

Cette section présente une implantation du module de retransmission pour les réseaux mobiles spontanés spécifié dans la section 4.3.3. Soit p un processus stable et *dest* un ensemble de processus stables inclus dans $\Diamond PART_p$. Si p diffuse un message m β^n fois et si *dest* est l'ensemble des processus destinataires de m, alors le message m originellement diffusé par p arrive à chaque processus $q \in dest$ au plus en $\beta^n\eta + n\delta$ secondes, avec η la période de temps maximale qui sépare deux diffusions consécutives du message m et n la longueur du chemin SADDM le plus long entre p et tout processus $r \in dest$.

Pour tolérer les pertes de messages acheminés à travers les chemins SADDM, chaque message diffusé doit être retransmis périodiquement et la valeur de la période ne doit pas être trop élevée de telle sorte que les rediffusions ne soient pas trop dispersées dans le temps. L'expéditeur originel du message continue à rediffuser son message tant qu'il n'a pas reçu un acquittement de tous les destinataires du message. Les processus qui ne sont pas parmi les destinataires du message participent à la rediffusion du message en jouant le rôle de « routeur ». Dans la section 5.3.1, nous détaillons cette implantation. Ensuite, dans la section 5.3.2, nous fournissons la correction de l'implantation.

5.3.1 Implantation

L'algorithme 4 implante pour le processus p le module de retransmission des messages pour les MANETs. Tous les messages qui doivent être retransmis sont stockés dans la variable $xmessages$. Chaque message m est associé à un ensemble de destinataires $dest$ et à un booléen $delivered$. $delivered = true$ signifie que le module de retransmission a déjà livré le message m avec $receive$ au processus p. Sinon $delivered = false$. En précisant les destinataires de chaque message m, l'expéditeur du message m peut arrêter de rediffuser m lorsqu'il a reçu un acquittement de tous les destinataires associés à m. Ainsi, l'algorithme est dit « silencieux » [Aguilera et al., 1999, Aguilera et al., 2000]. La variable $rmessages$ est utilisée pour stocker l'ensemble des messages reçus avec $xreceive$ et permet de ne livrer m qu'une seule fois.

Dans la phase **Init**, après avoir initialisé toutes les variables, la tâche de retransmission périodique est démarrée.

L'ajout d'un message m à l'ensemble $xmessages$ est réalisé par la procédure de diffusion $xbroadcast$. Quant à la suppression d'un message m de l'ensemble $xmessages$, elle est réalisée par la tâche 1 qui traite la réception d'un message et par la tâche de retransmission.

Après avoir exécuté la procédure de diffusion d'un message m (ligne 9), p vérifie s'il voit le message m pour la première fois, c'est-à-dire $((m, *, *) \notin xmessages)$. Si c'est le cas, le module reçoit le message diffusé (ligne 13). p ajoute le chemin d'amorçage ($originalSender$) au message m [24]. p ajoute le n-uplet $(m, dest, false)$ à l'ensemble $xmessages$, où m est le message, $dest$ est l'ensemble des destinataires de m et $false$ signifie que le module de retransmission n'a pas encore livré le message m au processus p (ligne 15). Ensuite, p diffuse le message m à l'ensemble de ses voisins en exécutant la primitive $broadcast_{nbg}(m)$ (ligne 16). Au besoin, p reçoit son propre message diffusé : m est reçu avec $xreceive$ par le module qui utilise le module de retransmission pour diffuser et recevoir des messages (ligne 17).

Dans la tâche 1 (lignes 21–35), p reçoit un message m de la forme ($originalSender$,$\langle$TYPE | $path, \ldots\rangle$,$dest$). p vérifie s'il figure en première position dans $path$, c'est-à-dire que l'expéditeur originel de m est p. Dans ce cas, p sait que son message a été reçu par les processus qui apparaissent après p dans $path$. Bien évidemment, $(m, dest, *) \in xmessages$. Puis, p met à jour l'ensemble $dest$ des destinataires du message m. Lorsque l'ensemble $dest$ est vide, p retire m de $xmessages$ car tous les destinataires associés à m ont reçu m (ligne 25). Sinon, lorsque p n'apparaît pas dans $path$, p vérifie s'il est un destinataire de m et s'il n'a jamais livré m (ligne 27–28). Le cas échéant, p livre le message m reçu (ligne 32) après avoir ajouté m à $rmessages$ (ligne 30) et supprimé l'attribut $path$ de m (ligne 31). Comme dans l'algorithme 3, si le chemin $path$ ne commence pas par le processus p et si p n'apparaît pas dans $path$ ou apparaît juste une fois, p concatène p à $path$ et diffuse m avec le nouveau chemin à ses voisins.

Sans traitement particulier, p pourrait continuer à diffuser un message m infiniment lorsqu'un processus destinataire associé à m disparaît de la partition de p. Pour remédier à ce problème, le module de transmission invoque le module $\Diamond\mathcal{PPD}$ pour obtenir la valeur de αSet_p (lignes 39–

24. Rappelons que le contenu d'un message peut être modifié, mais pas son identifiant, et qu'un message est seulement identifié par son identifiant.

40). Ainsi, p ne rediffuse plus le message m si un des processus destinataires de m a disparu de la partition stable.

5.3.2 Preuve de correction

Nous montrons que l'algorithme 4 implante le module de retransmission en démontrant le théorème qui suit.

Théorème 2. *Soit p un processus stable et $dest$ un ensemble de processus stables inclus dans $\Diamond PART_p$. Si p diffuse un message m β^n fois et si $dest$ est l'ensemble des processus destinataires de m, alors le message m originellement diffusé par p arrive à chaque processus $q \in dest$ au plus en $\beta^n\eta + n\delta$ secondes, avec η la période de temps maximale qui sépare deux diffusions consécutives du message m et n la longueur du chemin SADDM le plus long entre p et un processus $r \in dest$.*

Démonstration. Soit p_1 un processus stable, p_n (avec $n > 1$) un processus dans $\Diamond PART_{p_1}$ tel qu'il existe un instant t_1 après lequel il existe un chemin SADDM de p_1 à p_n. Après t_1, p_1 diffuse un message m β^{n-1} fois à l'ensemble $dest$ des destinataires incluant p_n. Par la suite, considérons que n est la longueur du chemin SADDM le plus long entre p et un processus dans $dest$.

Considérons l'instant $t_2 \geqslant t_1$ auquel p_1 diffuse originellement le message $m = (p_1, \langle \text{TYPE} \mid (p_1), \ldots \rangle, \{dest : p_n \in dest\})$ à p_n. Il existe deux cas possibles : *1)* m a déjà été diffusé par p_1 avant l'instant t_2 ou *2)* m n'a jamais été diffusé par p_1 avant l'instant t_2. Dans les deux cas, le n-uplet $(m, \{p_n\}, *)$ est inclus dans l'ensemble $xmessages$ à la fin de l'appel de la procédure $xbroadcast$ (lignes 16). À partir de l'instant t_2, p_1 continue à diffuser périodiquement m tant qu'il n'a pas reçu un acquittement de p_n (lignes 22–25) et que p_n n'est pas exclus de l'ensemble αSet fourni par le module $\Diamond\mathcal{PPD}$ (ligne 39). Selon le lemme 3, si $q \in \Diamond PART_p$, alors q appartient à αSet_p fourni par $\Diamond\mathcal{PPD}$ de façon permanente. Par conséquent, la seule condition qui empêche p_1 de continuer à diffuser m est qu'il ait reçu un acquittement de p_n, c'est-à-dire p_n a reçu m de p_1. Nous montrons qu'il existe un instant t_3 auquel p_n reçoit m de p_1 et tel que $t_3 \leqslant t_2 + \beta^{n-1}\eta + (n-1)\delta$.

Après l'instant t_2, comme m est présent dans $xmessages$, p_1 génère à nouveau la diffusion de m avec $m = (p_1, \langle \text{TYPE} \mid (p_1), \ldots \rangle, \{dest' : p_n \in dest'\})$ ($dest'$ est possiblement $dest$: p_1 n'a pas reçu d'acquittement de p_n) au bout de la période de temps maximale η (ligne 42). Rappelons que dans notre modèle, nous considérons que la durée d'exécution des actions locales comme nulle. Ainsi, pour tout message m' dans $xmessages$, m' est diffusé par l'expéditeur originel au bout de la période de temps maximale η (tâche de retransmission, lignes 37–42).

En suivant l'approche du lemme 1, dans le pire des cas, lorsque p_1 diffuse m à p_n pour la β^{n-1^e} fois après t_2, le message m est garanti d'être reçu par p_n à l'instant t_3 tel que $t_3 \leqslant t_2 + \beta^{n-1}\eta + (n-1)\delta$. Par conséquent, à l'instant t_3, tous les processus dans $dest$ ont reçu le message m car pour tout q dans $dest$, nous avons $|saddm\_path(p_1 \ldots p_n)| \geqslant |saddm\_path(p_1 \ldots q)|$. □

Algorithm 4 Implantation pour le processus p du module de retransmission pour les réseaux mobiles spontanés

```
1  Init():
2  Begin
3      xmessages ← ∅;                                  {Set of tuples (message, destinators, delivered)}
4      rmessages ← ∅;                                  {Set of received messages with receive}
5      ppd ← create ◇𝒫𝒫𝒟;
6      start Task Retransmission;
7  End
8
9  Procedure xbroadcast(m) with m = (originalSender,⟨TYPE | ...⟩,dest);
10 Begin
11     If (m, ∗, ∗) ∉ xmessages then                   {See m for the first time}
12         If m ∈ dest then
13             xreceive(m);
14         m ← (originalSender,⟨TYPE | path = (originalSender), ...⟩,dest));   {Add bootstrap path}
15         xmessages ← xmessages ∪ (m, dest \ {originalSender}, false);
16     broadcast_nbg(m);
17     receive(m);
18 End
19
20 Task 1: upon receive(m) with m = (originalSender,⟨TYPE | path, ...⟩,dest)
21 Begin
22     If first process in path is p then                {originalSender is p}
23         For all q : q appears after the first process in path ∧ q ≠ p do
24             xmessages ← xmessages \ {(m, dest, ∗)} ∪ {(m, dest \ {q}, ∗)};
25         xmessages ← xmessages \ {(m, ∅, ∗)};          {Remove if destinators set is empty}
26     Else
27         If p does not appear in path then
28             If m ∉ rmessages then                     {First reception of m}
29                 rmessage ← rmessages ∪ {m};
30                 xmessages ← xmessages \ {(m, dests, ∗)};
31                 m ← (originalSender,⟨TYPE | ...⟩,dest);   {Remove attribute path before delivering}
32                 xreceive(m);
33         If p appears at most once in path then
34             broadcast_nbg(p,⟨TYPE | path ∘ (p), ...⟩,dest);
35 End
36
37 Task Retransmission: every η seconds
38 Begin
39     For all (m, dest, ∗) ∈ xmessages ∃q ∈ dest : q ∉ ppd.participants().αSet do
40         xmessages ← xmessages \ {(m, dest, ∗)};
41     For all (m, ∗, ∗) ∈ xmessages do
42         broadcast_nbg(m);
43 End
```

5.4 Registre ultime par partition

Cette section présente l'implantation du registre ultime par partition $\Diamond\mathcal{RPP}$. $\Diamond\mathcal{RPP}$ permet à un ensemble de processus αSet de choisir ultimement une même valeur. Dans la gestion de groupe partitionnable, chaque valeur val choisie correspondra à une vue successeur. Rappelons aussi que seuls les proposeurs peuvent proposer des valeurs. Avant que la condition de stabilité ne soit satisfaite, plusieurs processus peuvent croire qu'ils sont leader de αSet et tentent de convaincre les autres accepteurs de choisir leur valeur de manière persistante. Mais, aucun d'entre eux ne réussit. Le but de l'implantation de $\Diamond\mathcal{RPP}$ pour chaque processus p est de trouver le premier sous-ensemble des processus stables $S \subseteq \alpha Set_p$ (avec $\alpha_p \leqslant |S|$) qui choisissent tous une valeur proposée.

Dans la section 5.4.1, nous détaillons l'implantation du module $\Diamond\mathcal{RPP}$. Ensuite, dans la section 5.4.2, nous fournissons la preuve de correction de l'implantation.

5.4.1 Implantation

L'algorithme 5 implante pour le processus p le registre ultime par partition. Chaque proposition est associée à une valeur, qui est composée d'un ensemble de processus $vset \in 2^{\mathbb{P}}$ et d'un identifiant de vue $vid \in \mathbb{VID}$. $vset$ et vid correspondent respectivement aux membres et à l'identifiant de la vue successeur potentielle de la vue courante de p. Les estimations courantes des membres et de l'identifiant de la vue successeur de p sont stockées respectivement dans les variables $vset_p$ et vid_p. vid_p est aussi utilisé pour identifier de manière unique une proposition et permet au processus d'ignorer les « anciennes » propositions. La variable $lastProposal_p$ sert à stocker la dernière proposition de p ou la dernière proposition que p a acceptée. $lastProposal_p$ est stockée dans la mémoire stable locale de p. Ainsi, p peut récupérer sa dernière proposition lors du recouvrement. Toutes les variables sont initialisées dans la phase **Init** incluant la création d'une instance du module $\Diamond\mathcal{PPD}$.

Chaque appel de la procédure $\mathcal{RPP}$-*propose* commence par la mise à jour des valeurs $vset_p$ et vid_p (lignes 12 et 14). Pour cela, p vérifie s'il n'a pas déjà rencontré une proposition ayant un identifiant avec une valeur strictement plus grande que celle de l'identifiant de la proposition courante (c'est-à-dire, $lastProposal_p.id \geqslant vid.p$). Si c'est le cas, l'identifiant de la proposition courante vid prend la valeur $lastProposal_p.id + 1$. La valeur de $lastProposal_p$ est ensuite écrite dans la mémoire locale stable de p (tâche 1, lignes 21–24). Ainsi, le proposeur p choisit de façon strictement croissante les identifiants qu'il associe à ses propositions. Ensuite, p invoque la primitive *xbroadcast* pour diffuser sa proposition sous forme d'un message READ associé à l'identifiant vid_p. Cette diffusion correspond au début de la première des deux phases de la procédure : une phase de lecture (lignes 28–50) puis une phase d'écriture (lignes 51–72). Nous décrivons maintenant ces deux phases :

– phase de lecture : le but de la phase de lecture est de préparer l'ensemble de processus $vset_p$ à accepter sa proposition avec l'identifiant vid_p. Pour cela, p vérifie qu'il n'existe aucun processus $q \in vset_p$ qui ait proposé une proposition contenant le même ensemble $vset_p$ mais avec un identifiant strictement plus grand que vid_p. Cette phase constitue les

tâches 2 et 3. La tâche 2 traite les acquittements et les refus du message de lecture de p. La tâche 3 traite la réception des messages de lecture des autres processus. Nous détaillons maintenant ces deux tâches :

- dans la tâche 2, après avoir diffusé sa proposition, p attend de recevoir un acquittement de son message READ avec l'identifiant vid_p de tous les processus de $vset_p$ et vérifie si un des processus dans $vset_p$ n'est plus considéré comme stable dans sa partition. S'il existe au moins un processus r dans $vest_p$ qui n'est plus considéré comme stable, alors p abandonne sa proposition (ligne 32). p abandonne également sa proposition s'il existe une proposition concurrente avec un numéro d'identifiant plus grand que vid_p : c'est-à-dire que p reçoit un refus de son message READ d'un processus q de $vset_p$ (ligne 58). Notons que le message $(q,\langle\{\text{NACKREAD} \mid vid_p\rangle,\{p\})$ reçu par p correspond au message de refus que q a diffusé à p suite à la réception du message $(p,\langle\text{READ} \mid vid_p\rangle,vset_p)$ de p. De la même manière, le message $(q,\langle\{\text{ACKREAD} \mid vid_p, vid_q\rangle,\{p\})$ reçu par p correspond au message d'acquittement que q a envoyé à p suite à la réception du message $(p,\langle\text{READ} \mid vid_p\rangle,vset_p)$ de p (ligne 49). À la ligne 35, dans les messages ACKREAD et NACKREAD, l'attribut vid_p permet de sélectionner les messages correspondant à la phase READ courant de numéro vid_p. Les messages des anciennes phases READ sont ainsi ignorés. En ce qui concerne les messages ACKREAD, le second numéro vid_q est utilisé dans la suite de la tâche. La diffusion du message $(p,\langle\text{READ} \mid vid_p\rangle,vset_p)$ par p est réalisée à ligne 18. Lorsque p a reçu un acquittement de son message READ en provenance de tous les processus de $vset_p$, p sélectionne la réponse ayant le plus grand numéro d'identifiant (ligne 36). Puis, p met à jour vid_p (ligne 37) et $lastProposal_p$ (ligne 38). Ensuite, p commence la phase d'écriture en diffusant un message WRITE contenant sa proposition avec l'identifiant vid_p ;
- dans la tâche 3, p traite la réception des messages READ reçus. Quand p reçoit un message READ du processus q contenant la proposition de q, p vérifie si la proposition de q est éligible : c'est-à-dire $(vid_p > vid_q \wedge vset_q \not\subseteq \alpha Set_p$ provided by $\Diamond\mathcal{PPD}$) (ligne 45). L'éligibilité d'une proposition se traduit par le fait que tous les membres de la vue proposée doivent être inclus dans αSet_p. En effet, seuls les processus considérés stables peuvent participer à l'exécution de l'algorithme de la gestion de groupe partitionnable. Si c'est le cas, alors l'identifiant de la vue proposée doit être strictement plus grand que le dernier identifiant vu par p car la vue proposée doit être proposée par le leader de $vset$ avec le plus grand identifiant parmi ceux déjà vus par les processus de $vset$. Lorsque la condition $(vid_p > vid_q \wedge vset_q \not\subseteq \alpha Set_p)$ est vérifiée, p vérifie qu'il est inclus dans $vset_q$ avant de répondre à q avec un message ACKREAD incluant l'identifiant vid_q et son identifiant vid_p (ligne 49). Sinon, p refuse la proposition de q et notifie ce dernier par un message NACKREAD (ligne 46) ;

– phase d'écriture : le but de la phase d'écriture est de convaincre l'ensemble des processus de $vset_p$ d'écrire la valeur de p dans le registre. Cependant, p peut échouer dans le cas où il existe un autre processus qui proposer de manière concurrente. La phase d'écriture est

composée des tâches 4 et 5. La tâche 4 traite les acquittements, les refus d'écriture et le changement de valeur de αSet. Quant à la tâche 5, elle traite les requêtes d'écriture. Nous détaillons maintenant ces deux tâches :

- dans la tâche 4, lorsque p reçoit un message NACKWRITE associé à son identifiant courant vid_p, p abandonne sa proposition courante. Comme dans la phase READ, p abandonne également sa proposition courante lorsqu'un ou plusieurs processus de $vset_p$ sont exclus de l'ensemble αSet_p fourni par $\Diamond\mathcal{PPD}$. Sinon, lorsque p reçoit un acquittement pour son message WRITE avec son identifiant courant vid_p de tous les processus de $vset_p$, p décide avec la valeur qu'il a proposée ;
- dans la tâche 5, lors de la réception d'un message WRITE d'un processus q avec un numéro de proposition vid_q, p vérifie s'il est inclus dans $vset_q$ et s'il peut accepter la valeur contenue dans cette proposition, c'est-à-dire $vset_p = vset_q \ \wedge \ vid_q \geqslant vid_p$. Si c'est le cas, p met à jour la valeur de $lastProposal_p$ avec $(vset_q, vid_q)$ (ligne 60). Sinon, p refuse la valeur proposée et notifie l'expéditeur q en émettant un message NACKWRITE.

5.4.2 Preuve de correction

Nous montrons que l'algorithme 5 implante le registre ultime par partition spécifié dans la section 4.3.5 car il respecte les propriétés suivantes :

- $\mathcal{RPP}$-Validité par partition : dans une partition, si un processus p décide la valeur $val = (vset, vid)$, alors $val' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans $vset$;
- $\mathcal{RPP}$-Non trivialité d'abandon : dans une partition, s'il existe un ensemble αSet constitué d'au moins α processus stables et si un processus p de cet ensemble propose une valeur $val = (vset, vid)$ avec $vset \subseteq \alpha Set \wedge |vset| \geqslant \alpha$ une infinité de fois, alors p décide ultimement $val' = (vset, vid')$. S'il n'existe pas un ensemble αSet constitué d'au moins α processus stables, alors p abandonne ;
- $\mathcal{RPP}$-Accord par partition : dans une partition, deux processus dans l'ensemble αSet des processus stables ne décident pas ultimement différentes valeurs.

Lemme 8. *Si le processus p décide la valeur $val = (vset, vid)$, alors $val' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans $vset$.*

Démonstration. Soit p un processus qui décide la valeur $val = (vset, vid)$ à l'instant t. Considérons par contradiction qu'il n'existe pas de processus q dans $vset$ qui a proposé $val' = (vset, vid')$ à l'instant $t_s < t$. p décide val signifie que l'événement $\mathcal{RPP}$-`return`$(p, vset, vid')$ est généré (ligne 60) : c'est-à-dire que p a reçu un acquittement de tous les processus de $vset_p$ pour son message WRITE diffusé. Ce message WRITE est généré suite à la fin d'un message READ, qui à son tour est généré par l'appel de la procédure $\mathcal{RPP}$-*propose*. Ceci est vrai car les liens de communication sont supposés ne pas créer de message. □

Algorithm 5 Implantation de $\diamondsuit\mathcal{RPP}$: phase d'intitialisation (1/3)

```
1   Init():
2   Begin
3       vset_p ←⊥;                                   {p's estimate of next view's members}
4       vid_p ←⊥;                                    {p's estimate of next view's identifier}
5       lastProposal_p ← (vset_p, vid_p);                              {p's last proposal}
6       ppd ← create ◇PPD;
7   End
8
9   Procedure RPP-propose( vset, vid )
10  Begin
11      If lastProposal_p.id ⩾ vid then
12          vid_p ← lastProposal.id + 1;
13      Else
14          vid_p ← vid;
15      vset_p ← vset;
16      lastProposal_p ← (vset_p, vid_p);
17      store(lastProposal_p);
18      xbroadcast(p,⟨READ |vid_p⟩,vset_p);
19  End
20
21  Task T1 : upon recovery
22  Begin
23      retrieve(lastProposal_p);
24  End
25
26  Read and write phases are described in pages 99 and 100, respectively.
```

Algorithm 5 Implantation de $\Diamond\mathcal{RPP}$: phase de lecture (2/3)

```
27  Task T2: upon
28   [xreceive(m) with m = (q,⟨{ACKREAD | vid_p, vid_q⟩,{p}) ∨ m = (q,⟨NACKREAD | vid_p⟩,{p}) : ∀q ∈ vset_p]
29   ∨ [αSet_p provided by ppd has changed ∧ ∃r ∈ vset_p : r ∉ αSet_p]
30  Begin
31      If (αSet_p provided by ppd has changed  ∧ ∃r ∈ vset_p : p ∉ αSet_p) then
32          generate RPP-return(⊥, ⊥);
33      If received at least one (q,⟨NACKREAD | vid_p⟩,{p}) then
34          generate RPP-return(⊥, ⊥);
35      Else
36          select the response (q,⟨ACKREAD | vid_p, vid_q⟩,{p}) with the highest vid_q;
37          vid_p ← max(vid_p, vid_q) + 1;
38          lastProposal_p ← (vset_p, vid_p);
39          store(lastProposal_p);
40          xbroadcast(p,⟨WRITE | vid_p⟩,vset_p);
41  End
42
43  Task T3: upon xreceive(m) with m = (q,⟨READ | vid_q⟩,vset_q)
44  Begin
45      If (vid_p > vid_q ∧ vset_q ⊈ αSet_p provided by ◇PPD ) then
46          xbroadcast (p,⟨NACKREAD | vid_p⟩,{q});
47      Else
48          If p ∈ vset_q then
49              xbroadcast(p,⟨ACKREAD | vid_q, vid_p),{q}⟩;
50      End
```

Algorithm 5 Implantation de $\diamondsuit\mathcal{RPP}$: phase d'écriture (3/3)

```
51  Task T4: upon
52    [xreceive(m) with m = (q,⟨{ACKWRITE | vid_p⟩,{p}) ∨  m = (q,⟨NACKWRITE | vid_p⟩,{p}) : ∀q ∈ vset_p]
53    ∨ [αSet_p provided by ppd has changed ∧ ∃r ∈ vset_p : r ∉ αSet_p]
54  Begin
55      If (αSet_p provided by ppd has changed  ∧ ∃r ∈ vset_p : p ∉ αSet_p) then
56          generate RPP-return(⊥, ⊥);
57      If received at least one (q,⟨NACKWRITE | vid_p⟩,{p}) then
58          generate RPP-return(⊥, ⊥);
59      Else
60          generate RPP-return(vset_p, vid_p);
61      End
62
63  Task T5: upon xreceive(m) with m = (q,⟨WRITE | vid_q⟩,vset_q)
64  Begin
65      If (p ∈ vset_q) then
66          If (vid_q ⩾ vid_p ∨ vid_p =⊥) then
67              lastProposal_p ← (vset_p, vid_q);
68              store(lastProposal_p);
69              xbroadcast(p,⟨ACKWRITE | vid_q⟩,{q});
70          Else
71              xbroadcast(p,⟨NACKWRITE | vid_q⟩,{q});
72  End
```

Lemme 9. *Si p est le seul processus stable dans l'ensemble $\alpha Set \subseteq \diamondsuit PART_p$ avec $|\alpha Set| \geqslant \alpha$ qui propose une valeur $val = (vset_p, vid_p)$ avec $vset_p \subseteq \alpha Set \wedge |vset_p| \geqslant \alpha$ une infinité de fois, alors p décide ultimement $val' = (vset, vid')$.*

Démonstration. Soit p un processus dans l'ensemble des processus stables $\alpha Set \subseteq \diamondsuit PART_p$ qui continue une infinité de fois à proposer sa valeur $val = (vset_p, vid_p)$ avec $vset_p \subseteq \alpha Set \wedge |vset_p| \geqslant \alpha$. Puisque p est le seul proposeur qui continue ultimement à proposer des valeurs, il doit exister un instant t_1 après lequel aucun processus $q \neq p$ dans αSet ne propose de valeur. Montrons par contradiction qu'il existe un instant $t_2 > t_1$ auquel p décide.

Considérons par contradiction qu'il n'existe pas d'instant $t_2 > t_1$ auquel p décide. Après t_1, p propose sa valeur en appelant la procédure $\mathcal{RPP}$-*propose*$(vset_p, vid_p)$ (ligne 9) infiniment souvent. p choisit de façon strictement croissante les identifiants qu'il associe à ses propositions (lignes 12 et 14). Il doit exister un instant $t_3 > t_1$ après lequel vid_p devient plus grand que tous les identifiants déjà rencontrés par les processus de $vset_p$. D'après le lemme 2, il doit exister un instant $t_4 > t_3$ auquel p réussit ultimement à diffuser son message $(p,\langle \text{READ } |vid_p\rangle,vset_p)$ à l'ensemble des processus dans αSet_p incluant $vset_p$. La seule raison empêchant p de diffuser son message WRITE serait qu'il ait reçu un message $(r,\langle \text{NACKREAD} \mid vid_r\rangle,\{p\})$ (ligne 52) d'un autre processus $r \in vset_p$ tel que $vid_r > vid_p$. Ceci contredit le fait que p possède le plus grand identifiant après t_3. D'après le lemme 2, les processus de $vset_p$ reçoivent ultimement le message READ de p contenant vid_p. Ils répondent p avec un message d'acquittement ACKREAD contenant vid_p. p réussit ultimement à terminer la phase de lecture pour sa proposition avec l'identifiant vid_p lorsque p reçoit un acquittement de son message READ avec vid_p. Après t_4, p démarre la phase WRITE contenant $vset_p$ et vid'_p. L'ensemble des processus $vset_p$ est stable. Ces processus reçoivent donc ultimement le message $(p,\langle \text{WRITE} \mid vid'_p\rangle,vset_p)$ et répondent à p avec un message ACKWRITE avec vid'_p. Ainsi, p termine la phase WRITE de sa proposition lorsqu'il reçoit ces acquittements. p décide $val' = (vset_p, vid'_p)$ après $t_4 > t_3 > t_1$. Ceci contredit le fait qu'il n'existe pas d'instant $t_2 > t_1$ auquel p décide. □

Lemme 10. *Soit αSet un ensemble de processus stables, et p et q deux processus tels que $p, q \in \alpha Set \subseteq \diamondsuit PART_p$. Il existe un instant après lequel p et q ne décident pas différentes valeurs.*

Démonstration. Soit αSet un ensemble de processus stables incluant p tel que $p \in \alpha Set \subseteq \diamondsuit PART_p$. D'après le lemme 6, cela signifie qu'il existe un instant t après lequel chaque processus $q \in \alpha Set_p$ possède la même valeur $\alpha Set_l = \alpha Set_p$ pour son propre αSet_q avec l le leader ultime de αSet. Soit $val = (vset_l, vid_l)$ la valeur décidée par l après t. Pour montrer que p et q ne décident pas ultimement différentes valeurs, nous montrons que pour tout $q \in \alpha Set_l$, q décide ultimement la même valeur que l, s'il décide.

Comme l est le leader, il doit exister un instant t_1 après lequel l est le seul proposeur qui continue à proposer des valeurs. D'après le lemme 9, l décide la valeur $val = (vset_l, vid_l)$ qu'il propose : c'est-à-dire la phase de lecture de la proposition avec $vid'_l \leqslant vid_l$ contenant la valeur $vset_l$ est terminée avec succès. Après avoir décidé, l diffuse la valeur décidée à tous les processus de $vset_l$ via la primitive *xbroadcast* (ligne 40). D'après le lemme 2, les processus de

$vset_l$ reçoivent ultimement le message WRITE de p contenant la valeur $val = (vset_l, vid_l)$. Cette valeur est acceptée par ces processus comme dernière proposition acceptée. □

Théorème 3. *$\diamondsuit\mathcal{RPP}$ satisfait les propriétés $\mathcal{RPP}$-Validité par partition, $\mathcal{RPP}$-Accord par partition et $\mathcal{RPP}$-Non trivialité d'abandon.*

Démonstration. Considérons un processus stable p. D'après le lemme 8, si p décide une valeur $val = (vset, vid)$, alors $val' = (vset, vid')$ a été proposée par un processus $q \in vset$ (possiblement p) dans $vset$. Ceci satisfait la propriété $\mathcal{RPP}$-Validité par partition. Selon le lemme 9, si p est le seul processus stable qui propose infiniment souvent une valeur val alors p décide cette valeur. Ceci satisfait la propriété $\mathcal{RPP}$-Non trivialité d'abandon. Selon le lemme 10, si p décide une valeur $val = (vset, vid)$ alors pour tout processus $q \in vset$, q décide val. Ceci satisfait la propriété $\mathcal{RPP}$-Accord par partition. □

Après l'implantation des modules $\diamondsuit\mathcal{RPP}$ et $\diamondsuit\mathcal{PPD}$ ainsi que l'implantation du module de retransmission, nous présentons dans la section 5.5 l'implantation du consensus abandonnable.

5.5 Consensus abandonnable

Cette section présente l'implantation du consensus abandonnable. Celle-ci consiste à combiner une instance du module $\diamondsuit\mathcal{PPD}$ avec une instance du module $\diamondsuit\mathcal{RPP}$. La sûreté du consensus abandonnable provient de la sûreté du module $\diamondsuit\mathcal{RPP}$ tandis que la vivacité du consensus abandonnable est garantie par le module $\diamondsuit\mathcal{PPD}$. Ainsi, comme dans $\diamondsuit\mathcal{RPP}$, tous les processus ne sont pas supposés proposer une valeur, mais seulement ceux qui croient qu'il existe un ensemble αSet constitué d'au moins α processus stables et qui croient être leader de leur partition. L'existence de αSet pendant les périodes stables est garantie par $\diamondsuit\mathcal{PPD}$.

Dans la section 5.5.1, nous détaillons l'implantation du consensus abandonnable. Ensuite, dans la section 5.5.2, nous prouvons sa correction.

5.5.1 Implantation

L'algorithme 6 implante pour le processus p le consensus abandonnable. L'algorithme combine une instance *rpp* du module $\diamondsuit\mathcal{RPP}$ et une instance *ppd* du module $\diamondsuit\mathcal{PPD}$. *decision* sert à stocker la valeur décidée par le consensus abandonnable.

Après avoir proposé une valeur $val = (set, id)$ par appel de la procédure $\mathcal{AC}$-*propose*, le proposeur p vérifie *1)* si l'ensemble αSet fourni par *ppd* est toujours constitué d'au moins α processus, *2)* s'il est le leader de l'ensemble αSet, et *3)* s'il n'a pas déjà décidée. Si c'est le cas, p continue à proposer sa valeur. La décision retournée par le consensus correspond à celle retournée par le module $\diamondsuit\mathcal{RPP}$ (ligne 11). Ainsi, une proposition peut être abandonnée par le module $\diamondsuit\mathcal{RPP}$ lorsqu'il existe un autre proposeur qui propose de façon concurrente, mais l'instance du consensus n'est pas abandonnée tant que le proposeur croit qu'il existe au moins α processus stables dans sa partition et qu'il est le leader de la partition. Le proposeur abandonne une instance de consensus, c'est-à-dire une proposition, sinon.

Observons que l'algorithme du consensus abandonnable peut produire des désaccords entre les processus pendant des périodes instables. Pendant les périodes stables, l'accord sur une valeur unique entre un ensemble de processus est obtenu aussitôt que la condition de stabilité est satisfaite.

Algorithm 6 Implantation pour le processus p du consensus abandonnable

```
1  Init():
2  Begin
3      rpp ← create ♢RPP;
4      ppd ← create ♢PPD;
5      decision ← (⊥, ⊥);
6  End
7
8  Procedure AC-propose(vset, vid)
9  Begin
10     While (|ppd.participants().αSet| ⩾ α ∧ ppd.participants().l = p ∧ decision = (⊥, ⊥)) do
11         decision ← rpp.RPP-propose(vset, vid);
12     generate AC-return(decision);
13 End
```

5.5.2 Preuve de correction

Dans cette section, nous montrons que l'algorithme 6 implante le consensus abandonnable en montrant qu'il satisfait les propriétés suivantes :

- $\mathcal{AC}$-Validité par partition : dans une partition, si un processus p décide la valeur $val = (vset, vid)$, alors $val' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans $vset$;
- $\mathcal{AC}$-Accord par partition : dans une partition, deux processus dans l'ensemble αSet des processus stables ne décident pas ultimement différentes valeurs ;
- $\mathcal{AC}$-Terminaison : dans une partition, le processus p propose une valeur. S'il existe un ensemble αSet constitué d'au moins α processus stables incluant p, alors p décide ultimement. Sinon, p abandonne.

Lemme 11. *Si un processus p décide la valeur $val = (vset, vid)$, alors $val' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans $vset$.*

Démonstration. Soit p un processus qui décide la valeur $val = (vset, vid)$ à l'instant t. D'après la ligne 11, val a été décidée par le module $\diamondsuit\mathcal{RPP}$. D'après le lemme 8, $val' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans $vset$ à l'instant $t' < t$. □

Lemme 12. *Soit αSet un ensemble de processus stables, et p et q deux processus tels que $p, q \in \alpha Set \subseteq \Diamond PART_p$. Il existe un instant après lequel p et q décident la même valeur.*

Démonstration. Soit αSet un ensemble de processus stables, et p et q deux processus tels que $p, q \in \alpha Set \subseteq \Diamond PART_p$. D'après la ligne 11, *val* a été décidée par le module $\Diamond\mathcal{RPP}$ et d'après le lemme 10, il existe un instant après lequel p et q décident la même valeur. □

Lemme 13. *Le processus p propose une valeur. S'il existe un ensemble αSet constitué d'au moins α processus stables élisant p comme leader ultime, alors p décide ultimement. Sinon, p abandonne.*

Démonstration. Le processus p propose une valeur. S'il n'existe pas un ensemble αSet constitué d'au moins α processus stables incluant p, alors p abandonne (lignes 12). Considérons qu'il existe un ensemble αSet constitué d'au moins α processus stables incluant p et que p est le leader. Montrons que p décide ultimement. D'après le lemme 9, si p est le seul processus stable dans l'ensemble $\alpha Set \subseteq \Diamond PART_p$ avec $|\alpha Set| \geqslant \alpha$ qui propose une valeur $val = (vset, vid)$ avec $vset \subseteq \alpha Set \wedge |vset| \geqslant \alpha$ une infinité de fois, p décide ultimement $val' = (vset, vid')$. Si la partition de p n'est pas stable, c'est-à-dire s'il n'existe pas un ensemble αSet constitué d'au moins α processus stables, alors p arrête de proposer à la ligne 11 et sort de la boucle. Par ailleurs, comme p n'avait pas précédemment décidé, *decision* est égal à $(\perp, \perp)$ et p abandonne. □

Théorème 4. *Le consensus abandonnable satisfait les propriétés $\mathcal{AC}$-Validité par partition, $\mathcal{AC}$-Terminaison et $\mathcal{AC}$-Accord.*

Démonstration. Considérons un processus p. D'après le lemme 11, si p décide la valeur $val = (vset, vid)$, alors $val' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans $vset$ et $vset$ contient au moins α_p processus. Ceci satisfait la propriété $\mathcal{AC}$-Validité par partition. Selon le lemme 13, le processus p propose une valeur. S'il existe un ensemble αSet constitué d'au moins α processus stables élisant p comme leader, alors p décide ultimement. Ceci satisfait la propriété $\mathcal{AC}$-Terminaison. Selon le lemme 12, soit αSet un ensemble de processus stables, et p et q deux processus tels que $p, q \in \alpha Set \subseteq \Diamond PART_p$. Il existe un instant après lequel p et q décident la même valeur. Ceci satisfait la propriété $\mathcal{AC}$-Accord. □

5.6 Gestion de groupe partitionnable

Dans cette section, nous complétons la mise en œuvre de notre solution en présentant une implantation qui utilise le consensus abandonnable comme service de base pour installer les vues. Rappelons que la couche applicative est responsable du choix des processus stables inclus dans la vue successeur. La contrainte est que l'ensemble des membres de la vue successeur doit être inclus dans l'ensemble αSet des processus stables fourni par $\Diamond\mathcal{PPD}$. Pour cela, à chaque fois que $\Diamond\mathcal{PPD}$ détecte un changement dans la composition de αSet, le module $\Diamond\mathcal{PPD}$ notifie la couche applicative de la nouvelle valeur de αSet en générant l'événement `alpha_set`. Pour simplifier la présentation sans perdre en généralité, la génération de l'événement `alpha_set` est réalisée de façon implicite.

5.6.1 Implantation

L'algorithme 7 implante pour le processus p le service de gestion de groupe partitionnable. Le but de cet algorithme est double. D'une part, p vérifie s'il existe un ensemble de processus stables αSet dans sa partition. D'autre part, p tente d'installer la même vue inclus dans αSet chez les processus de la vue.

Dans l'algorithme 7, *ac* représente une instance du consensus abandonnable qui est composée d'une instance de $\Diamond\mathcal{PPD}$ et d'une instance de $\Diamond\mathcal{RPP}$. La vue successeur est stockée dans la variable *decision*. α_p est la même variable que celle utilisée dans $\Diamond\mathcal{PPD}$. Toutes les variables sont initialisées dans la phase **Init**.

Pour installer une vue successeur v_{new}, p fait appel à la procédure *propose* qui déclenche un consensus abandonnable avec pour valeur proposée v_{new} telle que $v_{new} = (set, id) \wedge |vset_{new}| \geqslant \alpha_p \wedge id > v.id$. La procédure *propose* se termine ultimement en retournant une valeur qui sera stockée dans *decision*. Cependant, la valeur retournée n'est pas nécessairement une valeur qui a été proposée par un processus, c'est-à-dire que *decision* peut être égale $(\perp, \perp)$. Ceci signifie que la proposition de p pour installer la vue v_{new} a été abandonnée. La couche applicative est alors notifiée de cet abandon par un événement `nack_view_change` (ligne 14). Si $decision \neq (\perp, \perp)$, alors *decision* correspond à la vue successeur de la vue courante v de p. Ensuite, p diffuse de manière fiable la vue décidée *decision* à l'ensemble des membres de cette vue.

Dans la tâche 1, lors de la réception d'un message DECISION de q contenant la valeur décidée $decision_q$, p accepte d'installer la nouvelle vue $decision_q$ diffusée par q à condition que p soit un membre de cette vue et que l'identifiant de la vue courante ne soit pas plus grand que $decision_q.id$ (ligne 19). Si ces conditions sont vérifiées, la couche applicative est notifiée de l'installation de la nouvelle vue via un événement `view_change` (ligne 21).

5.6.2 Preuve de correction

Nous montrons que l'algorithme 7 implante la gestion de groupe partitionnable spécifiée dans la section 4.4.2 car il satisfait les propriétés suivantes :

- auto inclusion : si le processus p installe la vue v alors p est un membre de v ;
- monotonicité locale : si le processus p installe la vue v après avoir installé la vue v' alors l'identifiant de v est plus grand que celui de v' ;
- $\mathcal{PGM}$-Validité : si le processus p installe la vue v alors v a été proposée par un processus q (possiblement p) dans $v.members$ et tel que $|v.members| \geqslant \alpha_p$;
- $\mathcal{PGM}$-Accord sur la vue finale : dans une partition, s'il existe un ensemble $S \subseteq \alpha Set \subseteq \Diamond PART_p$ de processus stables qui souhaitent installer la vue v_f avec $v_f.members = S$, alors tous les processus dans S installent la même vue finale v_f.

Lemme 14. *Si le processus p installe la vue v alors p est un membre de v.*

Démonstration. Soit p un processus qui installe la vue v. Comme les messages DECISION sont envoyés aux processus de la vue v et contiennent la vue v (ligne 12), v est incluse dans le message de décision que p a reçu (ligne 17). Lors de la réception du message décision

Algorithm 7 Implantation pour le processus p de la gestion de groupe partitionnable

```
1  Init():
2  Begin
3      ac ← createAC;                                   {Abortable consensus instance}
4      α_p ← n ⩾ 1;                                     {Same as α_p in ◇PPD_p}
5      decision ← (⊥, ⊥);                               {Tuple (members, id)}
6  End
7
8  Procedure propose(v_new) with v_new = (set, id) ∧ |set| ⩾ α_p ∧ id ⩾ decision.id ∨ decision.id =⊥
9  Begin
10     decision ← ac.propose(v_new.set, v_new.id);      {Abortable consensus decision}
11     If decision ≠ (⊥, ⊥) then
12         xbroadcast(p,⟨DECISION | decision⟩, decision.members);
13     Else
14         generate nack_view_change(v_new);
15 End
16
17 Task 1: upon xreceive(m) with m = (q,⟨DECISION | decision_q⟩,∗)
18 Begin
19     If (p ∈ decision_q.members ∧ decision_q.id > decision.id ∨ decision.id =⊥) then
20         decision ← decision_q;
21         generate view_change(decision);              {Notify application layer about current view}
22 End
```

$(q, \langle \text{DECISION} \mid decision_q \rangle, *)$, p n'installe la nouvelle vue v avec $v.members = decision_q.members$ et $v.id = decisions_q.id$ (lignes 20–21) que s'il appartient à $v.members$ (ligne 19). □

Lemme 15. *Si le processus p installe la vue v après avoir installé la vue v' alors l'identifiant de v est plus grand que celui de v'.*

Démonstration. Soit p un processus et v' sa vue courante. p installe la nouvelle vue v successeur de v' seulement si $v.id > v'.id$ et $p \in v.members$ (ligne 19). □

Lemme 16. *Si un processus p décide la valeur $val = (vset, vid)$, alors $val' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans $vset$ et $vset$ contient au moins α_p processus.*

Démonstration. Soit p un processus qui décide la valeur $val = (vset, vid)$ à l'instant t. D'après le lemme 11, $val' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans $vset$ à l'instant $t' < t$. Supposons que l est le processus qui a proposé la valeur val' et est le premier à décider val : c'est-à-dire l est le premier qui a diffusé originellement le message DECISION contenant la valeur val. Cela signifie que l a terminé son instance de consensus avec comme valeur proposée $val = (vset, vid)$ et que l'événement $\mathcal{AC}\text{-}\texttt{return}(p, decision_p)$ avec $decision = (vset, vid)$ a été généré. Ceci n'est possible que si l croit être le leader de $\alpha Set \supseteq vset$ (car seuls les proposeurs peuvent proposer des valeurs) et si $|vset| \geqslant \alpha_l$ (ligne 8) à l'instant auquel l propose la valeur $val = (vset, vid)$. Par conséquent, pour tout processus q dans $vset$, $\alpha_l \geqslant \alpha_q$. Les processus dans l'ensemble $vset$ des processus stables reçoivent ultimement la décision de l contenant val avec $|val.vset| \geqslant \alpha_l \geqslant \alpha_q$ pour tout q dans $vset$. □

Lemme 17. *S'il existe ultimement un ensemble $S \subseteq \alpha Set \subseteq \Diamond PART_p$ des processus stables avec $\alpha \leqslant |S| \leqslant |\alpha Set|$, alors tous les processus dans S installent la même vue finale.*

Démonstration. Soit αSet un ensemble de processus stables, et p et q deux processus tels que $p, q \in \alpha Set \subseteq \Diamond PART_p$. D'après le lemme 10, il existe un instant après lequel p et q décident la même valeur $val = (S, vid)$. D'après le lemme 16, $val' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans $vset$ et S est inclus dans αSet_p et contient au moins α_p processus. □

Théorème 5. *L'algorithme 7 satisfait les propriétés d'auto-inclusion, de monotonicité locale, $\mathcal{PGM}$-Validité et $\mathcal{PGM}$-Accord sur la vue finale.*

Démonstration. Soit p un processus. D'après le lemme 14, si p installe la vue v' alors p est un membre de v. Ceci satisfait la propriété d'auto-inclusion. D'après le lemme 15, si p installe la vue v après avoir installé la vue v' alors $v.id > v'.id$. Ceci satisfait la propriété de monotonicité locale. Selon le lemme 16, si un processus p décide la valeur $val = (vset, vid)$, alors $val' = (vset, vid')$ a été proposée par un processus q (possiblement p) dans $vset$ et $vset$ contient au moins α_p processus. Ceci satisfait la propriété $\mathcal{PGM}$-Validité. D'après le lemme 17, s'il existe un ensemble $S \subseteq \alpha Set \subseteq \Diamond PART_p$ de processus stables, alors tous les processus dans S installent la même vue finale. Ceci satisfait la propriété $\mathcal{PGM}$-Accord sur la vue finale. □

5.7 Travaux connexes

Cette section discute des travaux rencontrés dans la littérature. Dans la section 5.7.1, nous discutons des travaux qui ont introduit le concept de détecteur de participants dans les MANETs. Ensuite, dans la section 5.7.2, nous présentons quelques travaux sur les registres dans les systèmes dynamiques. Enfin, dans la section 5.7.3, nous présentons d'autres approches pour la gestion de groupe pour les MANETs.

5.7.1 Détecteurs de participants dans les réseaux mobiles spontanés

[Aguilera et al., 1999] présente un détecteur de défaillances à base de battements de cœur (en anglais, *heartbeat failure detector* ou $\mathcal{HB}$) pour un réseau partitionnable. Le module fournit à chaque processus p un vecteur dont les éléments correspondent à des compteurs de battements de cœur des messages reçus par les processus du système. Le nombre de battements cœur de chaque processus qui n'est plus dans la même partition que p est borné. La notion de détecteur ultime des α participants d'une partition s'est inspirée de ces travaux. Cependant, à la différence de notre approche, le nombre et l'identité des processus présents dans le système sont connus dès l'initialisation. Autrement dit, le système proposé par [Aguilera et al., 1999] est statique. Les auteurs considèrent deux types de liens : équitables et ultimement inactifs. Au dessus des liens équitable, les liens quasi-fiables-∞ [Boichat et al., 2003a] et quasi-fiables [Aguilera and Toueg, 1997] peuvent être implantés en retransmettant les messages diffusés. Cependant, les propriétés des liens quasi-fiables-∞ et quasi-fiables ne peuvent pas être garanties quand le système se partitionne de façon permanente. En outre, les liens équitables ne sont pas créés dynamiquement, mais sont supposés exister dès l'initialisation. Dans notre modèle, les liens SADDM sont une combinaison des liens équitables et des liens à garantie temporelle de telle sorte que la vivacité d'une partition peut être garantie, même si la partition n'est pas complètement stable. Un lien SADDM est plus faible qu'un lien à garantie temporelle puisqu'il autorise que des messages soient perdus et que d'autres soient transmis avec des délais de transmission non bornés. Un lien SADDM est plus fort qu'un lien équitable puisqu'il assure qu'un sous-ensemble des messages transportés sont reçus ultimement par le destinataire et que les messages de cet ensemble ne sont pas trop dispersés dans le temps. En outre, nous définissons les chemins SADDM de sorte à capturer le fait que les chemins dans les MANETs sont dynamiquement créés. Nous estimons que ce type de lien et de chemin correspond mieux aux communications dans les MANETs.

[Chockler et al., 2001, Babaoğlu et al., 2001] ont étendu le concept de détecteur de défaillances ultimement parfait $\Diamond\mathcal{P}$ pour l'adapter aux systèmes partitionnables afin de proposer une solution pour la gestion de groupe partitionnable. La version adaptée du module $\Diamond\mathcal{P}$ proposée par [Chockler et al., 2001] reste très similaire à $\Diamond\mathcal{P}$ dans le sens où la vivacité d'une partition ne peut être garantie que si la partition est ultimement complètement stable, c'est-à-dire qu'ultimement aucun message échangé n'est perdu et aucun message n'est échangé entre les processus dans la partition et les processus hors de la partition. [Babaoğlu et al., 2001] propose une version adaptée du module $\Diamond\mathcal{P}$ qui est basée sur la notion d'atteignabilité. Si un processus p envoie un message m à un processus q à l'instant t alors q reçoit m si

et seulement si q est atteignable par p à l'instant t. Cependant, comme il a été souligné dans [Pleisch et al., 2008], la relation d'atteignabilité n'est pas un invariant dans le temps : q peut être atteignable par p à l'instant t, et non atteignable à l'instant $t' > t$. Enfin, le modèle de système dans [Chockler et al., 2001, Babaoğlu et al., 2001] est statique (Π est connu et fixe) et le graphe du réseau est initialement fortement connexe.

Par analogie avec le concept de détecteur de participants dans une partition, la notion de détecteur de participants non fiable (en anglais, *participant detector*) a été introduite dans [Cavin et al., 2004] pour résoudre le problème d'auto-amorçage dans un MANET. [Cavin et al., 2004] définit un détecteur de participants conçu pour les MANETs. De manière similaire à notre approche, le modèle défini considère que l'identité et le nombre de nœuds dans le réseau sont initialement non connus. Cependant, le graphe de réseau considéré est toujours ultimement connexe durant une exécution et les liens qui lient les nœuds entre eux sont bidirectionnels. [Cavin et al., 2004] étudie les hypothèses minimales qui doivent être ajoutées au système afin de permettre aux processus de résoudre le problème du consensus uniforme en dépit de l'absence de connaissance initiale de l'ensemble Π. [Greve and Tixeuil, 2007] étend [Cavin et al., 2004] et identifie une hypothèse de synchronie dite minimale pour résoudre le consensus régulier et le consensus uniforme dans des scénarios autorisant les défaillances des processus. Cependant, les modèles dans ces deux travaux ne considèrent pas le cas du partitionnement permanent du réseau. Ces modèles sont conçus pour les systèmes de partition primaire dans lesquels le nombre total de processus dans le système est fini et le graphe du réseau est toujours connexe. Dans notre modèle, le système est dynamique et partitionnable, et le réseau n'est pas toujours ultimement connexe.

[Nesterenko and Schiper, 2007] propose un détecteur d'atteignabilité $\diamondsuit\mathcal{R}$ qui fourni un ensemble de processus corrects, appelé quorum, à chaque processus. Les auteurs définissent le concept de graphe d'atteignabilité R qui est un mutigraphe orienté dans lequel les nœuds dans R représentent les processus participant au système. Dans le multigraphe orienté, il existe un chemin du processus p vers le processus q si la valeur de sortie du quorum de p contient q. Les auteurs montrent que le module $\diamondsuit\mathcal{R}$ peut être étendu et adapté au réseau partitionnable si les propriétés de complétude (ultimement, chaque processus, soit défaille, soit obtient un quorum ne contenant que des processus corrects) et d'intersection (le quorum fourni à un processus contient les processus corrects) de $\diamondsuit\mathcal{R}$ sont limitées aux processus dans la même partition. $\diamondsuit\mathcal{R}$ est similaire à $\diamondsuit\mathcal{PPD}$ dans le sens où $\diamondsuit\mathcal{R}$ fournit à chaque processus un quorum qui contient les processus mutuellement atteignables dans la partition. Cependant, le modèle défini est différent du nôtre dans le sens où le système n'est pas dynamique, le nombre et l'identité des processus dans le système sont connus des processus, et les liens sont supposés fiables. En outre, aucune implantation de $\diamondsuit\mathcal{R}$ n'est fournie.

[Greve et al., 2011] étend le mécanisme de communication QUERY-RESPONSE de [Mostefaoui et al., 2005] en considérant la mobilité des nœuds, et propose un détecteur de défaillances $\diamondsuit S^M$ qui détecte ultimement l'ensemble des processus connus et stables : un processus est *connu* s'il a rejoint le système et a été identifié par un processus stable ; un processus est *stable* si après qu'il a rejoint le système, il ne quitte plus le système. De manière similaire à notre approche, les auteurs supposent qu'il existe α processus stables

dans le système pour garantir la progression et la terminaison de l'exécution de l'algorithme. Dans [Greve et al., 2011], la valeur α d'un processus p est fonction de la densité du voisinage de p moins le nombre maximal de processus pouvant subir des défaillances dans le voisinage de p. Cependant, comme dans [Chandra and Toueg, 1996], $\diamondsuit S^M$ est conçu pour les systèmes de partition primaire.

[Conan et al., 2008] propose un détecteur ultime d'une partition qui utilise les informations fournies par un détecteur de battements de cœur [Aguilera et al., 1999] et un détecteur déconnexion. Cependant, le nombre de nœuds est connus et la solution proposée n'est basée ni sur la définition des périodes stables ni sur la définition des chemins dynamiques.

Dans notre travail précédent [Arantes et al., 2010], nous avons proposé un modèle de système dynamique pour les MANETs. Comme dans [Chockler et al., 2001], les propriétés de vivacité du système sont garanties seulement dans les partitions complètement stables. Une implantation d'un détecteur de participants de partition utilisant des chemins dynamiques est également proposée afin de capturer cette propriété de vivacité. Nous avons étendu le concept de chemin dynamique afin de proposer la notion de chemin SADDM qui permet de garantir les propriétés de vivacité dans les partitions non complètement stables. L'idée de la combinaison des chemins équitables avec des chemins à garantie temporelle provient de [Sastry and Pike, 2007] via la notion de lien ADD (en anglais, *Average Delayed/Dropped*).

5.7.2 Registres pour les systèmes dynamiques

Cette section présente quelques travaux liés au concept de registre pour les systèmes dynamiques tels que les MANETs. Des études sur les registres conçus pour les systèmes statiques, incluant le registre sûr, le registre régulier et le registre atomique peuvent être trouvées dans [Lamport, 1986, Attiya and Bar-Noy, 1995].

[Baldoni et al., 2009a] implante un registre régulier dans un système dynamique/sans recouvrement [25]/sans partitionnement avec arrivée infinie à niveau d'accès simultanée bornée [Tucci-Piergiovanni and Baldoni, 2010]. Dans [Baldoni et al., 2009a], un processus est dit actif à partir du moment où il a rejoint le système et jusqu'à ce qu'il quitte le système. En outre, le taux de désabonnement C (en anglais, *churn rate*) est constant : le nombre de processus qui rejoignent le système est égal au nombre de processus qui quittent le système à chaque unité de temps [26]. La notion de registre régulier a été adaptée pour les systèmes dynamiques. L'algorithme satisfait les deux propriétés suivantes [Baldoni et al., 2009a] :

- *vivacité* : si un processus invoque une opération de lecture ou d'écriture et qu'il ne quitte pas le système, alors l'opération se termine ultimement en retournant une valeur (s'il agit d'une opération de lecture) ou en écrivant une valeur dans le registre (s'il s'agit d'une opération d'écriture) ;

25. Dans [Baldoni et al., 2009a], un processus défaillant correspond à un processus qui quitte involontairement le système.

26. Notons que la notion de taux de désabonnement est étudié dans le contexte des systèmes répartis à large échelle tels que les systèmes pair-à-pair, les grappes et les grilles de calcul [Ko-Steven et al., 2008]. Cette notion de désabonnement est aussi utilisée dans le contexte des MANETs expérimentaux [Fernández et al., 2006].

– *sûreté* : une opération de lecture retourne la dernière valeur écrite par une opération d'écriture qui précède l'invocation de cette opération de lecture ou une valeur écrite par une opération d'écriture concurrente.

[Baldoni et al., 2009a] propose deux implantations : une pour les systèmes synchrones et une autre pour les systèmes ultimement synchrones [27]. Nous nous intéressons à la deuxième implantation. Dans cette dernière, il existe un instant après lequel les liens de communication possèdent des garanties temporelles : il existe un instant τ et une borne δ tel que chaque message envoyé (ou diffusé) à l'instant $\tau' \geqslant \tau$ est reçu avant l'instant $\tau' + \delta$ par le(s) processus destinataire(s) dans le système. τ et δ sont inconnus des processus. En outre, il est supposé que la majorité des processus qui constituent le système sont actifs.

De manière similaire aux système statiques, la majorité des processus actifs est nécessaire à chaque instant pour garantir la cohérence du registre régulier. Les auteurs supposent également que $C < 1/(3\delta n)$ reste constant et est connu des processus. Une implantation du registre est fournie. Cette implantation est similaire à l'implantation de la gestion de groupe de partition primaire présentée dans le chapitre 3 dans le sens où il est supposé une majorité de processus corrects parmi les membres de la vue courante. En revanche, dans l'implantation de la gestion de groupe de partition primaire que nous avons présentée, le taux de désabonnement n'est pas considéré : les contraintes sont que l'intersection des membres de deux vues consécutives est non vide et que la majorité des membres dans chaque vue sont corrects. [Baldoni et al., 2009b] étend le modèle proposé dans [Baldoni et al., 2009a] pour construire le registre dynamique quand le taux de désabonnement n'est pas constant.

5.7.3 Gestion de groupe partitionnable pour les réseaux mobiles spontanés

[Filali et al., 2006a] propose un algorithme de gestion de groupe partitionnable pour les MANETs dans lesquels les nœuds et les liens de communication sont supposés apparaître et disparaître continuellement. Les partitions correspondent aux cliques [28] dans le graphe du réseau. Chaque processus installe ultimement la vue qui contient les membres de sa clique courante. Un critère de stabilité, appelé critère maximal, est proposé qui est basé sur la non-extensibilité des cliques : deux cliques ne peuvent pas être fusionnées pour former une clique plus large. Pour construire les cliques non extensibles, l'algorithme de gestion de groupe partitionnable procède principalement en deux phases : phase de découverte et phase de publication. Durant la phase de découverte, chaque processus détermine l'ensemble de ses voisins directs. Ensuite, il commence la phase de découverte en diffusant la liste de ses voisins. Quand un nœud reçoit ces listes de tous ses voisins, il possède une vision de l'ensemble de ses voisins à deux sauts. Avec cette connaissance, chaque nœud, soit décide d'attendre une requête d'inclusion, soit décide de déterminer une clique (c'est-à-dire une vue) et diffuse cette vue aux nœuds membres. Seuls les nœuds leaders (par exemple, ceux qui possèdent les plus grands identifiants) sont autorisés à déterminer leur clique ; ils sélectionnent seulement leurs voisins qui se trouvent au plus à deux

27. Comme montré dans [Baldoni et al., 2009a], il est impossible d'implanter le registre régulier dans un système complètement asynchrone.

28. Informellement, une clique est un sous-graphe complet.

sauts. Ainsi, si le même nœud est sélectionné par deux nœuds alors la distance entre ces deux nœuds est au plus 2. Par conséquent, le nœud est ultimement sélectionné par au plus un des deux nœuds (celui avec le plus grand identifiant). Ceci permet de garantir la non-extensibilité quand le système reste relativement stable. [Filali et al., 2006a] ne donne aucune définition de la stabilité et note seulement qu'un système relativement stable n'exclut pas les défaillances et les déconnexions des processus. Cette approche est similaire à notre approche dans le sens où l'intersection de deux ensembles de processus stables $\alpha Set'$ et αSet est vide pendant les périodes stables. Autrement dit, un nœud ne peut pas être membre de deux vues différentes. Cependant, dans notre solution, les membres du groupe ne sont pas limités aux nœuds voisins à deux sauts. En outre, dans une partition stable, seule l'existence d'un chemin SADDM entre deux processus est requise. Enfin, les périodes stables associées aux chemins SADDM sont définies.

[Boulkenafed et al., 2005] propose un service de gestion de groupe partitionnable pour les MANETs. Selon [Boulkenafed et al., 2005], la gestion de groupe pour les MANETs peut être définie de manière similaire à celle pour les réseaux fixes car elle doit être définie selon la propriété fonctionnelle f à réaliser. f correspond à différents attributs d'intérêt supportés par les nœuds. Les différentes attributs d'intérêt considérés sont la localisation, le domaine de sécurité, les contraintes liés à la qualité de service et la connectivité. Chaque membre du groupe est supposé offrir une telle propriété fonctionnelle f. Un groupe qui réalise une propriété fonctionnelle f est dénotée par G^f. Un nœud n'est inclus dans un groupe que s'il est éligible selon les attributs requis par le groupe. [Boulkenafed et al., 2005] propose une solution qui est proche de notre approche et les attributs d'intérêt peuvent peut-être être utilisés comme critères de stabilité. Il est important de noter que la plupart des attributs fonctionnels sont liés à la connectivité du réseau. Nous avons proposé le critère de stabilité nombre de battements de cœur par période afin de capturer la connectivité (l'atteignabilité) et la stabilité du réseau. En outre, comme dans notre approche, [Boulkenafed et al., 2005] propose un mécanisme d'élection du leader afin de gérer l'installation de la vue du groupe parmi les membres du groupe.

5.8 Conclusion

Dans ce chapitre, nous avons mis en œuvre une implantation de notre approche pour la résolution de la gestion de groupe partitionnable en utilisant le consensus abandonnable comme service de base. Le consensus abandonnable est construit comme la combinaison des deux modules détecteur ultime des α participants d'une partition $\Diamond\mathcal{PPD}$ et registre ultime par partition $\Diamond\mathcal{RPP}$. Pour tolérer les pertes de messages entre les processus à travers les chemins SADDM, la gestion de groupe et le registre ultime par partition utilisent le module de retransmission des messages pour diffuser les messages dans la partition de manière fiable. Chacun des modules $\Diamond\mathcal{PPD}$, module de retransmission, $\Diamond\mathcal{RPP}$, consensus abandonnable et gestion de groupe partitionnable est implanté de façon indépendante. La preuve de correction de chaque implantation est fournie.

Concernant les implantations des modules que nous avons présenté, plusieurs perspectives nous semblent intéressantes. Tout d'abord l'étude de la complexité des algorithmes proposés en terme de temps de détection de la condition de stabilité dans le pire des cas nous semble

intéressante. En guise de premier élément, nous avons montré que dans le pire des cas, si S est constitué de α processus stables, alors le délai de transfert d'un message d'un processus p vers un processus q dans ce groupe est au plus $\beta^{\alpha-1}\eta + (\alpha-1)\delta$ secondes car le chemin le plus long entre deux processus du groupe est au plus constitué de $\alpha - 1$ processus. β et δ sont des constantes utilisées dans la définition d'un lien SADDM et η est la période de temps maximale qui sépare deux diffusions consécutives dans le module de retransmission.

Dans les implantations de $\diamondsuit\mathcal{PPD}$ et de la gestion de groupe partitionnable, α est une valeur fournie par l'application et correspond au nombre minimum de processus stables dans la partition pour exécuter les algorithmes. Il nous semble intéressant d'étudier la possibilité de rendre la valeur de α dynamique. La couche applicative peut modifier la valeur α en fonction du nombre de processus détectés comme stables qui constituent l'ensemble αSet. Intuitivement, si la valeur α est trop importante par rapport au nombre de processus présents αSet, alors la couche applicative peut diminuer la valeur de α. Au contraire, si la valeur de α est beaucoup plus faible que le nombre de processus présents dans l'ensemble αSet, alors l'application peut augmenter la valeur de α. Malgré les changements de la valeur de α, $\diamondsuit\mathcal{PPD}$ devrait détecter de manière cohérente un ensemble de processus stables et un leader ultime parmi ces processus.

En complément de l'étude de la dynamicité de la valeur de α, nous pouvons étudier l'influence du taux de désabonnement tel qu'étudié dans les systèmes pair-à-pair. En fonction du taux de désabonnement, la couche applicative pourrait peut-être adapter la valeur de α.

Dans l'implantation du module $\diamondsuit\mathcal{PPD}$, les valeurs de temporisations ${}_{part}Timer$ et ${}_{proc}Timer$ augmentent indéfiniment si la condition de stabilité à l'instant de stabilisation local à la partition n'est pas encore atteint. Une autre perspective est d'étudier la possibilité de décrémenter les valeurs des temporisations de telle sorte que la période de détection de la condition de stabilité ne devienne pas trop grande.

Concernant l'implantation de $\diamondsuit\mathcal{RPP}$, observons que dans le scénario d'exécution de l'algorithme du consensus Synod de Paxos, un proposeur de p_j (avec $j \in [2,4]$) choisit un numéro de scrutin $B_{i,j}$ pour sa i^e (avec $i \geqslant 1$) proposition, qui est égal à $3(i-1)+j$. Plus généralement, s'il existe n processus dans le système qui exécutent le consensus, alors un proposeur p_j (avec $j \in [1,n]$) choisit un numéro de scrutin $B_{i,j}$ pour sa i^e proposition avec $B_{i,j} = n(i-1)+j$. Ce choix est optimal dans le sens où le pas de valeur n est optimal, c'est-à-dire qu'un proposeur ne peut pas choisir un pas de valeur $n' < n$ sans violer l'unicité des numéros de scrutins et donc sans altérer la correction de l'algorithme. Par analogie avec les numéros de scrutins utilisés dans Synod, il nous semble intéressant d'étudier comment les proposeurs dans l'algorithme $\diamondsuit\mathcal{RPP}$ pourrait choisir les numéros des identifiants de la manière la plus optimale possible.

Une dernière perspective concerne la communication efficace. Selon [Delporte-Gallet et al., 2001], dans un système de partition primaire composé de n processus, la communication est efficace si, ultimement, seuls n liens transportent des messages entre processus corrects et les messages transportés sont garantis d'être reçus ultimement. Le modèle de [Delporte-Gallet et al., 2001] est un modèle de système de partition primaire statique dans lequel le graphe du réseau est fortement connexe et il existe un lien bidirectionnel qui relie chaque paire de processus du système. Dans notre modèle de système adapté aux MANETs, le graphe du réseau n'est pas nécessairement fortement connexe et est partitionnable. Les nœuds

stables communiquent entre eux à travers des chemins SADDM représentés par des séquences de liens SADDM unidirectionnels établis entre les différents nœuds du réseau sans fil de façon dynamique. Ainsi, la notion de communication efficace proposée par [Delporte-Gallet et al., 2001] doit être adaptée. Il nous semble intéressant d'étudier une autre forme de communication efficace pour $\Diamond\mathcal{PPD}$. Comme la communication dans les MANETs est multisaut, il est donc impossible d'assurer la communication entre les processus de αSet avec seulement $|\alpha Set|$ liens SADDM. En guise de premier élément, nous pouvons observer que dans l'implantation du module $\Diamond\mathcal{PPD}$, ultimement seul le leader de αSet continue à diffuser originellement les messages ALPHASET.

Enfin, en complément de l'étude théorique, nous évaluons par simulation l'algorithme qui implante $\Diamond\mathcal{PPD}$. Cette évaluation est l'objectif du chapitre 6.

Chapitre 6

Évaluation du détecteur des α participants d'une partition ultime par simulation

Sommaire

6.1 Introduction

L'évaluation d'un système conçu au dessus des MANETs peut être effectué selon deux approches. La première approche consiste à utiliser des bancs d'essai (en anglais, *testbeds*). [Hogie et al., 2006, De et al., 2005] présente un état de l'art sur les bancs d'essais construits pour l'évaluation des MANETs. Bien qu'ils soient précis et pertinents, les bancs d'essai possèdent plusieurs inconvénients. Leur mise en œuvre est onéreuse tant en efforts qu'en matériels. À cela s'ajoute la difficulté du déploiement et de la gestion de la plateforme pour obtenir une analyse et une visualisation pertinentes de l'exécution du système simulé. En outre, il est très difficile de faire passer à l'échelle le système étudié. Enfin, il est difficile, si ce n'est impossible, de reproduire les résultats obtenus. Dans la seconde approche, l'évaluation est réalisée par simulation. Les simulations permettent de modéliser le système et de l'étudier dans son ensemble ou par morceaux. Elles fournissent des mécanismes pour le contrôle et la visualisation, et permettent de suivre l'évolution du système. En outre, les expérimentations sont décrites dans des scénarios qui sont facilement reproductibles. Par ailleurs, la taille du réseau à simuler est seulement limitée par la puissance de calcul disponible pour les simulations. Enfin, les simulations permettent de tester à moindre coût les nouveaux systèmes ou algorithmes. La faiblesse principale des études basées sur la simulation est la fiabilité des résultats obtenus qui dépend de la qualité de la modélisation des caractéristiques physiques du réseau. Par conséquent, puisque nous ne possédons ni le matériel ni le temps pour réaliser un banc d'essai, nous choisissons l'approche par simulation pour évaluer nos algorithmes.

Dans notre étude, nous considérons les quatre critères d'efficacité proposés par [Aschenbruck et al., 2010] :

- reproductibilité : les résultats obtenus doivent être reproductibles afin de permettre la vérification ;
- absence de biais : les résultats ne doivent pas être biaisés parce que spécifiques aux scénarios choisis ;
- rigueur : les scénarios et conditions utilisés dans la simulation doivent prendre en compte les propriétés des MANETs ainsi que la nature de l'application ;
- analyse statistique : l'exécution et l'analyse de l'expérimentation doivent être basées sur des principes mathématiques.

Certaines techniques et méthodes de simulation des MANETs peuvent être adoptées [Heidemann et al., 2001, Hogie et al., 2006]. Elles consistent d'abord à étudier le degré de granularité utilisé dans la modélisation du système étudié. Il est en effet impossible de modéliser tous les détails du monde réel. Ainsi, le réseau est nécessairement modélisé avec un certain degré de granularité en ce qui concerne la mobilité des nœuds, la propagation radio et la consommation d'énergie. Dans le choix de l'outillage, nous mettons l'accent sur deux éléments pour lesquels nous étudions l'état de l'art : la mobilité des nœuds, c'est-à-dire les modèles de mobilité, et les simulateurs, c'est-à-dire leur précision.

Dans la section 6.2, nous présentons des modèles de mobilité pour les MANETs et sélec-

tionnons les plus pertinents pour notre étude. Puis, dans la section 6.3, nous comparons les différents simulateurs récemment utilisés dans les études en environnement MANETs et choisissons le couplage BonnMotion / OMNeT++ / MiXiM. Dans la section 6.4, nous analysons et discutons l'évaluation des performances de $\diamondsuit\mathcal{PPD}$. Enfin, nous concluons dans la section 6.5.

6.2 Modèles de mobilité pour les réseaux mobiles spontanés

Un modèle de mobilité reflète le comportement spatio-temporel des nœuds mobiles dans un réseau. Le but d'un modèle de mobilité est de représenter le plus fidèlement possible les conditions de déplacement des nœuds mobiles dans un contexte particulier du monde réel. Dans notre étude, nous nous intéressons à la modélisation des déplacements humains dans différents scénarios. Dans le reste de ce chapitre, quand ce n'est pas ambigu, nous utilisons le terme « nœuds (mobiles) » pour désigner les individus à pieds, en vélo ou en voiture. Le choix du modèle de mobilité influence le comportement et les performances des protocoles étudiés [Camp et al., 2002].

La littérature identifie trois lois de mouvement des nœuds mobiles : *1)* aléatoire, *2)* déterministe, et *3)* hybride. Les modèles aléatoires, aussi appelés modèles synthétiques (en anglais, *synthetic models*), sont des modèles mathématiques : un ensemble d'équations tente de capturer les mouvements des nœuds. Au contraire, les modèles déterministes s'appuient sur des traces de mouvements réelles. Les traces consistent en des informations de connectivité ou de localisation enregistrées avec des dispositifs GPS et Bluetooth (cf. par exemple les études de l'Université de Cambridge et d'Intel [Hui et al., 2005, Chaintreau et al., 2005]). Ces données de traces contiennent entre autres les identifiants des appareils mobiles se trouvant dans la portée de transmission d'un appareil. Enfin, les modèles hybrides sont une approche alternative et consistent à proposer un modèle synthétique à partir de traces réelles.

Dans la plupart des travaux de la littérature, les modèles synthétiques sont préférés aux autres modèles [Camp et al., 2002, Musolesi and Mascolo, 2007]. Nous adoptons ce choix qui peut être justifié comme suit. Tout d'abord, la plupart des traces ne sont pas libres de droit et il existe seulement un nombre limité de traces publiées. Notons que pour pallier ce problème, le projet CRAWDAD mené à l'Université de Dartmouth [Yeo et al., 2006] a vu le jour avec pour but la création d'un dépôt de traces libres mises à disposition des chercheurs. Mais, ces traces sont collectées à partir de scénarios très spécifiques tels que les déplacements des étudiants et des chercheurs dans un campus à partir desquels il est difficile de généraliser leur valeur. En outre, les traces disponibles ne permettent pas de faire des analyses de sensibilité [29] des performances des algorithmes étudiés car les paramètres caractérisant les scénarios de simulation tels que la distribution de la vitesse et la densité spatiale des nœuds ne peuvent pas être modifiées. Par ailleurs, il est intéressant de posséder un modèle mathématique pour analyser formellement l'impact de la mobilité sur la conception des algorithmes. Notons que les modèles hybrides réalisent un compromis entre la simplicité (des modèles synthétiques) et le réalisme des traces réelles. Cependant, ils restent difficiles à mettre en œuvre. La difficulté est de capturer et de

29. Valeurs d'entrées extrêmes du modèle ou changements drastiques dans la structure du modèle [Kleijnen, 1999].

modéliser les propriétés statistiques intrinsèques des traces réelles afin de les reproduire, et si possible, de les généraliser.

La littérature propose plusieurs modèles de mobilité pour la génération des traces synthétiques. Un survol des modèles de mobilité synthétiques pour les MANETs peut être consulté dans [Davies, 2000, Camp et al., 2002, Feeley et al., 2004, Pelov, 2009]. Plus spécifiquement, les scénarios de mobilité considérés dans les MANETs comprennent par exemple des utilisateurs nomades dans une grande surface, des groupes de touristes en visite dans un musée, des voitures circulant dans un quartier, ou encore des unités militaires lors d'une opération. Ces modèles de mobilité sont classés principalement en deux catégories [Camp et al., 2002] : *1)* mobilité individuelle et *2)* mobilité de groupe. Dans les modèles de mobilité individuelle, les mouvements des nœuds sont déterminés indépendamment les uns des autres. Dans les modèles de mobilité de groupe, il existe une corrélation entre les mouvements de certains nœuds.

Dans cette section, nous commençons par définir nos critères d'évaluation des modèles de mobilité dans la section 6.2.1. Ensuite, nous passons en revue l'état de l'art des modèles de mobilité individuelle et des modèles de mobilité de groupe pour les réseaux partitionnables dans les sections 6.2.2 et 6.2.3, respectivement. Avant la synthèse effectuée en section 6.2.5, nous présentons nos exigences concernant l'outillage de la génération automatique de scénarios de mobilité dans la section 6.2.4.

6.2.1 Critères d'évaluation des modèles de mobilité

Notre algorithme $\diamondsuit\mathcal{PPD}$ est conçu pour des MANETs et peut être utilisé dans plusieurs scénarios de mobilité différents. Les modèles de mobilité existants ne sont en général pas conçus pour représenter le comportement des nœuds mobiles dans tous les scénarios. Donc, nous devons évaluer $\diamondsuit\mathcal{PPD}$ avec plusieurs modèles de mobilité différents comme suggéré dans [Camp et al., 2002]. Nous distinguons dans la suite deux ensembles de critères : les critères généraux puis les critères spécifiques aux réseaux partitionnables.

Critères généraux

Nous avons retenu les critères d'appréciation généraux proposés par [Pelov, 2009]. Un modèle de mobilité idéal devrait présenter les propriétés suivantes :

- présence de modèle de mobilité de groupe : indique si le modèle de mobilité peut générer des patrons de mouvements de groupe ou non ;
- modèle analysé : le modèle de mobilité doit être analysé, c'est-à-dire que certaines de ses propriétés mathématiques ont été analysées ;
- mouvements de nœuds limités : le mouvement des nœuds ne doit pas être complètement stochastique [30] afin de pouvoir représenter le comportement des déplacements humains. De plus, le modèle de mobilité doit pouvoir prendre en compte les limitations de mouvements imposées par les infrastructures telles qu'elles existent dans le monde réel : routes,

30. Le résultat est un mouvement brownien qui correspond plutôt au mouvement erratique (des arrêts brutaux, des changements de direction à angles aigus) des entités mobiles dans la nature qu'au déplacement humain.

bâtiments, signaux de circulation, etc. Le but est de reproduire les mouvements limités des nœuds dans un espace spécifique réaliste tel qu'un espace urbain ;
- distribution des temps d'inter-contact connue : la distribution des temps d'inter-contact permet de caractériser la mobilité des nœuds et l'interaction de chaque paire entre elles. Elle doit donc être déterminée afin de faciliter le choix du modèle par rapport à un scénario spécifique ;
- flexibilité : un modèle de mobilité flexible permet de représenter différents types de mouvements en modifiant ses paramètres ;
- modèle validé : le modèle de mobilité doit être validé dans un contexte réel afin de montrer sa pertinence. Un modèle de mobilité est validé s'il génère des traces de mouvements qui possèdent certaines propriétés observées dans des scénarios réels. La validation du modèle peut être empirique ;
- simplicité d'utilisation : la simplicité d'utilisation est basée sur le nombre de paramètres utilisés et la complexité de leur génération. De plus, le rôle d'un paramètre intégré au modèle de mobilité doit être clairement déterminé. Il est également intéressant de savoir quel est l'impact du changement d'un paramètre sur les mouvements des nœuds. La mise en place du modèle, incluant sa description, son implantation, etc., doit être relativement simple afin que le modèle soit utilisable. Notons que la présence d'outils de génération de traces contribue à la simplicité d'utilisation.

Critères spécifiques aux réseaux en grappes

Dans nos travaux, nous nous intéressons aux MANETs formant des grappes très dynamiques : les arrivées et les départs de nœuds peuvent avoir lieu de façon arbitraire et à une fréquence élevée. Par ailleurs, dans ces réseaux opportunistes et spontanés, la mobilité des nœuds tend à être hétérogène [Hu and Dittmann, 2009]. Par conséquent, les critères spécifiques de notre domaine d'étude sont la dynamicité de groupe et l'hétérogénéité de la mobilité :

- dynamicité de groupe : le modèle de mobilité doit prendre en compte la dynamicité des MANETs. Pour évaluer le critère et la condition de stabilité de $\Diamond\mathcal{PPD}$, le modèle doit aussi garantir l'existence de grappes stables dans le temps ;
- mobilité hétérogène : le modèle doit pouvoir représenter l'hétérogénéité de la mobilité des nœuds, de l'espace et du temps. Chaque nœud possède son patron de mouvement. Durant une période de temps relativement courte, chaque nœud peut visiter un certain nombre d'endroits à l'intérieur d'une zone donnée plutôt que d'autres endroits à l'extérieur de cette zone. Enfin, chaque nœud peut répéter le même patron de mouvement périodiquement pendant une période de temps beaucoup plus longue. Cela permet de modéliser les mouvements des nœuds de façon réaliste dans des scénarios complexes et spécifiques. Par exemple, dans un scénario de secours dans une zone sinistrée, certaines unités de secours (médecins) peu mobiles restent dans une zone donnée pour soigner des patients. D'autres unités (ambulances) plus mobiles se déplacent d'une zone à une autre en transportant des blessés et en opérant des mouvements cycliques.

Maintenant que nous avons présenté nos critères de choix des modèles de mobilité, nous parcourons dans les sections qui suivent la littérature distinguant trois grandes classes de modèles : les modèles de mobilité individuelle, les modèles de mobilité de groupe, puis les modèles de mobilité pour les réseaux partitionnables.

6.2.2 Modèles de mobilité individuelle

Dans les modèles de mobilité individuelle, les mouvements des nœuds sont indépendants les uns des autres. Dans cette section, nous présentons les principaux modèles de mobilité individuelle proposés dans la littérature. Ce sont les principaux dans le sens où ce sont les plus utilisés dans les publications comprenant des évaluations par simulation d'algorithmes ou de protocoles répartis. Les modèles sont groupés et nous indiquons au fur et à mesure des paragraphes ceux que nous sélectionnons pour notre étude.

Random Walk, *Random Waypoint*, *Random Direction*, *Random Trip*

Le modèle de mobilité individuelle synthétique le plus simple est la marche aléatoire (en anglais, *Random Walk* ou RW) introduit par [Guerin, 1987]. C'est une version discrète du mouvement brownien qui a été proposé pour imiter le mouvement erratique des entités mobiles dans la nature. Un nœud mobile se déplace de sa position courante vers une nouvelle position en choisissant aléatoirement une vitesse et une direction. Les nouvelles vitesse et direction sont deux paramètres choisis uniformément et respectivement dans les intervalles $[V_{min}, V_{max}]$ et $[0, 2\pi]$. Chaque mouvement s'opère dans un intervalle de temps constant à la fin duquel une nouvelle direction et une nouvelle vitesse de déplacement sont calculées. De par son aspect simple et concis, le modèle de marche aléatoire est largement utilisé dans la littérature [Bar-noy and Kessler, 1994, Zonoozi and Dassanayake, 1997, Badarneh and Kadoch, 2009]. Cependant, le processus de déplacement est sans mémoire, c'est-à-dire que les nœuds ne conservent pas la connaissance de leurs positions et vitesses passées. Cette caractéristique « sans mémoire » peut générer des mouvements non réalistes tels que des arrêts brutaux et des changements de direction à angles aigus [Camp et al., 2002]. De plus, la distribution des temps d'inter-contacts observée suit une loi exponentielle qui est considérée non réaliste [Chaintreau et al., 2007]. Néanmoins, [Karagiannis et al., 2007] montre que, dans certains cas, le modèle produit une distribution qui suit la loi de puissance (telle qu'observée dans les traces réelles).

Le modèle *Random Waypoint* (RWP) est une variante du modèle RW qui ajoute des pauses entre les changements de direction et de vitesse des nœuds [Johnson and Maltz, 1996]. Le mouvement des nœuds est ainsi divisé en plusieurs époques de déplacement et de pause. C'est le modèle de base pour les simulations dans les MANETs. Ses caractéristiques ont été analysées dans plusieurs travaux : taux de changement de la topologie du réseau [Pérez-Costa et al., 2003], distribution stationnaire des nœuds [Bettstetter et al., 2004, Bettstetter et al., 2003], connectivité dans le réseau [Lassila et al., 2005], et évolution de la répartition des nœuds par le biais d'équations aux dérivées partielles [Garetto and Leonardi, 2006], etc. Cependant, l'existence d'un régime stationnaire, qui permet des études sur des valeurs moyennes de certains paramètres au cours du temps, ne peut pas être garantie dans tous les scénarios. De ce fait, [Navidi and Camp, 2004]

propose une version modifiée du modèle RWP pour obtenir un régime stationnaire dès le début de la simulation ; c'est le modèle *steady-state* RWP ou SSRPW. En outre, RWP souffre du problème de flot de densité : les nœuds ont tendance à se concentrer au centre de l'aire de simulation [Royer et al., 2001]. La raison de cette anomalie vient du fait que la probabilité qu'un nœud choisisse une nouvelle destination située au centre de l'aire de simulation, ou une destination qui requiert un déplacement à travers le centre, est élevée [Camp et al., 2002]. Une solution possible pour avoir une distribution spatiale uniforme est de considérer des formes particulières telles que des sphères ou des tores pour les aires de simulations. Cependant, ces abstractions géométriques sont non réalistes. Un autre problème du modèle RWP est que la vitesse moyenne de tous les nœuds à un instant donné ne correspond pas à la distribution de leur vitesse et tend à se dégrader au cours du temps. En particulier, [Yoon et al., 2003] montre que si $V_{min} = 0$, la vitesse moyenne des nœuds diminue au cours du temps et tend vers 0 (au lieu de $V_{max}/2$).

Le modèle *Random Direction* (RD) a été proposé par [Royer et al., 2001] afin de résoudre le problème de flot de densité présent dans les modèles RW et RWP. Au lieu de sélectionner un point de destination dans l'aire de simulation, un nœud mobile choisit une direction. La direction est mesurée en termes de degrés. Au début de la simulation, chaque nœud sélectionne une direction dans l'intervalle $[0, 359]$ et choisit une destination située sur la bordure en suivant cette direction. Comme dans le modèle RWP, le nœud se déplace vers ce point avec une vitesse sélectionnée aléatoirement et uniformément dans un intervalle donné. Une fois que le nœud atteint sa destination, il s'arrête pendant un certain temps et choisit une nouvelle direction limitée dans l'intervalle $[0, 180]$: la direction est limitée car le nœud est sur la bordure et ne la traverse pas. [Camp et al., 2002] observe que le nombre de sauts moyen pour acheminer les messages entre les nœuds est beaucoup plus élevé que dans d'autres modèles incluant le modèle RWP. De plus, le réseau se partitionne plus souvent. Les résultats obtenus dans [Royer et al., 2001] montrent également que le modèle RD génère beaucoup moins de fluctuations dans la distribution spatiale des nœuds. Ainsi, le problème de flot de densité est moins important. Cependant, les traces de mouvements générées ne semblent pas réalistes car dans le monde réel les individus ne se déplacent pas en traversant une aire (un bâtiment ou une ville). En outre, il est peu probable que les individus ne s'arrêtent que sur les bordures de l'aire.

Le modèle *Random Trip* (RT) a été proposé par [Le Boudec and Vojnovic, 2006]. Le modèle RT est un modèle de mobilité générique qui fournit des bases théoriques pour concevoir des modèles de mobilité avec des régimes stationnaires. Le modèle RT est défini par un ensemble de conditions de « stabilité ». Ces conditions permettent de garantir l'existence ou la non-existence d'un régime stationnaire de la mobilité des nœuds. Un modèle de mobilité peut être considéré comme un modèle RT s'il satisfait ces conditions. Les modèles RW et RWP appartiennent à cette catégorie.

Parmi les modèles de mobilité présentés ci-dessus, nous choisissons le modèle RWP pour notre étude. C'est le modèle de base le plus utilisé dans les simulations des MANETs. De plus, [Camp et al., 2002] montre que le modèle RWP peut générer des modèles de mobilité réalistes dans certains scénarios : mouvements des individus lors d'une conférence ou dans un musée. Pour pallier au problème du régime stationnaire, nous considérons également le modèle SSRWP.

Aire non bornée

Le modèle de mobilité avec aire non bornée (en anglais, *Boundless Simulation Area* ou BSA) a été proposé par [Haas, 1997]. Contrairement aux modèles précédents, le modèle BSA présente un processus de déplacement avec mémoire. Le mouvement de chaque nœud mobile est caractérisé par un vecteur vitesse $\vec{v} = (v, \theta)$, où v est la valeur de la vitesse du nœud et θ sa direction. Le vecteur vitesse $\vec{v}$ et la position du nœud (x, y) sont changés périodiquement tous les Δt secondes selon les formules suivantes :

$$\begin{aligned} v(t + \Delta t) &= min[max(v(t) + \Delta v, 0), V_{max}], \\ \theta(t + \Delta t) &= \theta(t) + \Delta\theta, \\ x(t + \Delta t) &= x(t) + v(t) \times \cos(\theta(t)), \\ y(t + \Delta t) &= y(t) + v(t) \times \sin(\theta(t)), \end{aligned}$$

où V_{max} est le vecteur vitesse maximal considéré dans la simulation et Δv est la différence de changement de vitesse. La valeur de Δv est uniformément distribuée dans l'intervalle $[A_{max} * \Delta t, A_{max} * \Delta t]$, avec A_{max} l'accélération/décélération maximale du nœud mobile. $\Delta\theta$ est la différence dans le changement de direction comprise entre $[-\alpha\Delta t, \alpha\Delta t]$, et α est le changement de l'angle de direction maximal par unité de temps.

Une autre différence par rapport aux modèles précédents est la gestion des bordures de l'aire de simulation. Les bordures sont reliées deux à deux et l'aire de simulation prend alors la forme d'un tore. Lorsqu'ils atteignent une bordure, ils continuent leur chemin et réapparaissent du côté opposé. Cette abstraction topologique de l'air de simulation permet d'atténuer le problème de flot de densité du modèle RWP. De plus, le modèle BSA génère des patrons de mouvements observables dans le monde réel [Camp et al., 2002]. Cependant, il existe des effets non désirables dus au fait que les nœuds sont autorisés à se déplacer tout au long du tore. Par exemple, un nœud statique (ou peu mobile) resté fixe à un point donné et un nœud mobile qui continue à se déplacer dans la même direction peuvent devenir périodiquement voisins. En outre, une aire de simulation sans limite de frontière peut forcer la modification du modèle de propagation radio afin de couvrir la portée de transmission d'un bord à un autre. Enfin, un tore est une abstraction géométrique non réaliste. Par conséquent, nous ne considérons pas ce modèle dans notre étude.

Gauss-Markov

Le modèle de mobilité *Gauss-Markov* (ou G-M) a été conçu pour les réseaux de services de communication personnels (en anglais, *Personal Communication Services* ou PCS) [Liang and Haas, 2003] afin de s'adapter à différents degrés d'entropie via un paramètre d'ajustement. Une implantation du modèle G-M a été présentée dans [Toley, 1999] pour l'étude des performances d'un protocole conçu pour les MANETs. Initialement, les nœuds sont répartis aléatoirement sur l'aire de simulation. Une vitesse et une direction de déplacement sont affectées à chaque nœud. Les valeurs de la vitesse v_n et de la direction d_n à la n^e itération sont déduites à partir de leur valeur à l'itération $n - 1$ selon les équations suivantes :

$$v_n = \alpha \times v_{n-1} + (1 - \alpha) \times \bar{v} + \sqrt{(1 - \alpha^2)} \times v_{x_{n-1}}$$

$$d_n = \alpha \times d_{n-1} + (1-\alpha) \times \bar{d} + \sqrt{(1-\alpha^2)} \times d_{x_{n-1}},$$

où $\bar{v}$ et $\bar{d}$ sont des constantes correspondant respectivement à la moyenne de la vitesse et la moyenne de la direction. $v_{x_{n-1}}$ et $d_{x_{n-1}}$ sont deux variables aléatoires issues d'une distribution gaussienne. α est le paramètre d'ajustement pour le degré d'entropie de la mobilité dont la valeur est comprise entre 0 et 1. Lorsque la valeur de α est proche de 0, un mouvement de type marche aléatoire est obtenu. Au contraire, les déplacements sont linéaires quand α est égal à 1.

À chaque itération, les coordonnées du nœud sont calculées à partir de la position, de la vitesse et de la direction courantes. Pour s'assurer que les nœuds ne restent pas trop longtemps près d'une bordure, ils doivent se déplacer dans une autre direction. Ainsi, contrairement au modèle BSA, les effets indésirables de bordure sont évités. De plus, grâce à la mémoire, les arrêts brutaux et les changements de direction à angle aigu sont limités. Par ailleurs, le modèle G-M est flexible et peut générer des patrons de mouvements réalistes. Par conséquent, nous le choisissons pour l'étude des performances de nos algorithmes.

Marche aléatoire probabiliste

[Chiang, 1998] propose une alternative au modèle de marche aléatoire en définissant un modèle de marche probabiliste contrôlée par une chaîne de Markov à trois états. Pour cela, une matrice de probabilité est utilisée pour déterminer la position d'un nœud donné à l'itération suivante. Cette matrice est représentée par trois états différents. L'état 0 représente la position courante d'un nœud donné. L'état 1 représente la position précédente du nœud. Et, l'état 2 représente la position suivante du nœud si le nœud continue à se déplacer dans la même direction. La matrice de probabilité de déplacement est la suivante :

$$P = \begin{bmatrix} p(0,0) & p(0,1) & p(0,2) \\ p(1,0) & p(1,1) & p(1,2) \\ p(2,0) & p(2,1) & p(2,2) \end{bmatrix}$$

Dans cette matrice, chaque élément $p(a,b)$ correspond à la probabilité qu'un nœud passe d'un état a à un état b. [Chiang, 1998] présente une implantation du modèle RWP permettant d'éviter des arrêts brutaux ou des changements de direction à angle aigu, et donc de produire des mouvements plus réalistes. Cependant, le choix des valeurs appropriées pour les éléments $p(a,b)$ peut s'avérer très délicat pour un scénario particulier. Par conséquent, nous ne choisissons pas ce modèle dans nos travaux.

Modèle basé sur les graphes

Pour prendre en compte la limitation des déplacements des nœuds imposée par l'infrastructure, une approche est de définir des chemins autorisés sur lesquels les nœuds peuvent se déplacer. [Tian et al., 2002] propose un modèle de mobilité basé sur un graphe. Les sommets du graphe représentent les destinations possibles et les arêtes correspondent aux chemins autorisés. Le graphe est supposé connexe. Initialement, chaque nœud est placé aléatoirement sur un sommet.

Le nœud choisit un sommet destination en adoptant des mouvements issus du modèle RWP. Arrivé à sa destination, le nœud effectue une pause et recommence le processus de déplacement. Ce modèle permet donc de modéliser une infrastructure telle qu'un espace urbain ou une ville : les arêtes représentent les rues ou les routes, et les sommets représentent des bâtiments ou des espaces de stationnement. La difficulté réside dans la définition d'un graphe réaliste et précis. N'ayant pas de scénario cible précis comme une ville donnée, nous n'utilisons pas ce modèle.

Random Waypoint City

Le modèle *Random Waypoint City* (RWPC) [Kraaier and Killat, 2005] simule les mouvements des véhicules dans un environnement urbain réel. Les informations sur le réseau routier incluent la limitation de vitesse, les carrefours, etc. L'aire de simulation est divisée en points. Le mouvement d'un nœud peut être décomposé en quatre étapes : *1)* choisir un point de destination selon la distribution spatiale initiale des destinations, et si le point choisi ne se trouve pas sur une route, remplacer ce point par l'endroit le plus proche se trouvant sur une route ; *2)* calculer la durée de voyage en utilisant le chemin le plus court ; *3)* déterminer le temps de pause à l'arrivée et planifier le mouvement suivant ; et *4)* mesurer la durée durant laquelle le nœud demeure sur chaque pixel dans le segment de la route utilisée.

Les résultats obtenus dans [Kraaier and Killat, 2005] montrent que, à cause des effets de bordure, le modèle RWPC présente le même problème de flot de densité que le modèle RWP. Les plans réalistes tels que ceux proposés par le projet libre OpenStreetMap (OSM) [OpenStreetMap Community, 2009] peuvent être exploités. Cependant, une telle approche peut être très longue et complexe à mettre œuvre. Mais, les services de localisation tels que OpenRouteService (ORS) [Neis and Zipf, 2009] sont une alternative plus souple et plus simple. Bien que ce modèle soit intéressant, nous ne l'utilisons pas par manque d'outil de génération de traces approprié.

Route aléatoire

Le modèle de route aléatoire (en anglais, *Random Street* ou RaSt) [Aschenbruck and Schwamborn, 2010] permet d'intégrer des cartes basées sur OSM dans l'outil de génération de traces BonnMotion [Aschenbruck et al., 2010] en utilisant le service OSR pour la génération des scénarios réalistes. Par ailleurs, les destinations sont uniformément distribuées sur l'aire de simulation. Ainsi, le modèle RaSt intégrant le service de localisation ORS dans l'outil de modélisation et d'analyse de scénarios, des traces synthétiques précises peuvent être générées. De plus, le modèle RaSt est facile à utiliser et à paramétrer. En outre, des améliorations sont envisagées pour modéliser les différents intervalles de vitesse selon les types de routes et les types de nœuds (ambulances, taxis, voitures familiales, etc.). Aussi, nous considérons ce modèle de mobilité dans notre étude.

Section d'une ville

[Davies, 2000] propose un modèle de mobilité qui simule les mouvements des nœuds dans une section d'une ville (en anglais, *City Section* ou CS). Les rues et les limitations de vitesse sont basées sur le type de ville à simuler. Par exemple, dans [Davies, 2000], l'aire de simulation est représentée par une grille qui symbolise les routes verticales et horizontales au sein d'une ville. Initialement, chaque nœud mobile est placé sur un point situé dans une rue. Le nœud choisit aléatoirement une destination qui est un point dans une autre rue. Le plus court chemin tient compte des règles de conduite : limitation de vitesse et distance minimale entre deux nœuds. Quand un nœud arrive à une destination, il s'arrête pour une période de temps donnée.

Dans le modèle CS, le comportement des nœuds mobiles est très restreint. Cela semble fournir des patrons de mouvements réalistes pour une section d'une ville [Camp et al., 2002, Davies, 2000]. En effet, les nœuds mobiles n'ont pas un comportement erratique. D'autres améliorations sont apportées comme les durées de pause sur certaines intersections ou la forte concentration de nœuds pendant les heures de pointe. Un élément important manquant dans ce modèle est l'utilisation de plans de ville réalistes. En outre, à notre connaissance, aucune implantation complète du modèle n'est disponible. Par conséquent, nous ne considérons pas ce modèle dans notre étude.

Manhattan

Similaire au modèle CS, le modèle Manhattan introduit par [Bai et al., 2003] émule les mouvement des nœuds dans un espace urbain. L'espace urbain est représenté par la grille de Manhattan composée de rues verticales et horizontales. À une intersection, la probabilité pour que le nœud reste sur la même rue est 50%, celle pour tourner à gauche 25% et celle pour tourner à droite 25%. La vitesse du nœud mobile dépend de la période précédente. De plus, la vitesse d'un nœud est limitée par celle du nœud qui le précède dans la même voie de circulation. À la différence du modèle CS et des autres modèles, les nœuds n'ont pas de destination à atteindre. Par conséquent, les nœuds peuvent errer sur toute la surface de l'aire de simulation avant d'atteindre une bordure. Notons qu'un nœud qui est sur le point de sortir de la zone de simulation peut être enlevé de la simulation. Le modèle Manhattan a l'avantage d'être très simple à utiliser et peut simuler les mouvements des nœuds dans une section d'une ville dont les routes sont perpendiculaires. Nous le considérons dans notre étude.

Conclusion sur les modèles de mobilité individuelle

Un modèle de mobilité individuelle simule le mouvement d'un nœud mobile indépendamment des mouvements des autres nœuds. Les nœuds peuvent se regrouper dans un espace donné, ce qui génère un effet de groupe. Cependant, ce regroupement des nœuds est généré par les déplacements aléatoires des nœuds dans une zone de taille limitée. Il n'existe pas de lien direct entre les déplacements des nœuds et la formation des grappes de nœuds.

6.2.3 Modèles de mobilité de groupe

Dans les MANETs, il existe plusieurs situations dans lesquelles il est nécessaire de modéliser le comportement de mobilité de nœuds qui se déplacent en groupes. Par exemple, un groupe de sapeurs-pompiers est affecté à la recherche de victimes suite à une catastrophe naturelle, ou un groupe de personnes travaillent d'une manière coopérative, etc. Dans cette section, nous présentons certains des modèles de mobilité de groupe proposés dans la littérature. Les modèles sont groupés et nous indiquons au fur et à mesure des paragraphes ceux que nous sélectionnons pour notre étude.

Colonne, communauté nomade, et poursuite

Dans [López and Manzoni, 2001], le mouvement de type brownien est considéré comme le modèle de mobilité de base à partir duquel plusieurs modèles de mobilité de groupe sont proposés. Cela inclut les modèles colonne, communauté et poursuite. L'approche adoptée est motivée par la simplicité de l'implantation du patron de mouvement de type brownien : la nouvelle position d'un nœud est calculée à partir de son ancienne position et d'un vecteur de déplacement aléatoire.

Le modèle de mobilité colonne simule le mouvement d'un groupe de nœuds qui se déplacent ensemble, toujours vers l'avant, d'un endroit vers un autre en formant une colonne. Pour cela, une grille initiale formant une colonne est définie pour tracer la trajectoire des nœuds. Chaque nœud est placé par rapport à son point de référence situé sur la colonne de référence et est autorisé à se déplacer aléatoirement autour de son point de référence (utilisant par exemple le modèle de marche aléatoire). La distance maximale qui sépare les nœuds de leur point de référence est déterminée par un paramètre. Ce modèle de mobilité simule des scénarios spécifiques comme une colonne de robots qui balayent une zone [López and Manzoni, 2001]. Une modification légère du modèle permet aux nœuds de se suivre les uns les autres.

Le modèle communauté nomade simule le mouvement des nœuds qui se déplacent collectivement d'un point vers un autre à la manière des individus dans les sociétés nomades. Dans chaque communauté, les nœuds maintiennent leur espace personnel dans lequel ils se déplacent de façon aléatoire. Chaque nœud se déplace autour d'un point de référence suivant un modèle de mobilité donné tel que le modèle RW. Le point de référence suit lui-même un mouvement aléatoire. Comme dans le modèle colonne, un nœud mobile ne peut pas se déplacer loin de son point de référence. Cependant, [Camp et al., 2002] observe que les mouvements des nœuds sont moins limités dans le modèle communauté nomade que dans le modèle colonne.

Le modèle poursuite cible des scénarios de poursuite. Les nœuds poursuivent le point de référence qui joue le rôle particulier de la cible. Tous les nœuds essayent d'atteindre la cible en adoptant un mouvement avec un certain degré d'aléa, qui est limité afin de maintenir l'efficacité de la poursuite.

Dans les modèles colonne, communauté nomade et poursuite, un nœud ne peut ni quitter ni rejoindre un groupe. Ces modèles possèdent donc une structure de groupe statique et sont spécifiques à des scénarios particuliers tels que le déplacement de groupes de touristes dans un musée, les courses poursuites, ou encore le mouvement de robots. Dans notre étude, nous ne ciblons pas un scénario particulier. De ce fait, nous ne considérons aucun des trois modèles

colonne, communauté nomade et poursuite dans notre étude.

Reference Point Group Mobility et ses variantes

Dans le modèle de mobilité *Reference Point Group* ou RPGM [Hong et al., 1999], chaque groupe possède un centre logique qui définit le mouvement du groupe entier. Chaque membre du groupe possède un point de référence qui est toujours défini par rapport au centre du groupe. Toutefois, les nœuds possèdent un certain degré d'entropie vis-à-vis de leur point de référence. La trajectoire du centre logique d'un groupe est spécifiée manuellement. Cependant, il est possible d'avoir d'autres scénarios dans lesquels le mouvement de centre du groupe est gouverné par un autre modèle de mobilité. Dans le modèle RPGM, les groupes sont dynamiques : quand un nœud entre dans l'aire d'un autre groupe (déterminée par le point de ce groupe référence), il change de groupe pour adopter ce dernier avec une probabilité c. La valeur de c est paramétrable. Quand la valeur de c vaut zéro, les groupes deviennent statiques.

Il est possible d'obtenir d'autres types de modèles de mobilité de groupe à partir du modèle RPGM. [Camp et al., 2002] montre qu'il est possible d'obtenir des modèles colonne, communauté nomade et poursuite.

[Wang and Baochun, 2002] propose une variante appelée *Reference Velocity* ou RVGM dans laquelle le comportement du groupe est caractérisé par un vecteur vitesse au lieu de la proximité des emplacements des nœuds : les nœuds membres d'un groupe se déplacent avec des vitesses proches de celle du groupe.

Une autre extension du modèle RPGM est proposée par [Blakely and Lowekamp, 2004] qui définit le modèle de groupe structuré (en anglais, *structured group mobility* ou SGM) dans lequel le point de référence c_l d'un groupe est représenté par le centre géographique du groupe ou la position d'un nœud leader. La position d'un nœud est calculée à partir de sa distance et de sa relation angulaire par rapport à c_l. La distance et l'angle sont choisis respectivement à partir de deux distributions D et A données. Le mouvement du groupe peut être gouverné par un modèle de mobilité appliqué au centre ou via un script qui définit des points de cheminement spécifiques établissant des chemins prédéfinis à suivre. La difficulté d'utilisation de ce modèle réside dans la modélisation de façon réaliste de la structure des groupes via les distributions A et D.

Parmi les modèles RPGM, RVPM et SM, nous considérons seulement le modèle RPGM car c'est l'un des modèles de mobilité de groupe les plus utilisés dans les études des MANETs [Davies, 2000, Kurkowski et al., 2005]. De plus, il est possible d'avoir des groupes dynamiques. À la manière du modèle RWP, le modèle RPGM nous offre une première approche de mobilité, mais de groupe, grâce à son aspect générique et simple d'utilisation. Les modifications apportées dans ses extensions ne sont pas justifiées pour notre étude.

Zone sinistrée, Mission critique mobile, et Composite

Le modèle zone sinistrée fournit un modèle de mobilité de groupe complexe simulant les mouvements des groupes de nœuds dans une zone sinistrée [Aschenbruck et al., 2007]. L'aire de simulation est divisée en plusieurs sous-aires tactiques disjointes. Les aires tactiques sont classifiées selon les concepts de la protection civile. Un nœud peut être de type stationnaire ou

transport. Un nœud stationnaire peut se déplacer selon un modèle de mobilité, par exemple le modèle de marche aléatoire, dans l'aire qui lui a été affectée. Un nœud transport peut transporter des patients entre les aires tactiques en produisant des mouvements cycliques. Plusieurs classes de nœuds peuvent être considérées : il est possible de distinguer les piétons des véhicules en définissant leurs vitesses minimale et maximale. Le mouvement de chaque groupe s'opère sur des chemins optimaux en évitant les obstacles. Chaque chemin optimal est déterminé par les méthodes dites de planification de mouvements pour robots. Ces méthodes permettent de trouver les chemins les plus courts en évitant les obstacles entre les aires tactiques. Comme dans le modèle RPGM, un nœud peut changer de groupe avec une certaine probabilité.

Le modèle mission critique mobile, comme le modèle zone sinistrée, tente de simuler les mouvements de nœuds dans une zone sinistrée en combinant plusieurs modèles existants [Papageorgiou et al., 2009]. Ce modèle est similaire au modèle zone sinistrée concernant la gestion des obstacles et le mouvement des nœuds mobiles individuels. Le modèle RPGM est utilisé pour la mobilité de groupe. Cependant, le modèle n'inclut pas la division de l'aire de simulation en sous-aires spécifiques.

[Pomportes et al., 2011] propose un modèle de mobilité composite composé de plusieurs modèles de mobilité existants afin de simuler la mobilité des nœuds dans des scénarios de catastrophe. Comme dans le modèle mission critique, le modèle RPGM est utilisé pour simuler la mobilité de groupe et le modèle RWP est remplacé par le modèle de marche de Levy [Rhee et al., 2011] considéré plus réaliste par [Pomportes et al., 2011] car il possède une distribution des temps inter-contacts suivant une loi de puissance.

Les modèles de mobilité de groupe zone sinistrée, mission critique mobile, et composite permettent de générer des scénarios similaires avec la possibilité d'avoir des groupes dynamiques. Parmi ces trois modèles, nous considérons le modèle zone sinistrée car il est plus réaliste que le modèle mission critique mobile et plus facile à utiliser que le modèle composite.

Circuit virtuel

[Zhou et al., 2004] propose le modèle de mobilité circuit virtuel qui tente de capturer la mobilité de groupes dynamiques. Il est composé de « stations d'échange » interconnectés par des « circuit virtuels ». Les stations d'échange sont déployées sur l'aire de simulation et sont reliées par des circuits virtuels. L'emplacement de ces stations peut être spécifié par l'utilisateur ou choisi aléatoirement. Les nœuds mobiles sont contraints à se déplacer sur les circuits virtuels. Les nœuds d'un même groupe possèdent le même patron de mouvement. Chaque groupe est défini par une valeur seuil de stabilité. Arrivé à une station d'échange, un nœud dans un groupe vérifie si sa valeur de stabilité est au delà du seuil. Si c'est le cas, ce nœud choisit un circuit différent de celui de son groupe. Quand plusieurs groupes arrivent en même temps sur une station d'échange et s'ils choisissent ensuite le même circuit, alors ils fusionnent. Plusieurs classes de nœuds mobiles (piétons, voitures, etc.) peuvent être définies. Seuls les nœuds appartenant à la même classe peuvent fusionner dans un même groupe. Il est aussi possible de modéliser les mouvements des nœuds individuels (qui n'appartiennent à aucun groupe) ainsi que les nœuds statiques tels que les capteurs.

Ce modèle cible les scénarios militaires plutôt que les environnements urbains. En effet, il est peu probable que les nœuds mobiles (personnes, voitures, etc.) se déplacent en groupes sur les routes. Les divisions ou les fusions ne se produisent qu'aux stations d'échange. En outre, [Zhou et al., 2004] n'explique pas comment choisir la valeur seuil de stabilité approprié pour un scénario particulier. Par ailleurs, comme tout modèle basé sur un graphe, la difficulté réside dans la définition d'un graphe réaliste et précis. Pour ces raisons, nous ne choisissons pas ce modèle.

Conclusion sur les modèles de mobilité de groupe

Les modèles de mobilité de groupe permettent de générer les scénarios de mobilité dans lesquels le comportement du mouvement d'un nœud peut dépendre de celui d'un ou plusieurs autres nœuds. Les nœuds qui appartiennent au même groupe adoptent le même comportement tandis que les nœuds dans différents groupes suivent différents patrons de mouvement. Un groupe de nœuds est dynamique si chaque nœud est autorisé à quitter et entrer dans le groupe, sinon il est statique. Les modèles de mobilité de groupe dynamique que nous avons étudié sont les modèles RPGM et ses variantes, zone sinistrée, mission critique mobile, composite, circuit virtuel. Les modèles colonne, communauté nomade et poursuite sont des modèles de mobilité de groupe statiques car ils n'autorisent pas un nœud à quitter ou entrer dans un groupe durant la simulation.

6.2.4 Modèles de mobilité pour les réseaux en grappes

Nous allons présenter dans cette section plusieurs modèles de mobilité qui tentent de capturer la formation dynamique de grappes de nœuds, qui sont des phénomènes observés dans les réseaux partitionnables.

Mobilité de grappe

[Lim et al., 2006] propose le modèle mobilité de grappe de nœuds (en anglais, *Clustered Mobility* ou CM) dans les réseaux sans fil à communication multisaut. L'aire de simulation est logiquement divisée en plusieurs zones $s_1, s_2, \ldots s_i, \ldots, s_{t-1}, s_t$, où t est le nombre total de zones. Plus une zone est peuplée, plus la probabilité qu'un nœud soit attirée par cet zone est élevée. Les zones les plus peuplées sont dites populaires. Durant la phase de croissance incrémentale qui correspond à la phase d'initialisation, la probabilité qu'un nœud soit affecté à une zone donnée augmente ou diminue en fonction du nombre de nœuds présents dans cette zone. La probabilité Φ_i pour qu'un nœud soit affecté à une zone s_i contenant k_i nœuds est la suivante :

$$\Phi_i = \frac{(k_i+1)^\alpha}{\sum_{j=1}^{t}(k_j+1)^\alpha},$$

où α est le coefficient de grappe qui sert à contrôler le degré de concentration des nœuds. La phase d'initialisation se termine lorsque tous les nœuds sont affectés à différents zones. Durant la phase d'attachement préférentiel, les temps de pause et les vitesses de déplacement des nœuds sont fournis par le modèle RWP_WRAP, qui est une variante du modèle RWP. Le point de

cheminement d'un nœud est choisi en sélectionnant d'abord une zone (choix aussi effectué avec Φ_i) et ensuite une position dans cette zone.

Ainsi, le modèle CM fournit une distribution spatiale non homogène des nœuds. Il s'ensuit la formation d'îlots de connectivité dans les zones populaires. Cependant, il n'est pas clairement établi que ces îlots restent stables. Par conséquent, nous ne choisissons pas ce modèle dans notre étude.

Mobilité basée sur la communauté

[Spyropoulos et al., 2006] propose le modèle de mobilité basé sur la communauté (en anglais, *community-based mobility* ou CBM) qui simule la mobilité non homogène et non aléatoire de nœuds dans des réseaux peu denses dans lesquels la plupart des nœuds sont isolés et quelques nœuds forment des grappes. Les auteurs constatent que dans le monde réel certains nœuds sont plus mobiles que d'autres ou peuvent visiter certains endroits plus souvent que d'autres. Dans ce modèle, le mouvement de chaque nœud alterne entre les états « local » et « errant ». Chaque nœud possède une communauté locale. Dans l'état local, le nœud possède un mouvement avec une direction aléatoire à l'intérieur de sa communauté locale. Dans l'état errant, le nœud se déplace aléatoirement dans l'ensemble de l'aire de simulation. Dans l'état local (respectivement errant), la probabilité de passer dans l'état errant (respectivement local) est $1-p_l$ (respectivement $1-p_r$). Ainsi, certains ajustements des paramètres amènent à la formation d'îlots de connectivité, par exemple, en synchronisant les mouvements d'un même groupe de nœuds de telle sorte qu'il y ait suffisamment de nœuds dans chaque grappe. [Spyropoulos et al., 2006] montre que les temps d'inter-contact entre les nœuds avec le modèle CBM sont plus réalistes que ceux obtenus avec la plupart des modèles tels que les modèles RWP, RD et RW. Par conséquent, bien que le choix des paramètres reste délicat et le modèle est intrinsèquement complexe, nous choisissons aussi ce modèle pour notre étude.

Mobilité basée sur la théorie des réseaux sociaux

[Musolesi and Mascolo, 2007] propose le modèle de mobilité basé sur la théorie des réseaux sociaux. Un des paramètres du modèle est le réseau social d'un ensemble d'individus qui transportent des appareils mobiles. Notons que, selon [Musolesi and Mascolo, 2007], le réseau social peut être généré synthétiquement. Le but est de prédire le mouvement des appareils qui est basé sur les décisions de ces individus et les relations sociales qui existent entre eux. La conception du modèle est organisée en trois étapes :

Modélisation des relations relations sociales. Les réseaux sociaux qui sont des paramètres du modèle sont représentés par des graphes pondérés. Chaque somme (ou nœud) du graphe représente une personne. Le poids associé à chaque arête du graphe est utilisé pour modéliser la force d'interaction entre les individus qui peut être exprimée, par exemple, par une mesure de probabilité que les individus se trouvent dans une même zone géographique. Le degré d'interaction sociale entre deux individus est une valeur comprise dans l'intervalle $[0, 1]$. La valeur 0 indique qu'il n'y a pas d'interaction tandis que la valeur 1 indique une interaction sociale forte entre eux. L'ensemble des degrés d'interaction entre chaque paire d'individus est

représenté par une matrice **M**, où $m_{i,j}$ représente le degré d'interaction entre les individus i et j. La matrice **M** est symétrique car les degrés d'interactions sont considérés comme symétriques [31]. Elle est utilisée pour générer une matrice de connectivité **C**, où $c_{i,j}$ vaut 1 si la valeur de $m_{i,j}$ est plus grande qu'un certain seuil (dans [Musolesi and Mascolo, 2007], le seuil est fixé à $0,25$). Intuitivement, le seuil exprime le fait qu'une relation sociale existe entre deux individus si elle est « assez importante ». La matrice d'interaction, et donc la matrice de connectivité, peut être dérivée des données réelles disponibles, par exemple, en faisant une enquête sociale. Il existe des modèles qui permettent de générer synthétiquement des réseaux sociaux.

Détection des structures de communauté. Le scénario de simulation est établi en affectant des groupes (ou communautés) de nœuds dans des zones différentes de l'aire de simulation. Les communautés sont déterminées par un algorithme basé sur le calcul de voisinage des arêtes. Intuitivement, l'algorithme est exécuté un certaine nombre de fois pour détecter les différentes communautés. La condition d'arrêt de l'exécution de l'algorithme est fournie par un autre mécanisme. Les résultats obtenus dans [Musolesi and Mascolo, 2007] montrent que les individus ayant des relations sociales fortes entre eux sont géographiquement proches fréquemment ou de temps en temps. L'aire de simulation est divisée en plusieurs zones (qui sont représentées par des surfaces carrées dans [Musolesi and Mascolo, 2007]). Chaque communauté identifiée est associée à un carré.

Mouvement des nœuds. Une fois que les nœuds sont placés sur la grille, ils peuvent se déplacer selon des lois d'attraction basées sur les relations sociales qui sont présentées comme suit. Le but est de placer un point dans un carré qui représente la destination finale d'un nœud. Le mouvement aléatoire de type RWP peut être adopté par un nœud. En revanche, la sélection des destinations est moins aléatoire. Quand un nœud atteint sa destination, une nouvelle destination est choisie selon le mécanisme suivant. Un certain nombre de nœuds (0 ou plusieurs) sont associés à chaque carré $S_{p,q}$ à l'instant t, où p et q représentent respectivement la position du carré en ligne et en colonne. En fonction des relations sociales, chaque carré est plus ou moins attractif. L'attractivité sociale d'un carré est une mesure de son importance en termes de relations sociales. L'importance sociale est calculée en évaluant les relations entre les nœuds qui se déplacent vers ce carré. L'attractivité sociale qu'un carré $S_{p,q}$ exerce sur un nœud i, $SA^i_{p,q}$, est présentée comme suit :

$$SA^i_{p,q} = \frac{\sum_{j \in C_{S_{p,q}}} m_{i,j}}{w},$$

où $C_{S_{p,q}}$ est l'ensemble des nœuds qui sont associés au carré $S_{p,q}$ et w est la cardinalité de $C_{S_{p,q}}$. Dans le cas où $w = 0$ (aucun nœud n'est associé au carré), la valeur de $C_{S_{p,q}}$ vaut 0.

Notons qu'il existe deux autres mécanismes pour la sélection des destinations : un mécanisme de sélection déterministe basé sur l'attractivité sociale avec la plus grande valeur et un mécanisme probabiliste dont la sélection d'une nouvelle destination est proportionnelle à l'attractivité de chaque carré. En outre, le modèle de [Musolesi and Mascolo, 2007] permet de prendre en compte différents types de relations durant une période de temps (une journée, une semaines, etc.). Par

31. Étant donné que les relations sociales sont exprimées par une seule mesure.

exemple, il peut être intéressant de pouvoir décrire que dans le matin et l'après-midi des jours de la semaine, les relations aux lieux du travail sont plus importantes que celles en famille. En revanche, c'est l'inverse pour les soirs et les week-ends.

Le modèle de mobilité basé sur la théorie des réseaux sociaux est plus réaliste que les autres modèles présentés dans le sens où le mouvement des nœuds est régi selon les relations qui existent entre les individus. En outre, les résultats obtenus dans [Musolesi and Mascolo, 2007] montrent une approximation de la loi de puissance pour le temps d'inter-contact entre les nœuds. En revanche, ce modèle est complexe et onéreux à mettre œuvre, par exemple, si nous devons faire une enquête sociale afin d'établir un scénario de mobilité des individus avec des relations sociales. Il existe des mécanismes et modèles qui permettent de générer synthétiquement des réseaux sociaux. Dans ce cas, les résultats obtenus sont moins précis. Toutefois, l'implantation du modèle reste relativement complexe et il n'existe pas d'outil de génération de scénarios pour ce modèle. Par conséquent, ce modèle n'est pas considéré dans notre étude.

Marche aléatoire hétérogène

Le modèle de mobilité marche aléatoire hétérogène (en anglais, *heterogeneous random walk* ou HRW) proposé par [Piórkowski et al., 2008] permet de simuler la mobilité hétérogène des nœuds spécifiquement dans les MANETs. L'aire de simulation est gérée comme un tore de côté $L = 1$. Le tore est décomposé en deux types d'aires C et $\bar{C}$. Pour cela, M cercles de rayon R dont les centres sont indépendamment et uniformément répartis sur l'aire de simulation, sont dessinés. C correspond à l'aire des cercles et $\bar{C}$ correspond à l'aire en dehors des cercles. Les nœuds mobiles effectuent des mouvements aléatoires avec des vitesses hétérogènes. La vitesse d'un nœud dans $\bar{C}$ est plus élevée que celle dans C. Par conséquent, la densité des nœuds dans C est plus élevée que celle dans $\bar{C}$. En choisissant de façon appropriée les paramètres du modèle, la distribution spatiale hétérogène conduit à des formations de grappes de nœuds stables dans le temps et dans l'espace. Un algorithme est également proposé qui initialise la mobilité des nœuds de telle sorte que la distribution spatiale des nœuds suit un régime stationnaire dès le début de la simulation. Cependant, la faiblesse principale de ce modèle est de considérer l'aire de simulation comme un tore, donc une abstraction géographique non réaliste. De plus, il est difficile à mettre en œuvre et il n'existe pas d'outil de génération de scénarios pour ce modèle. Par conséquent, nous ne choisissons pas ce modèle dans notre étude.

Conclusion sur les modèles de mobilité pour les réseaux en grappes

Les modèles de mobilité pour les réseaux en grappes tentent de détecter l'émergence de groupes de nœuds stables dans le réseau qui forment des îlots de connectivité. Pour cela, typiquement, l'aire de simulation est divisée en plusieurs zones. Une zone peut être plus ou moins populaire (ou attractive) vis-à-vis d'un nœud. Il existe différents mécanismes pour déterminer la popularité d'une zone : attachement préférentiel [Lim et al., 2006], purement probabiliste [Spyropoulos et al., 2006], basé sur les réseaux sociaux [Musolesi and Mascolo, 2007] et basé sur la division explicite de l'aire de simulation en deux zones populaires et non populaires [Piórkowski et al., 2008]. Les quatre modèles présentés sont analysés. Le plus réaliste

d'entre eux est le modèle de mobilité basée sur les réseaux sociaux car le mouvement des nœuds est régi selon les relations qui existent entre les individus. Mais, c'est aussi le modèle le plus complexe à mettre œuvre. La faiblesse principale de tous ces modèles est le manque d'outil de génération et d'analyse de scénarios. Nous choisissons le modèle CBM parce qu'il ne considère pas un tore comme aire de simulation comme dans le modèle HRW. Il est plus facile à mettre en œuvre que le modèle basé sur les réseaux sociaux et présente les temps d'inter-contact entre les nœuds plus réalistes que ceux obtenus avec la plupart des modèles tels que les modèles RWP, RD et RW.

6.2.5 Outils de génération et d'analyse de scénarios

Nous avons vu qu'il existe beaucoup de modèles de mobilité proposés dans la littérature. Nous nous somme basés sur des critères d'évaluation généraux puis spécifiques à notre domaine pour étudier certains d'entre eux. Il s'avère que beaucoup de ces modèles ne sont pas implantés ou sont complexes et difficiles à mettre en œuvre. Pour diminuer en partie ces difficultés, des outils de génération et d'analyse de scénarios sont proposés. Nous avons choisi l'outil BonnMotion car il est plus simple à utiliser et couvre beaucoup de modèles de la littérature [Aschenbruck et al., 2010]. En outre, les scénarios générés par BonnMotion peuvent être utilisés de façons native ou exportés vers certains simulateurs MANETs comme OMNeT++.

Bonnmotion

BonnMotion [Aschenbruck et al., 2010] est un outil libre qui permet la génération et l'analyse de scénarios de mobilité. Il a été développé à l'Université de Bonn dans le but d'étudier les caractéristiques de la mobilité des nœuds dans les réseaux mobiles à communication multisaut.

Actuellement, il existe plusieurs modèles de mobilité intégrés dans BonnMotion version 2.0. Nous retrouvons les modèles de mobilité individuelle tels que RW, RWP et communauté nomade, mais aussi les modèles de mobilité de groupe colonne, zone sinistrée et RPGM. Tous les modèles disponibles dans BonnMotion sont présentés dans le tableau 6.1. Notons que le modèle *ChainScenario* n'est pas lui-même un modèle de mobilité. Mais il permet de concaténer plusieurs modèles afin de créer des scénarios spécifiques. Par exemple, le modèle RWP peut être concaténé avec le modèle Manhattan pour créer un scénario dans lequel les étudiants se déplacent (selon le modèle RW) de leur domicile au campus (dans une grille de Manhattan).

Application	Scénarios de mobilité créés	Référence
ChainScenario	en concaténant plusieurs modèles	[BonnMotion, 2011]
Column	modèle colonne	[López and Manzoni, 2001]
DisasterArea	modèle zone sinistrée	[Aschenbruck et al., 2007]
GaussMarkov	modèle Gauss-Markov	[Camp et al., 2002]
ManhattanGrid	modèle grille de Manhattan	[Bai et al., 2003]
Nomadic	modèle communauté nomade	[López and Manzoni, 2001]
OriginalGaussMarkov	modèle Gauss-Markov	[Liang and Haas, 2003]
ProbRandomWalk	modèle marche aléatoire probabiliste	[Chiang, 1998]
Pursue	modèle poursuite	[Chiang, 1998]
RandomDirection	modèle direction aléatoire	[Royer et al., 2001]
RandomStreet	modèle route aléatoire	[Aschenbruck and Schwamborn, 2010]
RandomWalk	modèle marche aléatoire	[Guerin, 1987]
RandomWaypoint	modèle RWP	[Johnson and Maltz, 1996]
RandomWaypoint3D	modèle RWP avec aire en 3D	[Johnson and Maltz, 1996, BonnMotion, 2011]
RPGM	modèle RPGM	[Hong et al., 1999]
SLAW	modèle *Self-similar Least Action Walk model to construct mobility*	[Lee et al., 2009]
Static	sans mobilité des nœuds	[BonnMotion, 2011]
StaticDrift	sans mobilité des nœuds mais avec sélection des positions	[BonnMotion, 2011]
SteadyStateRandomWaypoint	modèle RWP avec régime stationnaire	[Navidi et al., 2004]

TABLE 6.1 – Modèles de mobilité implémentée dans BonnMotion (version 2.0).

6.2.6 Synthèse

Cette section synthétise l'ensemble des modèles de mobilité que nous avons étudiés. Le tableau 6.2 récapitule l'étude des modèles de mobilité individuelle et de groupe étudiés. Les colonnes correspondent aux critères d'évaluation des modèles de mobilité. Un modèle de mobilité idéal devrait posséder toutes ces caractéristiques. Nous utilisons les notations suivantes :

- un signe « + » indique que des éléments notables sont proposés dans le modèle ;
- un signe « # » indique que certains éléments sont présents mais de manière insuffisante ;
- un signe « − » indique qu'aucun élément n'est rempli, ou que des éléments ne sont pas pris en compte dans le modèle ;
- une case reste vide lorsque le modèle ne considère pas le critère.

Parmi les modèles de mobilité individuelle, nous avons choisi les modèles RWP (*Random Waypoint*), Steady-State RWP (*Steady-State Random Waypoint*), G-M (Gauss-Markov), Manhattan et Route aléatoire (*Random Street*). Le modèle RWP est le plus utilisé et analysé. Il présente une distribution exponentielle pour le temps d'inter-contact. Il est flexible et simple à utiliser. Le modèle Steady-State RWP permet en plus d'éviter le problème du régime stationnaire présent dans RWP. Le modèle G-M est plus réaliste que les modèles RWP et Steady-State RWP dans le sens où il s'agit d'un modèle avec mémoire. Cela permet aux nœuds d'éviter des changements de direction à angle aigu. En revanche, il est moins analysé et moins flexible que les deux précédents. Un seul paramètre d'ajustement est utilisé afin d'adapter le modèle à différents degrés d'entropie. Les modèles Random Street et Manhattan ont été choisis pour leur possibilité de limitation des mouvements qui permet de simuler le mouvement des nœuds de façon plus réaliste sur les routes d'une section d'une ville. Dans le modèle Manhattan, les routes sont matérialisées par des lignes perpendiculaires représentant la grille de Manhattan. Dans le modèle Random Street, les routes et la carte d'une section d'une ville sont obtenues à partir des données réelles via des services OSM et ORS. Enfin, l'outil de génération de traces BonnMotion permet de faciliter encore plus l'utilisation de ces modèles de mobilité dans notre étude.

Pour les modèles de mobilité de groupe avec groupes dynamiques, nous avons choisi les modèles RPGM (*Reference Point Group Mobility*) et zone sinistrée (*Disaster Area*). Le modèle zone sinistrée est le plus réaliste d'entre eux et peut générer des scénarios avec mobilité hétérogène, c'est-à-dire que différents nœuds peuvent posséder différents patrons de mouvement. De plus, il est flexible. Enfin, son utilisation est facilitée grâce à l'outil BonnMotion. Bien qu'il soit moins réaliste que les modèles zone sinistrée et circuit virtuel, nous retenons le modèle RPGM, car il est générique et flexible, et est couvert par BonnMotion.

Concernant les modèles de mobilité pour les réseaux en grappes *Clustered*, *Community-Based*, *Social Networsk-Based* et *Hetegenerous Random Walk*, bien qu'ils soient intéressants pour étudier des phénomènes d'émergence de grappes de nœuds, ils sont complexes et/ou onéreux à mettre œuvre. Nous choisissons le modèle Community-Based car il n'est pas le plus complexe, mais l'un des plus utilisés et présente les temps inter-contacts entres les nœuds suivant une loi de puissance.

Critère / Modèle de mobilité	Analysé	Limitation des mouvements	Mobilité hétérogène	Flexible	Mouvement de groupe	Dynamicité de groupe	Distrib. tps d'inter-contact connue	Validé	Simplicité d'utilisation	Modèle choisi pour notre étude
Random Walk	+			+			exp.		+ (bm)	
Random Waypoint	+			+			exp.	+	+ (bm)	∗
Steady-State Random Waypoint	+			+			exp.	+	+ (bm)	∗
Random Direction	+			+				+	+ (bm)	
Random Trip			+						#	
Boundless Simulation Areamodule de système	+			#					+ (bm)	
Gauss-Markov	#	−	−	#					+ (bm)	∗
Probabilistic Random Walk		#		#					+ (bm)	
Graph-Based		+							#	
Random Waypoint City		+	#						#	
Random Street		+	−						+ (bm)	∗
City Section		+							#	
Manhattan		+							+ (bm)	∗
Column		#			+				+ (bm)	
Nomadic Community		#			+				+ (bm)	
Pursue		#			+				+ (bm)	
Reference Point Group Mobility				+	+	+			+ (bm)	∗
Reference Velocity Group Mobility				+	+	+			#	
Disaster Area		+	+	#	+	+			+ (bm)	∗
Mission critical mobile		+	+	#	+	+			−	
Composite model		+	+	#	+	+			− (bm)	
Virtual Track	#	−	−		+	+			#	
Clustered	#		+			#	exp.	+	#	
Community-Based	#		+			#	puiss.	+	#	∗
Social Networks-Based	+	#	#	−		+	puiss.	+	−	
Hetegenerous Random Walk	+					+		+	#	

TABLE 6.2 – Tableau récapitulatif des modèles de mobilité. La mention « (bm) » indique que le modèle est implanté dans BonnMotion, contribuant ainsi à la facilité de son utilisation. Les termes « exp. » et « puiss. » désignent respectivement les lois exponentielle et puissance. Enfin, le caractère « ∗ » indique que le modèle est considéré dans notre étude.

6.3 Simulateurs utilisés pour les réseaux mobiles spontanés

La simulation d'un système comprend la modélisation de celui-ci afin de prévoir son comportement dans le monde réel. La simulation peut être de nature continue ou discrète [MacDougall, 1987]. La *simulation continue* modélise le système par une représentation dans laquelle les variables d'état évoluent en fonction du temps. Elle permet d'établir un modèle analytique du système étudié. Cependant, à cause de la complexité intrinsèque des MANETs, les modèles analytiques sont très difficiles à mettre en œuvre. La *simulation discrète* modélise le système par une représentation dans laquelle les changements des variables d'état du système sont réalisés à des instants discrets. Dans la littérature, l'approche par simulation discrète est reconnue comme une alternative viable pour étudier les systèmes basés sur les MANETs [Heidemann et al., 2001, Hogie et al., 2006]. Par conséquent, nous suivons l'approche par simulation discrète. Plus précisément, nous étudions dans cette section les simulateurs à événements discrets qui sont capables de modéliser les MANETs.

D'abord, dans la section 6.3.1, nous présentons les critères de comparaison des simulateurs que nous avons choisis. Puis, dans la section 6.3.2, nous comparons trois simulateurs pour les MANETs qui sont compatibles avec l'outil BonnMotion. Ensuite, dans la section 6.3.3, nous établissons une synthèse sur les simulateurs étudiés.

6.3.1 Critères de comparaison des simulateurs

Pour comparer les simulateurs dans la littérature, nous avons retenu les critères suivants repris de [Hogie et al., 2006, Varga, 2006] :

- degré de granularité : les modèles de propagation radio, la simulation des couches physiques et MAC... doivent être considérés afin de rendre les simulations plus réalistes. Le réseau est modélisé avec un certain degré de granularité. Il existe 5 niveaux de granularité du niveau le plus fin (ou le plus détaillé) jusqu'au niveau application (le moins détaillé) : « le plus fin » puis « meilleur », « fin », « moyen », « niveau application » ;
- passage à l'échelle : à cause de la nature large échelle des réseaux de systèmes communicants, il est important de permettre le passage à l'échelle pour analyser quantitativement la performance du système ;
- techniques d'accélération : les simulateurs peuvent mettre en œuvre des techniques d'optimisation telles que la simulation par étapes (en anglais, *staged simulation*) ou le parallélisme, la partition (en anglais, *bining*). La simulation par étapes consiste à identifier et éliminer les calculs redondants. Le parallèlisme consiste à répartir le code et/ou les données du programme à exécuter sur plusieurs machines. La simulation par partition consiste à diviser l'aire de simulation en plusieurs sous-aires, chacune étant simulée séparément. Le réseau est partitionné de telle sorte que le nombre de nœuds simulés par partition est homogène.
- modèle de programmation : le modèle de programmation indique l'orientation des langages utilisés pour écrire les algorithmes et pour décrire la topologie du réseau ;
- plateforme de simulation : le simulateur est disponible pour un ou plusieurs systèmes

d'exploitation ;
- extensibilité : un simulateur extensible permet l'intégration simple de modifications dans un modèle, la réutilisation ainsi que l'extension d'une partie d'un modèle. Cela est facilité par l'utilisation d'APIs et par l'adoption d'une approche modulaire par composition ;
- contrôle et visualisation : l'environnent d'exécution du simulateur doit intégrer une interface graphique interactive qui permet de visualiser et examiner la progression de la simulation (pour le débogage et le support de traçage) ainsi que de configurer la simulation en changeant les paramètres ;
- documentation : une documentation (manuel, tutoriels, interfaces de programmation, etc.) détaillée et à jour facilite la prise en main et l'utilisation du simulateur ;
- popularité : il est difficile de connaître le nombre d'études qui utilisent un simulateur particulier. Pour estimer la popularité d'un simulateur, nous comptons le nombre de pages référencés sur le moteur de recherche Google Schoolar en envoyant la requête « `mobile ad hoc networks « nom du simulateur »` ».

6.3.2 Simulateurs récents utilisés pour les réseaux mobiles spontanés

Nous étudions trois des quatre simulateurs compatibles avec BonnMotion : ns2, GloMoSim et OMNeT++. Le quatrième simulateur COOJA [Österlind et al., 2006] est plutôt utilisé pour simuler les réseaux de capteurs sans fil. COOJA est originellement conçu pour les réseaux de nœuds qui exécutent le système d'exploitation Contiki [Dunkels et al., 2004], un système d'exploitation développé pour les appareils qui possèdent très peu de capacité de mémoire et de calcul tels que les microcontrôleurs et les capteurs. Par conséquent, nous ne considérons pas COOJA dans notre étude.

ns-2

ns-2 [McCanne and Floyd, 1989, Fall et al., 2000] est un simulateur à événements discrets dont le noyau est écrit en C++ de façon monolithique. Il peut être étendu en ajoutant des modules C++. Des scripts OTcl (un dialecte du Tcl) sont utilisés pour contrôler la simulation et spécifier d'autres aspects tels que la topologie du réseau ou l'enregistrement des résultats.

La simulation avec ns-2 consiste en 4 étapes. La première étape est l'implantation de l'algorithme étudié en écrivant une combinaison de codes C++ et OTcl. La deuxième étape consiste à écrire les scénarios de simulation dans un script OTcl. Enfin, les étapes 3 et 4 correspondent respectivement à l'exécution et à l'analyse des fichiers de traces générés.

Implanter un nouveau protocole dans ns-2 consiste à écrire du code C++ pour les fonctionnalités du protocole et à modifier des fichiers de configuration OTcl pour intégrer le protocole et ses paramètres au simulateur. Le modèle du système est organisé selon le modèle OSI.

ns-2 était au départ conçu comme un simulateur pour les couches TCP/IP. Grâce à sa licence libre et à sa popularité, plusieurs extensions ont été proposées. En particulier, le projet CMU Monarch [CMU Monarch project, 1999] propose une extension qui permet d'intégrer une implantation des couches IEEE 802.11 pour les réseaux WiFi. Par ailleurs, [Dricot and Doncker, 1992] propose des modèles physiques avec une précision importante basée sur la technique du lancer

de rayon (en anglais, *ray tracing*) et des chaînes de Markov. Cependant, la performance du simulateur est alors dégradée d'un facteur 100 [Dricot and Doncker, 1992].

ns-2 fournit une précision importante dans la modélisation des couches physiques du réseau. Il propose également un large ensemble de protocoles de communication. Il est l'un des simulateurs les plus utilisés [Kurkowski et al., 2005]. En outre, il fournit un environnement graphique pour observer le comportement du système étudié durant la simulation. En particulier, l'interface graphique de ns-2, appelée NAM (pour *Network Animator*), est riche et permet de visualiser la topologie du réseau, les liens entre les nœuds, l'état des files d'attente ainsi que les différents paquets échangés entre les nœuds. Cependant, le simulateur ns-2 manque de modularité et est intrinsèquement complexe. Cela est dû au fait que l'ajout ou la modification des composants/protocoles n'est pas une tâche aisée. De plus, sa documentation est fragmentée. Par ailleurs, il n'existe pas de frontière claire entre les modèles et les bibliothèques de la simulation. La topologie du réseau, les paramètres, la configuration, etc. sont décrits dans le même script Tcl. En outre, ns-2 ne possède pas d'outil d'analyse, ce qui complexifie la collecte des mesures. Enfin, le débogage est difficile à cause de la nature duale C++/OTcl du simulateur.

Un autre problème connu de ns-2 est la consommation importante des ressources de calcul. C'est une conséquence directe du problème de passage à l'échelle de ns-2.

GloMoSim

GloMoSim (*Global Mobile system Simulator*) [Zeng et al., 1998] est basé sur l'environnent PARSEC (*PARallel Simulation Environnement for Complex systems*) [Bagrodia et al., 1998] qui est un environnent d'exécution séquentielle ou parallèle. GloMoSim peut ainsi bénéficier des avantages du langage PARSEC pour exécuter les simulations sur des machines avec des architectures multiprocesseurs symétriques (en anglais, *symetric multiprocessing* ou SMP) à mémoire partagée. Les nouveaux algorithmes ou modules pour GloMoSim doivent être également écrits en PARSEC.

Comme ns-2, GloMoSim adopte une architecture du système suivant le modèle standard OSI. Le système réseau est défini par un ensemble de couches possédant des APIs. Un certain nombre de protocoles ont été développés pour chaque couche. Les modèles de ces couches peuvent être développés à différents niveaux de granularité.

GloMoSim est conçu en utilisant le concept d'objet autonome, appelé entité, qui est introduit par PARSEC. Ces entités sont exécutées en parallèle et communiquent entre elles par échange de messages. GloMoSim peut simuler des réseaux avec des milliers de nœuds.

L'utilisateur définit les scénarios dans les fichiers de configuration `app.conf` et `config.in`. Le fichier `app.conf` contient la description du trafic à générer (le type d'application, le débit binaire, etc.). Le fichier `config.in` contient d'autres paramètres comme le temps de simulation, le nombre de nœuds, etc. Les résultats statistiques sont collectés en mode textuel ou graphique. L'outil de visualisation est basé sur Java.

Actuellement, GloMoSim ne supporte que des protocoles de communication sans fil. Il est l'un des simulateurs les plus utilisés et a été choisi pour être le cœur du simulateur commercial QualNet développé par Scalable Network Technologies [Scalable Network Technologies, 2008].

Pour accélérer la simulation, GloMoSim utilise la technique de partition et la technique de répartition dans les grilles de Beowulf [Hogie et al., 2006]. Cependant, GloMoSim souffre du manque d'une documentation détaillée.

OMNeT++/MiXiM

OMNeT++ [Varga, 2001] est un simulateur à événements discrets conçu pour modéliser les réseaux de communication, les systèmes répartis ou parallèles. OMNeT++ est également capable de simuler n'importe quel système composé d'objets qui communiquent entre eux.

Dans OMNeT++, un modèle consiste en des modules. Les modules simples sont des unités d'exécution et sont écrits en C++. Ils peuvent être composés pour former des modules composites. Les composites peuvent à leur tour être composés. Le nombre de niveaux dans la hiérarchie de composition n'est pas limité. Les modules communiquent entre eux en envoyant des messages via des ports (en anglais, *gates*). Les ports sont des interfaces d'entrée et de sortie de modules qui peuvent être reliées par des connexions. Les connexions sont configurables par des propriétés telles que le délai de propagation, le débit binaire, etc.

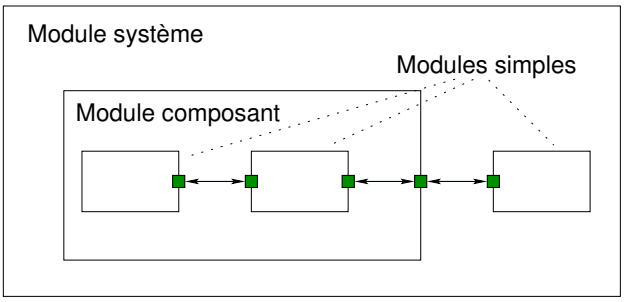

FIGURE 6.1 – Architecture du modèle de système dans OMNeT++.

Les modules simples et composites sont des instances de modules types. Pour décrire un modèle dans OMNeT++, l'utilisateur définit des modules types. Les instances de ces modules types servent à définir d'autres modules types plus complexes.

Le noyau du simulateur OMNeT++ est une bibliothèque, et les modèles sont indépendants du noyau. Plus précisément, les utilisateurs peuvent écrire leurs modules en utilisant les APIs fournies par le noyau. Les sources de OMNeT++ ne sont donc pas modifiés par l'intégration de nouveaux modèles (comme dans ns-2).

Le module qui représente le système correspond au module dont le niveau de hiérarchie est le plus haut (voir la figure 6.1) pour lequel il n'y pas de port de sortie. La réutilisation de module rend OMNeT++ flexible et a permis ses nombreuses extensions [OMNeT++ project, 2012]. En particulier, le projet MiXiM [MiXiM project, 2012, Köpke et al., 2008] montre que OMNeT++/MiXiM permet de simuler les MANETs incluant des réseaux sans fil pour les réseaux de capteurs, les réseaux véhiculaire, etc. MiXiM offre des modèles détaillés pour la propagation

radio, l'estimation des interférences, la consommation électrique et les protocoles de la couche MAC.

Les descriptions de topologie réseau peuvent être écrites dans des fichiers texte en utilisant le langage NED [Varga and Pongor, 1997] ou être dynamiquement créées à l'exécution. Il existe par ailleurs une interface graphique (GNED) pour créer et éditer les topologies. Les paramètres d'une simulation sont écrits dans le fichier de configuration `omnetpp.ini`. Cela permet de séparer les modèles des expérimentations. OMNeT++ possède un environnement graphique interactif qui permet d'examiner la progression de la simulation, de changer les paramètres, et de visualiser et d'analyser les résultats. OMNeT++ permet l'utilisation de la simulation parallèle simple basée sur MPI/PVM [Hogie et al., 2006, Sekercioglu et al., 2003]. OMNeT++ peut simuler des réseaux de grande taille. La limitation est la capacité de la mémoire virtuelle des machines utilisées. Enfin, la documentation de OMNeT++ est détaillée et régulièrement mise à jour. Celle de MiXiM reste insuffisante, mais la communauté OMNeT++/MiXiM est très dynamique et réactive.

6.3.3 Synthèse

Nous réalisons dans le tableau 6.3 une synthèse des trois simulateurs que nous avons étudiés. La première colonne regroupe l'ensemble des critères/concepts considérés pour comparer les simulateurs. Nous utilisons les notations suivantes :

- le mot « oui » indique que l'élément est géré ;
- le mot « non » indique que l'élément n'est pas géré ;
- le mot « insuffisant » indique que l'élément est géré, mais de manière insuffisante ;

Parmi les trois simulateurs étudiés, ns-2 est le simulateur le plus populaire et possède le degré de granularité « le plus fin ». ns-2 peut être utilisé pour évaluer des protocoles qui demandent un niveau de précision important. Il peut être utilisé pour les études des simulations de MANETs grâce à son large ensemble de protocoles de communication de la couche MAC à la couche réseau. Cependant, il est rigide et non extensible. En outre, ns-2 ne possède pas de documentation détaillée. Enfin, le problème principal de ns-2 est de ne pas pouvoir permettre le passage à l'échelle des simulations : il est difficile de simuler un système avec plus de 100 nœuds.

Un peu moins populaire que ns-2, GloMoSim est parmi les simulateurs les plus utilisés pour les simulations des MANETs. Il est considéré comme un bon candidat grâce à son aspect passage à l'échelle et son niveau de précision « fin ». La plupart des modèles de simulation de GloMoSim suivent l'architecture à sept couches de la norme OSI. La difficulté avec GloMoSim est d'implanter une application sans utiliser toutes ces couches OSI. Comme ns-2, GloMoSim souffre d'une documentation peu détaillée.

OMNeT++ possède une architecture orientée composant. Il possède un degré de modélisation « fin » et fournit un environnent d'exécution riche avec une interface graphique riche intégrée à Eclipse. Ces interfaces facilitent la construction, le suivi et l'analyse des modèles de simulation. OMNeT++ est moins populaire que ns-2 et GloMoSim. Ceci peut être expliqué par le fait que OMNeT++ est plus récent. Cependant, OMNeT++ possède une communauté très dynamique et est en constante évolution. Sa documentation est d'un bon niveau de détails.

Crièrte/Concept	Simulateurs récents utilisés pour les MANETs		
	GloMoSim	ns-2	OMNeT++/MiXiM
Degré de granularité	fin	le plus fin	fin
Modèle de programmation	PARSEC (basé sur C)	C++/OTcl	C++
plateforme de simulation	Unix, Linux, FreeBSD, Windows	FreeBSD, Linux, SunOS, Solaris, Windows (Cygwin)	Unix, Linux, Windows (MinGW)
Technique d'accélération	SMP/beowulf	non	MPI/PVM
Passage à l'échelle	oui	non	oui
Extensibilité	insuffisant	insuffisant	oui
Contrôle et visualisation	insuffisant	insuffisant	oui
Documentation détaillée	insuffisant	insuffisant	oui
Popularité	4300	21900	2220

TABLE 6.3 – Tableau récapitulatif des simulateurs pour les MANETs.

MiXiM suit de près les démarches adoptées par OMNeT++. Par conséquent, nous choisissons le simulateur OMNeT++/MiXiM, couplé avec BonnMotion, pour évaluer les performances de notre algorithme.

6.4 Évaluation des performances de $\Diamond\mathcal{PPD}$

Pour mesurer les performances de $\Diamond\mathcal{PPD}$, nous réalisons un modèle de réseau OMNeT++/MiXiM dans lequel nous mettons en œuvre l'implantation du détecteur (cf. l'algorithme 3) sous forme d'un module simple [32]. Dans cette étude préliminaire, nous choisissons un nombre limité de critères pour étudier les performances du détecteur $\Diamond\mathcal{PPD}$. Nous fixons aussi un nombre de valeurs limité pour certains paramètres généraux tels que la surface de simulation ou la carte d'interface réseau utilisée. En outre, nous considérons un nombre limité de valeurs pour chacun des paramètres spécifiques tels que le nombre minimum de processus stables α, les périodes des temporisations ${}_{part}timer$ et ${}_{proc}timer$ de l'algorithme 3.

Dans la section 6.4.1, nous présentons trois critères d'évaluation. Puis, dans la section 6.4.2, nous décrivons une vue d'ensemble des modules du modèle de réseau OMNeT++/MiXiM. Ensuite, dans la section 6.4.3, nous présentons les paramètres communs à tous les scénarios de simulation. Enfin, dans la section 6.4.4, nous présentons l'analyse des résultats des simulations.

32. Le code C++ du module OMNeT++ de l'algorithme $\Diamond\mathcal{PPD}$ est disponible à l'URL suivant : http://www-public.it-sudparis.eu/~lim_leon/simulations/alphaPPDSIM.tgz

6.4.1 Critères d'évaluation

Le détecteur $\diamondsuit\mathcal{PPD}$ permet aux nœuds mobiles de détecter ultimement leur ensemble stable α-Set. Rappelons que chaque processus p possède son ensemble local αSet_p. La valeur de αSet_p change durant les périodes instables. Pendant les périodes stables, ultimement, tous les processus stables de αSet_p peuvent communiquer entre eux via des chemins SADDM et possèdent la même connaissance de αSet_p. En outre, $|\alpha Set_p|$ doit être supérieur à α_q pour tout processus q de αSet_p, sinon les processus de αSet_p ne sont pas considérés comme stables. Autrement dit, pendant les périodes stables, $\forall q \in \alpha Set_p \; \exists \alpha : \alpha Set_q = \alpha Set_p \wedge |\alpha Set_p| \geqslant \alpha \geqslant \alpha_q$. Par ailleurs, l'intersection de deux ensembles de processus stables différents est vide. Dans la suite, nous notons $\alpha Set_p(t)$ la valeur retournée à p par $\diamondsuit\mathcal{PPD}_p$ à l'instant t.

Dans cette étude préliminaire des performances de $\diamondsuit\mathcal{PPD}$, nous nous focalisons sur les critères suivants :

- le nombre d'ensembles de processus stables α-Set construits au cours du temps,
- le nombre moyen de processus qui participent à la construction d'un ensemble de processus stable α-Set,
- la durée de vie moyenne des ensembles de processus stables α-Set.

L'ensemble d'ensembles de processus stables α-Set à un instant t est défini comme suit.

Définition 21. Ensemble d'ensembles de processus stables.
$\alpha SET(t) = \{\alpha Set_p(t) | \forall q \in \alpha Set_p(t) \; \exists \alpha : \alpha Set_q(t) = \alpha Set_p(t) \wedge |\alpha Set_p(t)| \geqslant \alpha \geqslant \alpha_q\}$.

Soit s_1 et s_2 deux ensembles de processus stables de αSET à l'instant t. Selon la définition de l'ensemble αSET, nous avons $s_1 \cap s_2 = \emptyset$. Le nombre d'ensembles de processus stables α-Set, dénoté A, correspond à la cardinalité de αSET.

Par la suite, nous notons respectivement $\alpha SET([t_1, t_2])$ et $A([t_1, t_2])$ l'ensemble d'ensembles de processus stables construits dans l'intervalle $[t_1, t_2]$ et sa cardinalité. Remarquons qu'un même ensemble de processus α-Set peut être inclus dans (considéré comme stable) et exclus (considéré comme instable) de l'ensemble αSET plusieurs fois dans l'intervalle de temps $[t_1, t_2]$.

Le nombre moyen de processus qui participent à la construction d'un ensemble de processus stables est déterminé comme suit.

Définition 22. Nombre moyen de processus participants.

$$K_{moy} = \begin{cases} \frac{\sum_{set \in \alpha SET([t_1,t_2])} |set|}{A([t_1,t_2])} & si\ \alpha SET([t_1, t_2]) \neq \{\} \\ 0 & sinon \end{cases}$$

Par exemple, considérons un scénario dans lequel l'ensemble des processus participant à la simulation $\mathbb{P}_{sim} \subseteq \mathbb{P}$ est constitué de quatre processus p, q, r et s avec leur valeur de α égale à 2. Supposons que la valeur de l'ensemble stable α-Set des processus p, q, r et s ne change pas pendant un intervalle de temps donné. Si $\alpha Set_p = \alpha Set_q = \alpha Set_r = \alpha Set_s = \{p, q, r, s\}$, alors $\alpha SET = \{\{p, q, r, s\}\}$, $A = 1$ et $K_{moy} = 4$. Si $\alpha Set_p = \alpha Set_q = \{p, q\}$ et $\alpha Set_r = \alpha Set_s =$

$\{r, s\}$, alors $\alpha SET = \{\{p, q\}, \{r, s\}\}$, $A = 2$ et $K_{moy} = 2$. Si $\alpha Set_p = \{p, q\}$, $\alpha Set_q = \{q, r\}$, $\alpha Set_r = \{r, s\}$ et $\alpha Set_s = \{s, p\}$, alors $\alpha SET = \{\}$, $A = 0$ et $K_{moy} = 0$.

La durée de vie d'un ensemble de processus stables α-Set correspond à la période de temps pendant laquelle α-Set est considéré comme stable, c'est-à-dire que tous les processus de α-Set possèdent la même connaissance de α-Set. La durée de vie moyenne des ensembles stables est définie comme suit.

Définition 23. Durée de vie moyenne des ensembles stables.

$$D_{moy} = \begin{cases} \frac{\sum_{set \in \alpha SET([t_1,t_2])} d_{set}}{A([t_1,t_2])} & si\ \alpha SET([t_1, t_2]) \neq \{\} \\ 0 & sinon \end{cases}$$

Dans la définition précédente, d_{set} est la durée de vie d'un ensemble stable *set* de αSET.

6.4.2 Vue d'ensemble des modules

La figure 6.2 présente l'architecture du modèle du réseau OMNeT++/MiXiM avec $\diamondsuit\mathcal{PPD}$. Le réseau contient plusieurs nœuds qui sont représentés par les modules Host_1, Host_2, ..., Host_n. L'architecture de chaque nœud est structurée en couche. Le module de la carte d'interface réseau Nic80211 est composé des modules PHY80211 et MAC80211. La couche MAC80211 est équipée d'un capteur de porteuse à accès multiple (en anglais, *Carrier Sense Multiple Access* ou CSMA) sans détection de collision. Les obstacles ne sont pas modélisés car nous ne ciblons pas des scénarios particuliers. Pour ajouter de l'entropie au système concernant les pertes de messages, nous utilisons le module ProbabilisticBroadcast fourni par MiXiM : chaque paquet (ou message) de la couche applicative est transmis à la couche MAC80211 pour être ensuite diffusé avec un taux de réussite fixé à 0,8. Cette valeur est la valeur fournie par défaut dans l'implantation du module ProbabilisticBroadcast. De plus, nous fixons la valeur du cœfficient d'affaiblissement [33] à 4.

Le module ConnectionManager est responsable de la création dynamique des ports qui relient les modules Host entre eux. Pour cela, il communique périodiquement avec le module BonnMotionMobility qui permet d'utiliser de façon native les traces de mouvements générées par l'outil BonnMotion. La communication entre les modules ConnectionManager et BonnMotionMobility n'est pas présentée sur la figure 6.2. Le module World est un module global qui fournit des méthodes génériques pour la collecte des données statistiques globales. Nous l'étendons pour calculer les données statistiques globales telles que le nombre des ensembles de processus stables A, le nombre moyen de processus participants K_{moy} et la durée de vie moyenne des ensembles de processus stables D_{moy}. Nous intégrons dans le module $\diamondsuit\mathcal{PPD}$ le code qui permet de collecter les données statistiques de $\diamondsuit\mathcal{PPD}$ spécifiques à chaque nœud telles que les valeurs des périodes associées aux temporisations des nœuds.

33. Dans les conditions idéales, à l'air libre, la valeur du cœfficient d'affaiblissement est 2. Nous choisissons la valeur donnée par défaut dans l'implantation du module PHY80211 de MiXiM.

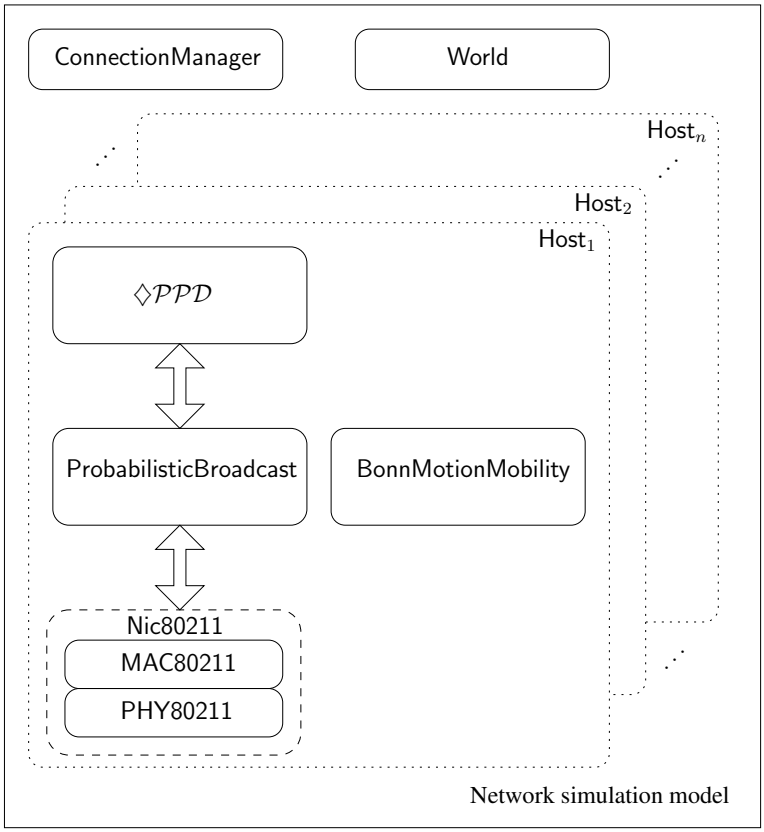

FIGURE 6.2 – Architecture du modèle du réseau OMNeT++/MiXiM avec ◇𝒫𝒫𝒟.

6.4.3 Paramètres généraux des simulations

Nous considérons un réseau mobile spontané composé de nœuds mobiles répartis de façon uniforme sur une surface de $1km \times 1km$ (pour les piétons) et de 7km × 5km (pour les véhicules). La couche physique possède un débit binaire de 2Mb par seconde. La probabilité de diffusion de chaque message est fixée à $0,8$. Les vitesses minimales et maximales, lorsqu'elles sont applicables à un modèle de mobilité donné, sont respectivement $0,5\text{ms}^{-1}$ et $1,5\text{ms}^{-1}$ pour les piétons, et $4,17\text{ms}^{-1}$ et $13,90\text{ms}^{-1}$ pour les véhicules. Le tableau 6.4 récapitule les paramètres généraux qui sont utilisés dans tous les scénarios de simulation en fonction du type de nœuds (piétons ou véhicules).

Paramètre	Valeur	
	piétons	véhicules
Surface de simulation	1km × 1km	7km × 5km
Carte d'interface réseau	IEEE 802.11	IEEE 802.11
Débit binaire	2Mbs^{-1}	2Mbs^{-1}
Type d'antenne	omnidirectionnelle	omnidirectionnelle
Coefficient d'affaiblissement	4	4
Probabilité de diffusion	0,8	0,8
Vitesse minimale	$0,5\text{ms}^{-1}$	$4,17\text{ms}^{-1}$
Vitesse moyenne	$1,0\text{ms}^{-1}$	$8,89\text{ms}^{-1}$
Vitesse maximale	$1,5\text{ms}^{-1}$	$13,89\text{ms}^{-1}$

TABLE 6.4 – Paramètres généraux de simulation.

Portée de transmission. L'outil BonnMotion permet l'analyse des traces de mouvements générées. Cette analyse est basée sur la portée de transmission des nœuds. Tous les nœuds sont considérés avoir la même portée de transmission. Le modèle radio utilisé dans BonnMotion est un mécanisme tout ou rien. Autrement dit, deux nœuds sont dits connectés lorsque la distance qui les sépare est en dessous de la valeur de la portée de transmission spécifiée. Sinon, ils sont considérés comme étant non connectés. Ainsi, le nombre de partitions réseaux à un instant donné correspond aux différents ensembles de nœuds connectés.

Comme nous utilisons BonnMotion pour analyser les traces de mouvements générées, il nous faut déterminer la valeur de la portée de transmission des nœuds dans les simulations. Dans le modèle radio des cartes d'interface réseau IEEE 802.11 de MiXiM, un message diffusé est reçu par le nœud récepteur si $txPower - attenuation > sensitivity$, où $txPower$ est la puissance de transmission, *attenuation* est l'affaiblissement du signal déterminé selon l'équation de Friis [Kvaksrud, T.I., 2008] et *sensitivity* est la sensibilité du signal[34]. Nous utilisons les valeurs de *sensibility* et de la puissance de transmission fournies par défaut dans MiXiM :

34. Pour qu'un signal soit intelligible pour le nœud récepteur, il faut que celui-ci ait une sensibilité suffisante.

respectivement −119.5 dBm et 110.11mW. Avec ces valeurs et la formule de Friis, la portée de transmission maximale vaut 313m [35]. En plus de la sensibilité, nous prenons également en compte le bruit thermique, ainsi la portée de transmission effective est plus faible [Köpke et al., 2008].

Bien que la portée de transmission maximale soit connue, le modèle radio IEEE 802.11 n'est pas un mécanisme tout ou rien : plus le nœud récepteur se trouve près du nœud émetteur plus le taux de perte est faible car le signal radio est assez fort. Au contraire, si le nœud récepteur se trouve au delà de la portée de transmission maximale du nœud émetteur, alors aucun message diffusé n'est reçu par le récepteur. Nous cherchons à déterminer la valeur de la portée de transmission effective avec les valeurs des paramètres présentés ci-dessus dans le tableau 6.4. Pour ce faire, nous considérons un scénario de simulation dans lequel deux nœuds jouent simultanément les rôles d'émetteur et de récepteur. Chaque nœud diffuse 100 messages avec un intervalle de 1s entre deux messages consécutifs.

La figure 6.3 montre le nombre de messages reçus en fonction de la distance qui sépare les deux nœuds. La courbe avec la ligne pleine et celle avec la ligne de tirets représentent respectivement le nombre de messages reçus par les nœuds $host1$ et $host2$. Nous observons que les deux courbes possèdent la même allure. Lorsque la distance qui sépare les deux nœuds est en dessous de 170m, le taux de réception des messages diffusés est 0,8. Cela correspond au taux de diffusion probabiliste que nous utilisons. À partir de 170m, le taux de réception des messages décroît rapidement. Il atteint 0% lorsque la distance qui sépare les nœuds devient supérieure à 218m. En conclusion, dans la suite, nous choisissons 170m comme étant la valeur de la portée de transmission utilisée pour analyser avec BonnMotion les traces de mouvements générées.

Délai de transfert des messages. Afin de fixer les valeurs des différentes temporisations de l'algorithme $\Diamond\mathcal{PPD}$, nous estimons le délai de transfert des messages dans des cycles composés de 2 à 5 nœuds. Rappelons que la temporisation ${}_{part}timer$ de l'algorithme 3 délimite une période d'une durée de ${}_{part}timeout$ secondes. À la fin de cette période, le processus p vérifie la condition de stabilité, c'est-à-dire si tous les processus dans αSet_p peuvent être encore considérés comme stables et si $|\alpha Set_p| \geqslant \alpha_p$. ${}_{part}timeout$ augmente si la condition de stabilité n'est pas satisfaite. Dès lors, un nœud qui reste isolé ou n'arrive pas à rejoindre une partition stable pendant une période de temps longue voit augmenter la valeur de ${}_{part}timeout$, qui peut atteindre une valeur élevée. Si ce nœud rejoint plus tard en tant que leader une partition stable, alors il impose aux autres nœuds de son ensemble stable un temps de latence de détection de la condition de stabilité très élevée, qui correspond à la valeur de ${}_{part}timeout$. Pour éviter ce scénario, nous imposons une valeur maximale à ${}_{part}timeout$. La valeur maximale de ${}_{part}timeout$, dénotée $\text{MAX}_{part}timeout$, dépend du délai de transfert des messages qui ont été transportés à travers un cycle, de la valeur seuil du comptage des battements de cœur THRESHOLD ainsi que du nombre de liens dans le cycle. Nous effectuons l'étude pour trois valeurs de THRESHOLD (1, 5 et 10) et trois valeurs de α (2, 3 et 5).

Pour mesurer le délai de transfert des messages transportés à travers un cycle, nous plaçons les nœuds sur un segment de telle sorte que la distance qui sépare deux nœuds consé-

35. Avec *attenuation* [dB] = $40LOG_{10}$(distance $[m]$) + $20LOG_{10}(maxTXPower\ [dBm])$ − 147.56 [dB] et $maxTXPower$ = 110.11mW = 20.42 dBm.

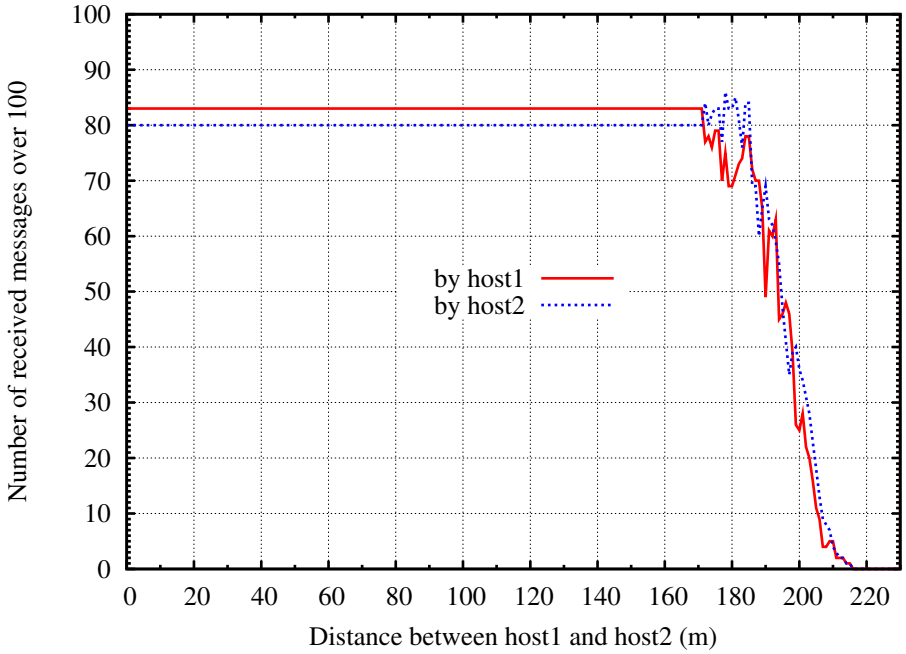

FIGURE 6.3 – Nombre de messages reçus en fonction de la distance qui sépare les nœuds *host*1 et *host*2.

cutifs soit égale à 150m ($<$ 170m). Considérons n nœuds $p_1 = p$, p_2,..., et $p_n = q$. p est le seul processus qui diffuse des messages et continue à faire ainsi tant qu'il n'a reçu aucun de ses messages diffusés ayant traversé un cycle en passant par q. Notons que les chemins associés à de tels messages sont composés de deux chemins SADDM : $saddm\_path(p_1p_2 \dots p_n)$ et $saddm\_path(p_np_{n-1} \dots p_1)$. Le nombre total de nœuds dans ces chemins est $2n$. Autrement dit, ces messages ont été transportés par $2(n-1)$ liens SADDM. Ainsi, le délai de transfert des messages correspondant est borné par $\beta^{2(n-1)}\eta + 2(n-1)\delta$ (cf. la définition 15 et le lemme 1). Comme dans [Friedman and Tcharny, 2009], nous fixons à 1s la période des battements de cœur η. Observons que p reçoit un message avec le chemin $(p_1p_2 \dots p_np_{n-1} \dots p_3p_2)$ avec la probabilité $0,8^{2(n-1)}$. Le taux de perte des messages augmente donc très rapidement avec le nombre de nœuds dans le cycle. Autrement dit, le délai de transfert des messages transportés à travers un cycle devient rapidement très important avec l'augmentation du nombre de nœuds dans le cycle.

Pour cette estimation des délais de transfert, nous effectuons 100 mesures pour chaque expérience et prenons la moyenne. Le tableau 6.5 présente le délai de transfert des messages transportés à travers un cycle en fonction du nombre de liens. Nous observons que quand le nombre de liens est égal à 8, ce délai est très élevé, ce qui traduit un taux de perte des messages important $(1 - 0,8^8 = 0,83)$.

Nombre de liens	Délai de transfert à travers un cycle
2	1,68s
4	4,48s
6	9,60s
8	65,62s

TABLE 6.5 – Délai de transfert des messages qui ont été transportés à travers un cycle composé de n liens.

Le tableau 6.6 présente les valeurs que nous choisissons au début des simulations pour ${}_{part}timeout$ ainsi que pour $\text{MAX}_{part}timeout$. Dans le meilleur des cas, les processus d'un ensemble stable peuvent directement communiquer entre eux, c'est-à-dire que le graphe du réseau est complet. Ainsi, ${}_{part}timeout$ est initialisée à $1,68\text{s} \times$ THRESHOLD. Ainsi, pour les valeurs 1, 5 et 10 de THRESHOLD, ${}_{part}timeout$ vaut respectivement 1.68s, 8,4s et 16,8s. Lorsque la valeur de la période ${}_{part}timeout$ n'est pas suffisante pour détecter un ensemble de processus stables α-Set, elle augmente avec un pas que nous fixons à 0,1s. Quant à la valeur de $\text{MAX}_{part}timeout$, dans le cas où α est égal à 2, nous la fixons à deux fois la valeur de ${}_{part}timeout$. Pour $\alpha \geqslant 3$, la valeur de $\text{MAX}_{part}timeout$ est fixée à 1,68s $\times$ le délai de transfert des messages ayant traversé un cycle composé de $2(\alpha - 1)$ liens. Par exemple, pour α égal à 3 et THRESHOLD égal à 5, nous avons ${}_{part}timeout = 5 \times 4,48s = 22,4s$.

Rappelons que la temporisation ${}_{proc}timer$ de l'algorithme 3 est utilisée pour compter les battements de cœur des processus mutuellement atteignables. La valeur de la période ${}_{proc}timeout$ associée à la temporisation ${}_{proc}timer$ est initialisée à 1,68s pour capturer le scénario du meilleur

des cas dans lequel les processus de l'ensemble stable peuvent tous directement communiquer entre eux. ${}_{proc}timeout$ augmente aussi avec un pas de 0,1s.

Paramètre	Valeurs								
α	2			3			5		
THRESHOLD	1	5	10	1	5	10	1	5	10
MAXHB	5	10	20	5	10	20	5	10	20
${}_{part}Timeout$	1,68s	8,4s	16,8s	1,68s	8,4s	16,8s	1,68s	8,4s	16,8s
MAX${}_{part}Timeout$	3,36s	17s	32s	4,5s	22,4s	45s	65s	328s	656s

TABLE 6.6 – Les valeurs des temporisations ${}_{part}timeout$ et MAX${}_{part}timeout$ en fonction de THRESHOLD et de α.

6.4.4 Génération et analyse des résultats de simulation

Nous utilisons BonnMotion pour générer des traces de mouvements des nœuds mobiles pour une période de 3600s. Pour diminuer le problème de la distribution spatiale uniforme des nœuds, sauf pour le modèle SSRWP, une phase initiale d'une durée de 3600s est ajoutée.

Dans l'analyse des traces de mouvements générées, nous étudions les paramètres suivants :

- degré moyen d'un nœud : le degré moyen du graphe de réseau non orienté, c'est-à-dire le nombre moyen de nœuds qui sont connectés directement à un nœud ;
- nombre moyen de partitions : le nombre moyen d'ensembles de nœuds connectés, c'est-à-dire le nombre de partitions réseaux. Quand il vaut 1, cela signifie que tous les nœuds sont connectés directement ou indirectement entre eux pendant toute la durée de la simulation ;
- vitesse moyenne d'un nœud : la vitesse moyenne des nœuds au cours du temps.

Nous évaluons les performances de l'algorithme avec cinq modèles de mobilité pour les piétons : RWP, SSRWP, Gauss-Markov, Manhattan et RPGM. Pour les véhicules, nous considérons le modèle Manhattan. Par manque de temps, les modèles *Random Street* et *Community Based* sélectionnés dans la section 6.2.6 ne sont ni implantés ni intégrés au simulateur OMNeT++/MiXiM, et ne sont donc pas considérés dans cette étude préliminaire. Pour chacun des six modèles étudiés, nous présentons maintenant l'analyse des traces de mouvements générées et des résultats des simulations.

Scénarios de mobilité avec le modèle RWP

Génération des traces de mouvements. Pour générer les scénarios de mobilité avec le modèle RWP avec 10, 25 et 50 nœuds, nous fixons les valeurs suivantes. Les vitesses minimale et maximale des nœuds sont respectivement 0,5ms^{-1} et 1,5ms^{-1}. Nous fixons le temps de pause maximum d'un nœud à 60s. Les nœuds sont placés initialement de façon uniforme dans une zone de 1km × 1km. La durée de la simulation est 3600s avec l'ajout de la phase initiale d'une durée de 3600s. Enfin, la graine aléatoire utilisée est 71777.

Analyse des traces de mouvements. Le tableau 6.7 présente les résultats de l'analyse des traces de mouvements générées avec 10, 25 et 50 nœuds. La vitesse moyenne des nœuds ne se dégrade pas de façon importante au cours du temps et s'approche de la distribution de leur vitesse moyenne au début des simulations. Le degré moyen d'un nœud augmente avec le nombre total de nœuds participant à la simulation : il est respectivement égal à 1,16, 2,6 et 5,46 pour les scénarios avec 10, 25 et 50 nœuds. En revanche, le nombre moyen de partitions diminue avec l'augmentation du nombre de nœuds : il est respectivement égal à 5,27, 5,63 et 2,83 pour les scénarios avec 10, 25 et 50 nœuds. Ces résultats peuvent être expliqués par le fait que le modèle RWP est un modèle de mobilité individuelle. Comme les nœuds sont répartis de façon uniforme dans la zone de simulation, l'augmentation du degré moyen d'un nœud reste relativement faible par rapport à l'augmentation du nombre total de nœuds participant à la simulation. L'effet de groupe (c'est-à-dire la constitution des partitions réseau) est généré par les déplacements aléatoires des nœuds dans une zone de taille limitée. Ainsi, plus le nombre de nœuds participant à la simulation augmente, plus le nombre de partitions réseaux diminue car les nœuds ont plus de chance de se trouver dans la portée de transmission d'autres nœuds.

Paramètre	Valeurs		
Nombre de nœuds	10	25	50
Degré moyen d'un nœud	1,16	2,6	5,46
Nombre moyen de partitions réseau	5,27	5,63	2,83
Vitesse moyenne d'un nœud	0.91ms^{-1}	0.88ms^{-1}	0.88ms^{-1}

TABLE 6.7 – Résultats de l'analyse des traces de mouvements générées avec le modèle RWP.

Analyse des résultats des simulations. Le tableau 6.8 présente les résultats des simulations obtenus avec le modèle RWP. Le nombre d'ensembles de processus stables A, le nombre moyen de processus participant à la construction d'un ensemble stable K_{moy}, la durée de vie moyenne d'un ensemble stable D_{moy} et la valeur moyenne des temporisations (à la fin de la simulation) ${}_{part}timeout_{moy}$ sont présentés pour différentes valeurs de α, du seuil THRESHOLD et du nombre de nœuds. Les valeurs de K_{moy} et de ${}_{part}timeout_{moy}$ sont présentées avec un intervalle de confiance de 95%. Par exemple, pour $\alpha = 3$, THRESHOLD $= 5$ avec 10 nœuds, nous avons $K_{moy} = 3,87 \pm 0,78$, ${}_{part}Timeout = 17,87 \pm 1,29$s, $A = 8$ et $D_{moy} = 158$s.

Le graphe de la figure 6.4 présente A en fonction de α, de THRESHOLD et du nombre de nœuds. A augmente lorsque la valeur de α diminue pour les trois scénarios. Par exemple, pour le scénario avec 10 nœuds, THRESHOLD égal à 5, et α égal à 2, 3 et 5, A est égal respectivement à 21, 8 et 2. Ceci traduit le fait que les ensembles stables de petite taille ont plus de chance d'être construits que des ensembles stables de grande taille. Ainsi, A augmente aussi lorsque la valeur de THRESHOLD diminue. En effet, plus la valeur de THRESHOLD est faible, moins la durée de présence des processus mutuellement atteignables dans la partition stable doit être importante avant que ces processus soient considérés comme stables. Cependant, la durée de vie moyenne de ces ensembles de processus stables est faible. En particulier, pour le scénario avec 50 nœuds, α égal à 2 et THRESHOLD égal à 1, il y a au total 92 ensembles de processus stables construits,

Paramètre	Valeurs								
α	2			3			5		
THRESHOLD	1	5	10	1	5	10	1	5	10
MAXHB	5	10	20	5	10	20	5	10	20

10 nœuds									
A	27	21	11	17	8	4	9	2	1
K_{moy}	2,6 ±0,26	3,14 ±0,64	2,72 ±0,6	4 ±0,53	3,87 ±0,78	4 ±0,80	5,66 ±0,32	5,5 ±0,97	5 –
D_{moy}	240s	193s	393s	118s	158s	172s	97s	259s	340s
${}_{part}timeout_{moy}$	3,28 ±0,00s	14,81 ±1.24s	19,94 ±0,99s	4,48 ±0,00s	17,87 ±1,29s	21,59 ± 1,18s	19,98 ±0,79s	20,51 ±0,77s	23,82 ±0,97s

25 nœuds									
A	70	17	6	35	5	3	13	2	1
K_{moy}	3,64 ±0,51	4,06 ±1,74	3,5 ±1,58	4,51 ±1,50	5,2 ±4,56	5 ±1,95	6,7 ±0,67	5,5 ±0,97	6 –
D_{moy}	90s	149,5s	111s	48s	163s	44s	50,3s	90s	84,5s
${}_{part}timeout_{moy}$	3.25 ±0,05	10.00 ±0,48s	17,26 ±0,18s	4,48 ±0,00	10,74 0,69s	17,54 ±0,32	9,98 0,80s	11,79 ±0,66s	17,83 ±0,40s

50 nœuds									
A	92	6	2	18	1	0	7	0	1
K_{moy}	2,63 ±0,29	2,5 ±0,97	2 ±0,00	3,89 ±0,41	3 –	0 –	6,14 ±0,99	0 –	6 –
D_{moy}	37,6s	552s	1153s	37s	51,5s	0s	60,3	0s	67,6s
${}_{part}timeout_{moy}$	3.28 ±0,00s	9,20 ±0,24s	17,00 ±0,06s	4,47 ±0,02s	9,50 ±0,29s	17,10 ± 0,10s	7,87 ±0,36s	9,86 ±0,30s	17,20 0,13s

TABLE 6.8 – Résultats des simulations obtenus avec le modèle RWP..

mais avec une durée de vie moyenne courte (37,6s).

Le graphe de la figure 6.5 présente K_{moy} en fonction de α, de THRESHOLD et du nombre de nœuds. K_{moy} est toujours supérieur à la valeur de α car chaque ensemble stable est toujours constitué d'au moins α processus stables. K_{moy} augmente donc avec la valeur de α. Notons que K_{moy} n'augmente pas nécessairement avec l'augmentation du nombre total de nœuds car les partitions réseau sont constituées suite aux déplacements aléatoires des nœuds. Ainsi, lorsque le nombre total de nœuds n'est pas assez important, les nœuds sont regroupés en petits groupes pendant de courtes périodes. De tels groupes de nœuds sont éparpillés sur l'aire de simulation. Observons aussi que quand THRESHOLD diminue, K_{moy} augmente légèrement. En effet, avec la diminution de THRESHOLD, la durée de présence des processus mutuellement atteignables dans la partition stable avant d'être considérés comme stables diminue également. Ainsi, les processus mutuellement atteignables sont considérés comme stables plus rapidement. Il est également intéressant d'observer que pour le scénario avec 50 nœuds et α égal à 5, K_{moy} vaut 6 lorsque THRESHOLD = 10, et 0 lorsque THRESHOLD = 5, avec des valeurs moyennes des tem-

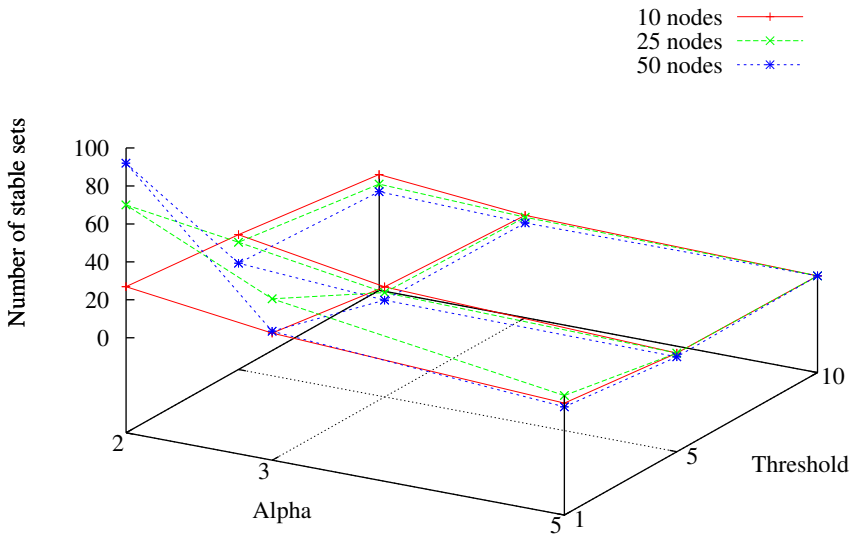

FIGURE 6.4 – Scénario RWP : le nombre d'ensembles de processus stables αSet en fonction du nombre de nœuds, de la valeur du seuil THRESHOLD et de α.

porisations ${}_{part}timeout$ atteignant respectivement 17,20s et 9,86s. Ceci semble à première vue contre-intuitif. Dès le début de la simulation, la valeur de ${}_{part}Timeout$ est plus importante pour THRESHOLD = 10 (16,8s) que pour THRESHOLD = 5 (8,4s). Ainsi, avec ${}_{part}Timeout = 16,8$s, les nœuds réussissent à constituer un ensemble stable car les processus disposent d'un délai suffisant pour que le nombre de battements de cœur des nœuds dépasse la valeur seuil (cf. tâche 5 de l'algorithme 3). En analysant pas à pas la simulation, ce n'est pas le cas lorsque ${}_{part}Timeout = 8,4$s. Ceci traduit le fait qu'un groupe de nœuds composé de α processus peut être formé si ces nœuds acceptent de travailler avec un temps de latence de détection de la condition de stabilité plus importante.

Le graphe de la figure 6.6 présente D_{moy} en fonction de α, de THRESHOLD et du nombre de nœuds. D_{moy} décroît avec l'augmentation du nombre total de nœuds. Observons que dans le scénario avec 50 nœuds, α égal à 2 et THRESHOLD égal à 5, la valeur de D_{moy} est importante (1153s). Ceci est un cas particulier dans le sens où parmi les traces de mouvements individuels générées avec une graine donnée, deux couples de deux nœuds restent proches les uns des autres dans leurs déplacements pendant de longues périodes. Le résultat est que ces deux couples de deux nœuds restent connectés entre eux pendant de longues périodes. Par ailleurs, D_{moy} décroît lorsque THRESHOLD diminue. Ceci traduit le fait que les ensembles stables peuvent avoir une durée de vie plus importante si les processus acceptent de travailler avec un temps de latence

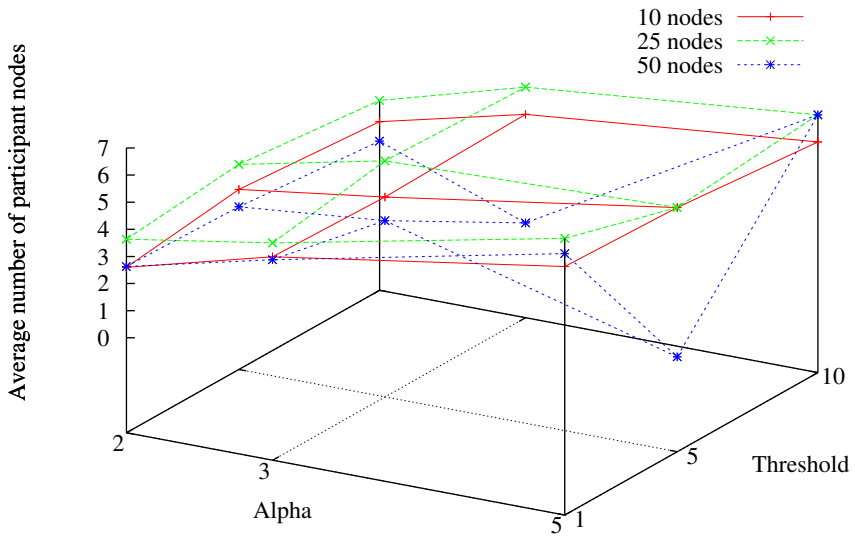

FIGURE 6.5 – Scénario RWP : le nombre moyen de processus stables participant à la construction d'un ensemble stable αSet en fonction du nombre de nœuds, du seuil THRESHOLD et de α.

plus élevé.

Concernant les autres autres modèles de mobilité individuelle (SSRWP, Gauss-Markov et Manahattan), les traces de mouvements générées avec ces modèles possèdent les mêmes caractéristiques que celles obtenues avec le modèle RWP. En particulier, le degré moyen d'un nœud augmente avec le nombre total de nœuds participant à la simulation, et le nombre moyen de partitions diminue avec l'augmentation du nombre de nœuds. En outre, les résultats des simulations obtenus avec les modèles SSRWP, Gauss-Markov et et Manahattan sont très similaires à ceux obtenus avec le modèle RWP. L'ensemble des résultats obtenus pour les modèles SSRWP, Gauss-Markov et Manhattan sont présentés dans l'annexe A.

Scénarios de mobilité avec le modèle RPGM

Génération des traces de mouvements. BonnMotion intègre la possibilité d'avoir des groupes dynamiques et statiques pour le modèle de mobilité RPGM. Dans notre étude, nous considérons les scénarios de mobilité avec des groupes dynamiques. Rappelons que dans ces derniers, quand un nœud entre dans l'aire d'un autre groupe, il devient membre de ce groupe avec une probabilité c. Nous considérons pour c la valeur donnée par défaut dans l'implantation du modèle RPGM de BonnMotion, qui est égal 0,5. Nous fixons le nombre moyen de nœuds par groupe à 5. Enfin, nous fixons la distance maximale qui sépare un nœud membre du centre de

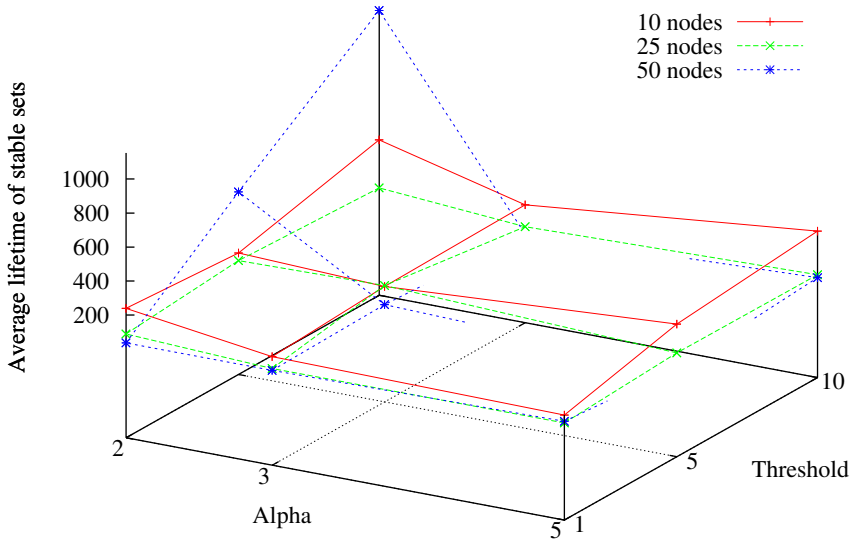

FIGURE 6.6 – Scénario RWP : la durée de vie moyenne des ensembles de processus stables αSet en fonction du nombre de nœuds, du seuil THRESHOLD et de α.

son groupe à 4m pour capturer par exemple le scénario dans lequel un groupe de personnes se déplacent ensemble le long d'une rue.

Analyse des traces de mouvements. Le tableau 6.9 présente les résultats de l'analyse des traces de mouvements générés avec 10, 25 et 50 nœuds. La vitesse moyenne des nœuds au cours du temps est peu dégradée ; elle est proche de 1ms^{-1}. Le degré moyen d'un nœud augmente avec le nombre total de nœuds participant à la simulation. Notons que le degré moyen d'un nœud est plus élevé que celui dans les modèles de mobilité individuelle car les nœuds forment des partitions réseaux (ou groupes au sens RPGM du terme). En outre, contrairement aux modèles de mobilité individuelle, dans le modèle de mobilité de groupe RPGM, plus le nombre de nœuds augmente plus le nombre moyen de partitions réseau augmente. En effet, plus il existe de nœuds, plus il existe de partitions réseau car la taille moyenne d'une partition est fixée à 5. Ainsi, pour les scénarios avec 10, 25 et 50 nœuds, le nombre moyen de partitions réseaux est égal respectivement à 1,34, 3,65 et 5,78.

Analyse des résultats des simulations. Le talbeau 6.10 présente les résultats des simulations obtenus avec le modèle RPGM. Comme pour les résultats avec le modèle RWP, nous analysons les résultats à travers plusieurs graphes.

Le graphe de la figure 6.7 présente A en fonction de α, de THRESHOLD et du nombre de nœuds. A augmente de façon importante lorsque α et THRESHOLD diminuent. Par exemple,

Paramètre	Valeurs		
Nombre de nœuds	10	25	50
Degré moyen d'un nœud	4,59	8,60	8,90
Nombre moyen de partitions réseau	1,34	3,65	5,78
Vitesse moyenne d'un nœud	0,86ms^{-1}	0,86ms^{-1}	0,85 ms^{-1}

TABLE 6.9 – Résultats de l'analyse des traces de mouvements générées avec le modèle RPGM.

Paramètre	Valeurs								
α	2			3			5		
THRESHOLD	1	5	10	1	5	10	1	5	10
MAXHB	5	10	20	5	10	20	5	10	20

10 nœuds									
A	9	5	4	4	5	4	4	2	1
K_{moy}	4,55 ±0,98	4,8 ±0,96	4,5 ±0,98	5,5 ±0,98	4,8 ±0,96	4,5 ±0,98	6 ±0	6 ±0	6 –
D_{moy}	628s	1286s	1606s	798s	1277s	1593s	728s	1739s	3566s
${}_{part}timeout_{moy}$	2,74 ±0,28s	8,53 ±0,04	16,9 ±0	2,75 ±0,42s	8,53 ±0,42s	16,9 ±0	11,76 ±7,06	15,55 ±5,61	21,81 ±3,92

25 nœuds									
A	29	20	16	16	14	10	35	12	15
K_{moy}	5,69 ±1,19	8,45 ±1,69	8,19 ±2,14	8,87 ±2,00	8,43 ±1,71	7,1 ±1,92	11,74 ±1,23	8 1,30	10,13 ±1,92
D_{moy}	113s	340s	411s	177s	312s	709s	109s	427s	362s
${}_{part}timeout_{moy}$	13 ±2,86s	9,3 ±0,65s	17,34 ±0,68s	4,45 ±0,05s	10,12 ±1,20s	17,6 ±0,74s	8,88 ±1,85s	11,45 ±3,84s	18,29 ±0,91

50 nœuds									
A	107	34	19	79	31	16	32	31	16
K_{moy}	5,21 ±0,56	8,76 ±1,70	7,10 ±1,37	7,98 ±0,69	6,13 ±0,89	5,75 ±0,85	7,69 ±0,65	10,10 ±1,57	9,62 ±1,78
D_{moy}	56s	465ss	811s	96s	561s	1268s	124s	228s	528s
${}_{part}timeout_{moy}$	3,28 ±0s	8,96 ±0,13s	16,94 ±0,02s	4,46 ±0,02	9,15 ±0,19	16,9 ±0,03	10,32 ±1,25	11,95 ±1,13s	18,27 ±0,7s

TABLE 6.10 – Résultats obtenus avec le modèle RPGM.

pour le scénario avec 10 nœuds et α égal à 2, A est égal à 27, 21 et 11 respectivement pour THRESHOLD égal à 1, 5 et 10. Ceci signifie que plusieurs ensembles de processus stables de petite taille sont construits au sein d'un même groupe au sens RPGM du terme. Notons que le nombre total de nœuds participant à la simulation contribue à l'augmentation de A.

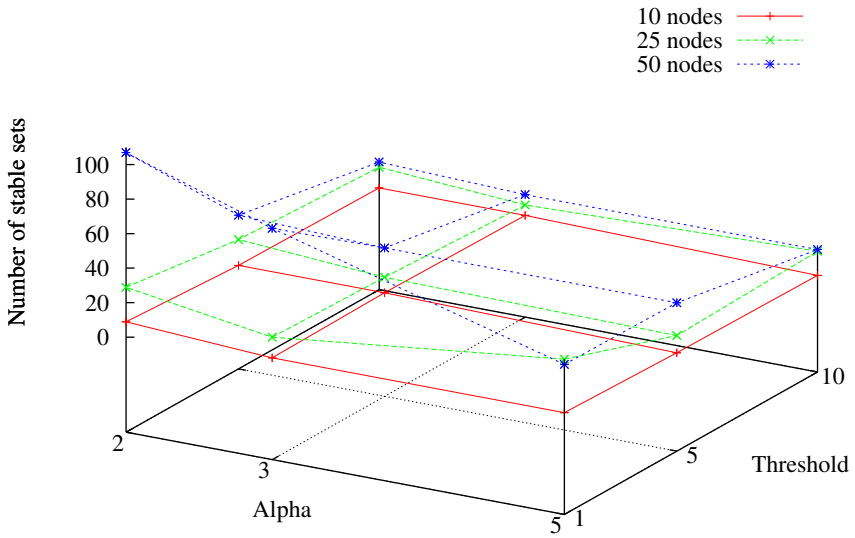

FIGURE 6.7 – Scénario RPGM : le nombre d'ensembles de processus stables en fonction de α, de THRESHOLD et du nombre de nœuds.

Le graphe de la figure 6.8 présente K_{moy} en fonction de α, de THRESHOLD et du nombre de nœuds. Comme nous l'avons déjà constaté lors de l'analyse des résultats pour les modèles de mobilité individuelle, K_{moy} est supérieur à α car chaque ensemble stable est composé d'au moins α processus stables. La particularité des résultats obtenus avec le modèle RPGM est que K_{moy} est supérieur à 4 et proche de la taille moyenne d'une partition réseau, qui vaut 5. Ceci montre les cas favorables dans lesquels les partitions réseaux de taille moyenne 5 existent dès le début de la simulation. Ainsi, des ensembles de processus stables de taille supérieure à α sont construits. En revanche, si la valeur de α est plus élevée que la taille de la partition réseau, alors aucun ensemble stable n'est construit. Par exemple, observons le scénario avec 10 nœuds, α égal à 5 et THRESHOLD égal à 10. Un ensemble stable composé de 6 nœuds est construit. Les quatre autres nœuds, bien qu'ils soient regroupés dans une partition réseau, ne peuvent pas constituer d'ensemble stable car ils sont en nombre insuffisant. Enfin, notons que la durée de vie de l'ensemble stable composé de six processus stables est importante (3566s). Ceci signifie que cet ensemble de processus stables n'a pas été « gêné » par les quatre autres

processus lors des fusions de partitions réseau : les processus de l'ensemble stable continuent à travailler ensemble comme si rien ne s'était passé. Plus précisément, lors de chaque fusion, chaque processus dans la partition réseau considère tous les autres processus comme stables, c'est-à-dire qu'ils sont inclus dans l'ensemble *tentative* du processus (cf. lignes 54–57 et tâche 5 de l'algorithme 3). Cependant, du point de vue des processus de l'ensemble stable composé de six processus, leur ensemble stable αSet reste inchangé car il est construit avant la fusion : c'est-à-dire que la condition de la ligne 24 de l'algorithme 3 reste fausse. Ainsi, les processus de l'ensemble stable ne changent pas leur ensemble stable. En outre, le leader de cet ensemble stable continue à diffuser son ensemble stable car la condition de la ligne 26 de l'algorithme 3 reste vraie : $6 = |\alpha Set| \geqslant \alpha = 5$.

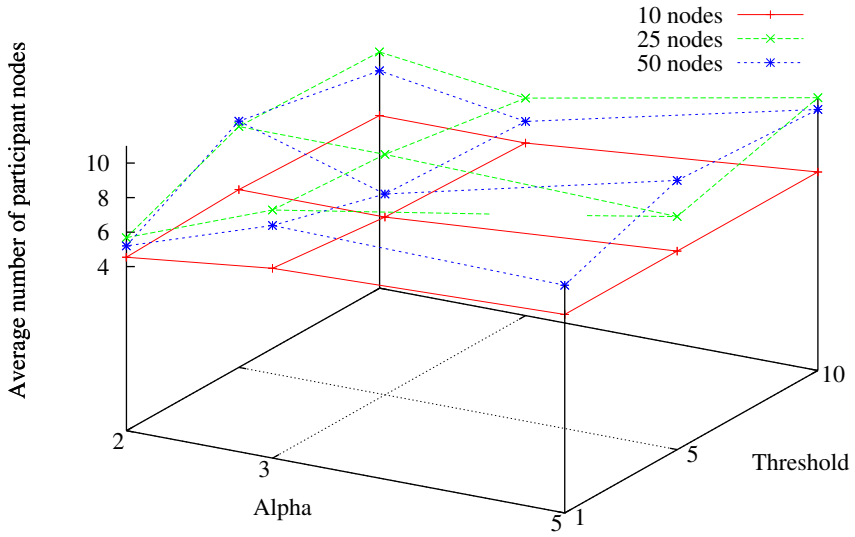

FIGURE 6.8 – Scénario RPGM : le nombre moyen de processus stables participant à la construction d'un ensemble stable αSet en fonction du nombre de nœuds, du seuil THRESHOLD et de α.

Le graphe de la figure 6.9 présente D_{moy} en fonction de α, de THRESHOLD et du nombre de nœuds. D_{moy} est beaucoup plus important pour le modèle RPGM que pour les modèles de mobilité individuelle. Pour l'ensemble des résultats avec le modèle RPGM, D_{moy} est compris entre 56s et 3566s, et la majorité des valeurs de D_{moy} sont supérieures à 400s. Observons que D_{moy} diminue avec l'augmentation du nombre total de nœuds. Autrement dit, les ensembles de processus stables vivent moins longtemps lorsque le nombre de partitions réseau augmente. En effet, plus le nombre de groupes RPGM augmente, plus ces groupes ont l'opportunité de

rencontrer d'autres groupes. Dès lors, ces groupes ont plus de chance de se décomposer pour ensuite se recomposer. Rappelons que la probabilité qu'un nœud change de groupe pour un autre, lorsque ces deux groupes se rencontrent, est fixée à 0.5. Dans l'ensemble des résultats, ${}_{part}timeout_{moy}$ est faible : elle varie entre 2,74s (pour le scénario avec 10 nœuds, $\alpha = 2$ et THRESHOLD $= 1$) et 21,81s (pour le scénario avec 10 nœuds, $\alpha = 5$ et THRESHOLD $= 10$). Ceci est dû au fait que les valeurs 2, 3 et 5 de α sont inférieures à la taille moyenne du groupe RPGM qui est égale à 5 et que les groupes RPGM existent dès le début de la simulation.

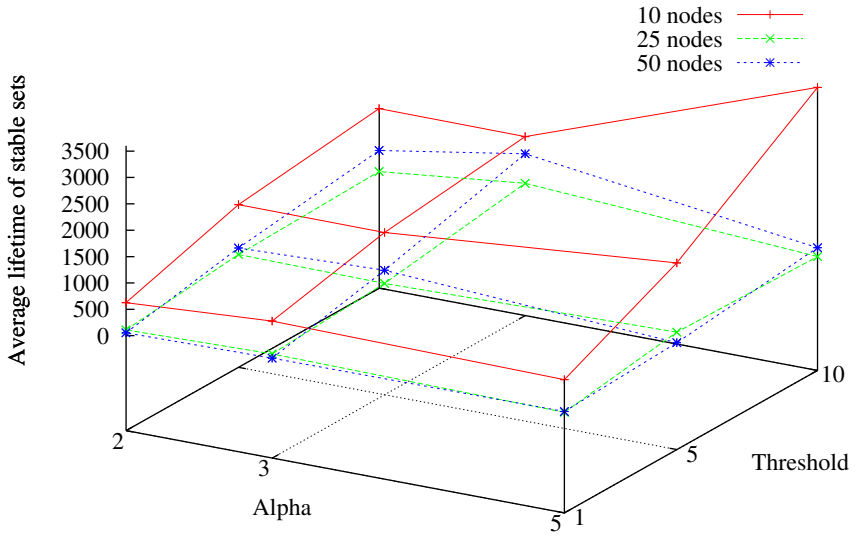

FIGURE 6.9 – Scénario RPGM : la durée moyenne des ensembles de processus stables αSet en fonction du nombre de nœuds, du seuil THRESHOLD et de α.

Le graphe de la figure 6.10 présente le nombre de partitions réseau à chaque fois qu'il change pour le scénario avec 10 nœuds ; il est obtenu à partir de l'analyse avec BonnMotion des traces de mouvements générées. Son évolution pour toute la durée de la simulation est représentée par la courbe pleine. Durant les intervalles]0 ;414]s,]599 ;1400]s et]1514 ;3600]s, deux partitions réseau (ou groupes au sens RPGM du terme) co-existent : le premier groupe est composé de six processus ($host_4$, $host_5$, $host_6$, $host_7$, $host_8$ et $host_9$) et le second groupe est composé de quatre processus ($host_0$, $host_1$, $host_2$ et $host_3$). Les fusions des deux groupes ont lieu à deux reprises aux instants 414s et 599s sans qu'il n'y ait ni décomposition ni recomposition de groupes. Dans le même graphe, la courbe avec les tirets représente l'évolution du nombre de processus stables pour le même scénario avec α égal à 5 et THRESHOLD égal à 10. Comme expliqué dans l'analyse du graphe de K_{moy} de la figure 6.8, lors des fusions des deux partitions réseau, l'ensemble de

processus stables αSet_9 composé des six processus reste inchangé. Durant les périodes de fusion, chaque processus considère tous les autres processus de la partition réseau comme stables, c'est-à-dire tous les processus sont inclus dans l'ensemble *tentative* (cf. lignes 54–57 et tâche 5 de l'algorithme 3). Du point de vue des processus stables de l'ensemble αSet_9, αSet_9 reste inchangé car la condition $6 = |\alpha Set_9| \geqslant \alpha = 5$ (cf. ligne 26 de l'algorithme 3) reste vraie. Ainsi, le leader de αSet_9 ($host_9$) continue à diffuser périodiquement son α-Set comme si rien ne s'était passé. Ceci explique pourquoi la durée de vie de l'ensemble stable αSet_9 atteint 3566s. Remarquons au passage que la latence de détection de la condition de la stabilité parmi les processus de αSet_9 est 34s.

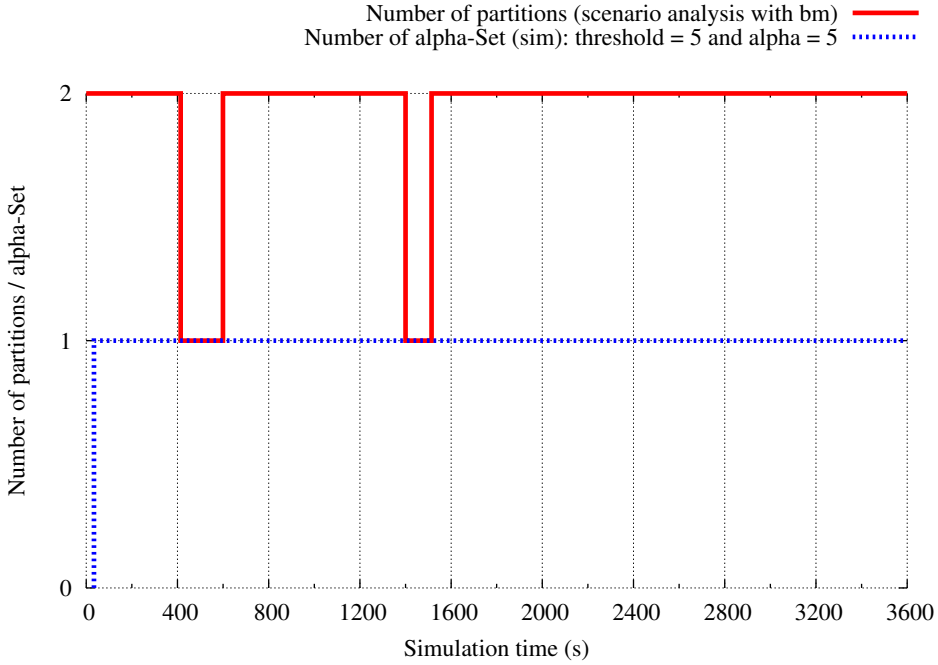

FIGURE 6.10 – Dans le scénario RPGM avec 10 nœuds, évolution du nombre de partitions et du nombre d'ensembles de processus stables α-Set au cours du temps.

6.5 Conclusion

Dans ce chapitre, nous avons évalué les performances de $\Diamond\mathcal{PPD}$ par simulation. Nous avons étudié de façon approfondie plusieurs modèles de mobilité de différents types (mobilité individuelle, mobilité de groupe et mobilité pour les réseaux en grappes) ainsi que les simulateurs utilisés pour les MANETs. Cela nous a permis de sélectionner cinq modèles de mobilité individuelle (RWP, SSRWP, Gauss-Markov, Manhattan et *Random Street*), un modèle de mobilité de groupe (RPGM) et un modèle de mobilité pour les réseaux en grappes (*Community-Based*).

Concernant le simulateur, nous avons choisi OMNeT++/MiXiM couplé avec l'outil de génération et d'analyse de traces de mouvements BonnMotion. OMNeT++/MiXiM (version 4.2.1) fournit un modèle de réseau avec un niveau de granularité fin. BonnMotion (version 2.0) permet la génération et l'analyse des traces de mouvements de plusieurs modèles de mobilité incluant les modèles RWP, SSRWP, Gauss-Markov, Manhattan et *Random Street*. Les modèles *Community-Based* et *Random Street* ne sont pas implantés dans BonnMotion.

Dans l'évaluation des performances de $\diamondsuit\mathcal{PPD}$, nous avons conçu un modèle de réseau OMNeT++/MiXiM dans lequel nous mettons en œuvre l'implantation du détecteur $\diamondsuit\mathcal{PPD}$ sous forme d'un module simple. Dans cette étude préliminaire, nous définissons trois critères : *1)* le nombre d'ensembles de processus stables construits au cours du temps A, *2)* le nombre moyen de processus qui participent à la construction d'un ensemble de processus stable K_{moy} et *3)* la durée de vie moyenne des ensembles de processus stables D_{moy}. Avec ces trois critères, nous évaluons les performances de l'algorithme $\diamondsuit\mathcal{PPD}$ avec les modèles de mobilité RWP, SSRWP, Gauss-Markov, Manhattan et RPGM pour les piétons. Pour les véhicules, nous considérons le modèle Manhattan. Par manque de temps, les modèles *Community-Based* et *Random Street* ne sont ni implantés et ni intégrés au simulateur OMNeT++/MiXiM ; l'évaluation des performances de $\diamondsuit\mathcal{PPD}$ n'a donc pas pu être effectuée avec ces deux modèles.

Les résultats des simulations obtenus avec les modèles de mobilité individuelle (piétons et véhicules) possèdent des caractéristiques très similaires. En particulier, A augmente de façon importante lorsque les valeurs de α et du seuil de comptage des battements de cœur THRESHOLD diminuent. Le nombre total de nœuds participant à la simulation contribue à l'augmentation de A. D'une manière générale, D_{moy} est faible. K_{moy} diminue avec l'augmentation du nombre total de nœuds. L'ensemble des résultats obtenus avec les modèles de mobilité individuelle peuvent être expliqués par le fait que les partitions réseaux ont une durée de vie faible. En effet, l'effet de groupe (c'est-à-dire la constitution des partitions réseau) est généré par les déplacements aléatoires des nœuds. Il est également intéressant de noter que dans certains cas des ensembles stables composés de plus de α nœuds sont formés si ces nœuds acceptent de travailler avec une latence de détection de la condition de stabilité plus importante.

Concernant, les résultats obtenus avec le modèle de mobilité de groupe RPGM, A augmente de façon importante lorsque THRESHOLD diminue. Ceci traduit le fait que plusieurs ensembles stables de petite taille sont construits au sein d'un même groupe (au sens RPGM du terme). K_{moy} est proche de, voire supérieure à, la taille moyenne d'un groupe. Ceci montre que l'ensemble stable a tendance à converger vers le groupe RPGM lorsque la valeur de α est inférieure à la taille du groupe. Autrement dit, même si la valeur de α est beaucoup faible que la taille du groupe, l'ensemble stable composé des processus de la taille du groupe est construit. Dans le cas contraire, aucun ensemble stable ne peut être construit si α dépasse la taille du groupe.

En conclusion, les résultats obtenus montrent que l'algorithme $\diamondsuit\mathcal{PPD}$ respecte le comportement spécifié. En particulier, l'algorithme permet aux processus de détecter de manière efficiente les ensembles stables dans les partitions réseaux pour le modèle de mobilité de groupe RPGM. Pour cela, la valeur de α ne doit pas dépasser la taille du groupe RPGM. Concernant les modèles de mobilité individuelle, comme les partitions réseaux possèdent une durée de vie faible, la valeur de α, de THRESHOLD ne doivent pas être trop importantes afin de permettre aux processus de

détecter des ensembles stables de petite taille.

Plusieurs perspectives nous semblent intéressantes pour compléter cette étude préliminaire. Tout d'abord, puisque la durée de la simulation est longue (3600s), il n'était pas possible de faire un grand nombre de simulations pour chaque scénario de mobilité. Nous envisageons d'accroître le nombre de simulations pour chaque scénario avec des graines différentes afin de consolider statistiquement les résultats obtenus. En outre, il nous semble également intéressant d'augmenter le nombre de valeurs pour chacun des paramètres spécifiques de l'algorithme tels que le nombre minimum de processus stables α, la période de la temporisation ${}_{part}timeout$ et le seuil de comptage des battements de cœur THRESHOLD. Ceci constitue notre première perspective.

Comme nous ne ciblons pas des scénarios particuliers, nous évaluons les performances de $\diamondsuit\mathcal{PPD}$ avec différents modèles de mobilité. Parmi les modèles de mobilité sélectionnés dans la première partie de ce chapitre, les modèles *Random Street* et *Community Based* n'ont pas été considérés dans cette étude préliminaire. L'évaluation des performances de $\diamondsuit\mathcal{PPD}$ avec les modèles *Random Street* et *Community Based* constitue donc notre seconde perspective.

Dans l'implantation du détecteur $\diamondsuit\mathcal{PPD}$, α est un paramètre important de l'application et la construction de l'ensemble stable en dépend directement. En particulier, si la valeur de α n'est pas adaptée, c'est-à-dire supérieure à la taille de la partition réseau, aucun ensemble stable ne peut être construit. Ceci confirme l'intérêt d'étudier la possibilité de rendre la valeur de α dynamique comme nous l'avons suggéré dans les perspectives du chapitre 5.

Conclusion et perspectives

Synthèse

Cette thèse étudie la gestion de groupe partitionnable en environnement réseau très dynamique tel que les MANETs. Les résultats de cette étude sont une solution de la gestion de groupe partitionnable adaptée aux systèmes répartis partitionnables à base de MANETs.

La spécification de la gestion de groupe partitionnable n'a pas encore atteint le niveau de maturité de celle de la gestion de groupe de partition primaire. Ainsi, l'étude débute par une analyse approfondie des spécifications existantes de la gestion de groupe partitionnable, en plus de la gestion de groupe de partition primaire. À partir de cette analyse, nous dégageons les problèmes des spécifications de la gestion de groupe partitionnable existantes. La spécification de la gestion de groupe partitionnable *1)* doit être assez forte pour fournir des garanties utiles aux applications réparties et *2)* doit être assez faible pour être résoluble (implantable). [Chockler et al., 2001] et [Babaoğlu et al., 2001] sont les spécifications les plus notables de la littérature. Ces travaux étendent la définition du détecteur de défaillances non fiable ultimement parfait $\Diamond\mathcal{P}$ afin d'assurer les propriétés de vivacité. Dans la première spécification, la vivacité est garantie uniquement dans les partitions ultimement complètement stables : ultimement, aucun processus n'est défaillant ou ne se recouvre, il n'y a pas de perte de message, et aucun changement dans la topologie du réseau ne se produit. Dans la deuxième spécification, la vivacité est garantie dans toutes les partitions en supposant que chaque paire de processus est reliée par un lien équitable. En outre, le modèle de système dans [Chockler et al., 2001, Babaoğlu et al., 2001] est statique dans le sens où l'ensemble Π des processus est connu et fixe, et le graphe du réseau est initialement fortement connexe.

Après l'étude approfondie des problèmes dans les spécifications de la littérature et une identification des caractéristiques des MANETs, notre objectif est la mise en œuvre d'une spécification de la gestion de groupe partitionnable qui réponde aux deux exigences exposées plus haut : *1)* la spécification doit être assez forte pour fournir des garanties utiles aux applications réparties et *2)* doit être assez faible pour être résoluble (implantable). Pour mettre en œuvre notre solution, nous procédons en trois étapes.

Tout d'abord, nous proposons un modèle de système répartir qui caractérise le comportement dynamique des partitions stables dans les MANETs. Les nœuds mobiles (processus) qui restent dans une partition pendant une période de temps assez longue sont dits stables. Un critère de stabilité basé sur le comptage des battements de cœur est proposé afin de sélectionner les

nœuds les plus stables, c'est-à-dire les nœuds qui peuvent être considérés comme faisant partie d'une éventuelle partition stable. Puisque les processus stables et instables peuvent coexister dans une partition, nous définissons une condition de stabilité de α processus stables dans une partition pour assurer la progression et la terminaison des exécutions même si la partition n'est pas complètement stable. α représente le nombre minimum de processus stables requis pour exécuter l'application. Les nœuds stables dans la même partition stable peuvent communiquer entre eux via des chemins SADDM, c'est-à-dire des séquences de liens équitables avec garantie temporelle ultime et qui sont créés dynamiquement. Un lien SADDM est plus faible qu'un lien à garantie temporelle puisqu'il autorise que des messages soient perdus et que d'autres soient transmis avec des délais de transmission non bornés. Un lien SADDM est plus fort qu'un lien équitable puisqu'il assure qu'un sous-ensemble des messages transportés soient reçus ultimement et que les messages de cet ensemble ne soient pas trop dispersés dans le temps.

Puis, nous proposons une solution pour la gestion de groupe partitionnable en adaptant Paxos pour les systèmes partitionnables. Le résultat de l'adaptation est une spécification du consensus abandonnable $\mathcal{AC}$ construit au-dessus des deux modules que sont le détecteur ultime des α participants d'une partition $\diamondsuit\mathcal{PPD}$ et le registre ultime par partition $\diamondsuit\mathcal{RPP}$. $\diamondsuit\mathcal{PPD}$ est spécifié pour abstraire la vivacité dans une partition tandis que $\diamondsuit\mathcal{RPP}$ préserve la sûreté dans la même partition. Le rôle de $\diamondsuit\mathcal{PPD}$ est de capturer le compromis entre l'accord et la progression en détectant ultimement la condition de stabilité des α processus stables dans la partition et en fournissant un leader ultime parmi ces processus. $\diamondsuit\mathcal{RPP}$ fournit la mémoire répartie (ou registre) partagée par les processus stables dans la même partition. L'acte de stocker une valeur dans le registre peut échouer dans deux cas : dans le cas de la congestion du réseau et dans le cas où la condition de stabilité n'est pas satisfaite. Le premier cas est le même que celui de l'algorithme Synod de Paxos : un proposeur abandonne sa proposition si un autre proposeur a commencé à proposer de façon concurrente. Dans ce premier cas, l'instance de consensus n'est pas abandonnée. Dans le second cas, le proposeur n'abandonne pas seulement sa proposition, mais aussi l'instance de consensus si la condition de stabilité n'est pas satisfaite. Intuitivement, cela signifie que, soit il n'existe pas α processus stables dans la partition, soit il existe au moins α processus dans la partition mais l'instant de stabilisation local à la partition n'est pas encore atteint. Ultimement, une valeur unique est écrite dans le registre de manière persistante lorsque la condition de stabilité est satisfaite.

Ensuite, la gestion de groupe partitionnable est résolue en la transformant en une séquence d'instances de $\mathcal{AC}$, où chaque instance de $\mathcal{AC}$ est exécutée par les processus participants de la vue successeur potentielle. Lorsque la décision retournée par le consensus abandonnable est un ensemble de processus αSet muni d'un identifiant, ces processus et l'identifiant constituent la vue successeur. Cependant, à la différence des consensus régulier et uniforme, la valeur retournée n'est pas nécessairement une valeur proposée par un processus. Elle peut être la valeur spécifique $(\perp, \perp)$ signifiant que l'instance de consensus a été abandonnée car la condition de stabilité n'est pas satisfaite. Dans ce cas, les processus recomposent à nouveau leur ensemble αSet et commencent à exécuter une nouvelle instance de $\mathcal{AC}$. Chacun des modules $\diamondsuit\mathcal{PPD}$, $\diamondsuit\mathcal{RPP}$, $\mathcal{AC}$, et gestion de groupe partitionnable est implanté et prouvé.

Enfin, nous analysons les performances de $\diamondsuit\mathcal{PPD}$ par simulation avec le simulateur OM-

NeT++/MiXiM couplé avec l'outil de génération et d'analyse de traces de mouvements Bonn-Motion. Dans cette étude préliminaire, nous définissons trois critères : *1)* le nombre d'ensembles de processus stables construits au cours du temps A, *2)* le nombre moyen de processus qui participent à la construction d'un ensemble de processus stables K_{moy}, et *3)* la durée de vie moyenne des ensembles de processus stables D_{moy}. Avec ces trois critères, nous évaluons les performances de l'algorithme $\diamondsuit\mathcal{PPD}$ avec les modèles de mobilité RWP, SSRWP, Gauss-Markov, Manhattan et RPGM pour les piétons. Pour les véhicules, nous considérons le modèle Manhattan. Les résultats obtenus montrent que l'algorithme $\diamondsuit\mathcal{PPD}$ respecte le comportement spécifié. En particulier, l'algorithme permet aux processus de détecter de manière efficiente les ensembles stables dans les partitions réseaux pour le modèle de mobilité de groupe RPGM. Pour cela, la valeur de α ne doit pas dépasser la taille du groupe RPGM. Concernant les modèles de mobilité individuelle, comme les partitions de réseau possèdent une durée de vie faible, les valeurs de α et de THRESHOLD ne doivent pas être trop importantes afin de permettre aux processus de détecter des ensembles stables de plus petite taille.

Perspectives

Nous concluons cette thèse en présentant quelques perspectives liées à la modélisation du système et aux solutions algorithmiques proposées.

Concernant la modélisation des MANETs, plusieurs travaux proposent des critères de stabilité basés par exemple sur le temps de présence du processus dans la partition, sur la distance entre nœuds mobiles, sur le niveau d'énergie des nœuds, sur la proximité géographique des nœuds, sur le nombre de sauts de communication, ainsi que sur la qualité de service (incluant la fiabilité, la sécurité, la performance, la bande passante, la charge de calcul et la mémoire). Nous observons que la plupart de ces critères reposent directement ou indirectement sur la connectivité du réseau. Nous avons proposé un critère de stabilité basé sur le comptage des battements de cœur afin de capturer la connectivité et la stabilité du réseau. Notre première perspective est donc d'étudier ces autres critères de stabilité et de sélectionner ceux qui seraient adaptés à notre modèle.

Au niveau algorithmique, plusieurs perspectives nous semblent intéressantes. En premier lieu, l'étude de la complexité des algorithmes proposés dans le pire des cas nous semble intéressante. Par exemple, en guise de premiers éléments, nous avons montré que, dans le pire des cas, si le groupe S est constitué de α processus stables, alors le délai de transfert d'un message d'un processus p vers un processus q dans ce groupe est au plus $\beta^{\alpha-1}\eta + (\alpha-1)\delta$ secondes car le chemin le plus long entre deux processus du groupe est au plus constitué de $\alpha - 1$ processus. β et δ sont des constantes utilisées dans la définition d'un lien SADDM et η est la période de temps maximale qui sépare deux diffusions consécutives dans le module de retransmission. Dans l'implantation du module $\diamondsuit\mathcal{PPD}$, η correspond à la valeur de la temporisation des battements de cœur. η est incrémentée si la condition de stabilité n'est pas satisfaite.

Après l'étude du pire des cas, une autre perspective est d'étudier la possibilité de décrémenter la valeur de η pour que la période de détection ne devienne trop grande. Par ailleurs, une étude au meilleur et au moyen des cas serait intéressante.

Dans notre modèle, α est une valeur fournie par l'application et correspond au nombre minimum de processus stables requis pour exécuter l'algorithme de gestion de groupe et faire progresser l'application répartie. Il nous semble intéressant d'étudier la possibilité de rendre la valeur de α dynamique. Intuitivement, munie de la valeur de αSet fournie par le module $\Diamond\mathcal{PPD}$, la couche applicative pourrait modifier la valeur de α si elle observe que le nombre de processus dans αSet devient trop important. Au contraire, si la cardinalité de l'ensemble αSet est trop grande par rapport au nombre minimum de processus stables disponibles, alors elle pourrait décider de diminuer la valeur de α afin de permettre la progression de l'application répartie avec un nombre moins important de participants. Malgré les changements de la valeur de α, $\Diamond\mathcal{PPD}$ devrait détecter de manière cohérente un ensemble de processus stables et un leader ultime parmi ces processus.

En complément de l'étude de la dynamicité de la valeur de α, nous pouvons étudier l'influence du taux de désabonnement tel qu'étudié dans les systèmes pair-à-pair. En fonction du taux de désabonnement, la couche applicative pourrait peut-être adapter la valeur de α.

Nous observons que dans le scénario d'exécution de l'algorithme du consensus Synod de Paxos, un proposeur de p_j (avec $j \in [2, 4]$) choisit un numéro de scrutin $B_{i,j}$ pour sa i^e (avec $i \geqslant 1$) proposition, qui est égal à $3(i-1)+j$. Plus généralement, s'il existe n processus dans le système qui exécutent le consensus, alors un proposeur p_j (avec $j \in [1, n]$) choisit un numéro de scrutin $B_{i,j}$ pour sa i^e proposition avec $B_{i,j} = n(i-1)+j$. Ce choix est optimal dans le sens où le pas de valeur n est optimal, c'est-à-dire qu'un proposeur ne peut pas choisir un pas de valeur $n' < n$ sans violer l'unicité des numéros de scrutins et donc altérer la correction de l'algorithme. Par analogie aux numéros de scrutins utilisés dans Synod, il nous semble intéressant d'étudier comment les proposeurs dans l'algorithme $\Diamond\mathcal{RPP}$ pourrait choisir les numéros des identifiants de la manière la plus optimale possible.

Une autre perspective concerne la communication efficace. Selon [Delporte-Gallet et al., 2001], dans un système de partition primaire composé de n processus, la communication est efficace si, ultimement, seuls n liens transportent des messages entre processus corrects et les messages transportés sont garantis d'être reçus ultimement. Le modèle de [Delporte-Gallet et al., 2001] est un modèle de système de partition primaire statique dans lequel le graphe de réseau est fortement connexe et il existe un lien bidirectionnel qui relie chaque paire de processus du système. Dans notre modèle de système adapté aux MANETs, le graphe du réseau n'est pas nécessairement fortement connexe et est partitionnable. Les nœuds stables communiquent entre eux à travers des chemins SADDM représentés par des séquences de liens SADDM unidirectionnels sans fil établis entre les différents nœuds du réseau de façon dynamique. Ainsi, la notion de communication efficace proposée par [Delporte-Gallet et al., 2001] doit être adaptée. Il nous semble intéressant d'étudier une autre forme de communication efficace pour $\Diamond\mathcal{PPD}$. Comme la communication dans les MANETs est multisaut, il est donc impossible d'assurer la communication entre les processus de αSet avec seulement $|\alpha Set|$ liens SADDM. En guise de premier élément, nous pouvons observer que dans l'implantation du module $\Diamond\mathcal{PPD}$, ultimement seul le leader de αSet continue à diffuser originellement les messages ALPHASET.

Enfin, concernant l'étude des performances de $\Diamond\mathcal{PPD}$ par simulation, plusieurs perspectives

nous semblent intéressantes pour compléter cette étude préliminaire. Tout d'abord, nous envisageons d'accroître le nombre de simulations pour chaque scénario étudié avec des graines différentes afin de consolider statistiquement les résultats obtenus. Il nous semble également intéressant d'augmenter le nombre de valeurs pour chacun des paramètres spécifiques de l'algorithme $\diamondsuit\mathcal{PPD}$ incluant le nombre minimum de processus stables α, la période de la temporisation ${}_{part}timeout$ et le seuil de comptage des battements de cœur THRESHOLD. Par ailleurs, parmi les modèles de mobilité sélectionnés, les modèles *Random Street* et *Community Based* n'ont pas été considérés dans cette étude préliminaire. Une autre perspective est d'effectuer l'étude des performances de $\diamondsuit\mathcal{PPD}$ avec ces modèles de mobilité. Enfin, nous envisageons d'étudier les performances de $\diamondsuit\mathcal{PPD}$ avec la valeur de α dynamique.

Annexe A

Analyse des résultats de simulation avec d'autres modèles de mobilité individuelle

Cette section regroupe les résultats des simulations obtenus dans l'évaluation des performances de $\diamondsuit\mathcal{PPD}$ avec les modèles de mobilité individuelle SSRWP, Gauss-Markov et Manhattan (piétons et véhicules). Rappelons que nous considérons un réseau mobile spontané composé de nœuds mobiles répartis de façon uniforme sur une surface de 1km × 1km (pour les piétons) et de 7km × 5km (pour les véhicules). La durée de la simulation est fixée à 3600s. En outre, sauf pour le modèle SSRWP, une phase initiale d'une durée de 3600s est ajoutée au début de la simulation afin d'atténuer le problème de la distribution spatiale uniforme des nœuds. Quand ce n'est pas précisé, les vitesses minimale, moyenne et maximale des nœuds, lorsqu'elles sont applicables à un modèle de mobilité donné, sont respectivement 0,5ms^{-1}, 1,0ms^{-1} et 1,5ms^{-1} (pour les piétons) et 4,17ms^{-1}, 8,89ms^{-1} et 13,89ms^{-1} (pour les véhicules). Nous considérons 3 scénarios pour chaque modèle de mobilité avec 10, 25 et 50 nœuds (pour les piétons) et 50, 100 et 150 (pour les véhicules).

Comme les résultats obtenus sont très similaires à ceux obtenus avec le modèle RWP, qui sont présentés et analysés dans la section 6.4.4, nous présentons maintenant de manière très succincte l'analyse des traces de mouvements avec BonnMotion ainsi que les résultats des simulations.

A.1 Modèle de mobilité SSRWP

Le modèle SSRWP permet un régime stationnaire dès le début de la simulation, ce qui pallie au problème de la distribution spatiale uniforme des nœuds. Ainsi, aucune phase initiale n'est ajoutée. La vitesse moyenne et le temps de pause moyen d'un nœud sont respectivement 0,5ms^{-1} et 60s. La vitesse et le temps de pause d'un nœud sont choisis selon des distributions uniformes respectivement dans les intervalles [0,5-0,1 ;0,5+0,1] et [60-5 ;60+5]. La graine aléatoire utilisée est 602171.

Le tableau A.1 présente les résultats de l'analyse des traces de mouvements générées avec le modèle SSRWP. Le tableau A.2 présente les résultats des simulations obtenus avec le modèle SSRWP.

Paramètre	Valeurs		
Portée de transmission	170m		
Nombre de nœuds	10	25	50
Degré moyen d'un nœud	1	2,90	5,49
Nombre moyen de partitions réseaux	5,94	4,87	3,48
Vitesse moyenne d'un nœud	0.45ms^{-1}	0,46ms^{-1}	0,45ms^{-1}

TABLE A.1 – Résultats de l'analyse des traces de mouvements générées avec le modèle SSRWP.

Paramètre	Valeurs								
α	2			3			5		
THRESHOLD	1	5	10	1	5	10	1	5	10
MAXHB	5	10	20	5	10	20	5	10	20
10 nœuds									
A	35	16	13	13	6	5	3	4	2
K_{moy}	2,91 ±0,27	2,63 ±0,47	2,77 ±0,55	3,92 ±0,41	3,83 ±0,60	4 ±0,60	5,33 ±0,65	6 ±1,39	5 ±0
D_{moy}	178s	368s	399s	256s	433s	482s	111s	93s	153s
${}_{part}timeout_{moy}$	3,28 ±0s	14,45 ±1,74s	20,47 ±1,57s	4,48 ±0s	18,87 ±1,80s	23,10 ±1,54s	21,71 ±1,76	22,52 1,77s	26,3 ±1,54s
25 nœuds									
A	32	15	11	22	10	4	3	1	–
K_{moy}	2,66 ±0,38	2,33 ±0,31	3 ±0,79	4,59 ±0,68	10,3 ±5,39	4,25 ±1,23	±7,33 ±1,31	6 –	– –
D_{moy}	192s	638s	432s	75s	156s	486s	81s	180s	–
${}_{part}timeout_{moy}$	3,21 ±0,07s	9,96 ±0,88s	17,48 ±0,46s	4,20 ±0,19s	11,45 ±1,11s	18 ±0,58s	10,59 ±1,66s	12,78 ±1,25s	18,79 ±0,71s
50 nœuds									
A	84	13	6	21	2	–	9	1	–
K_{moy}	2,65 ±0,25	2,31 ±0,46	2 ±0	3,80 ±0,29	4 ±1,96	– –	6,66 ±1,03	5 –	– –
D_{moy}	81s	440s	614s	54s	325s	–	35s	287s	–
${}_{part}timeout_{moy}$	3,28 ±0s	9,37 ±0,34s	17,18 ±0,17s	4,48 ±0	10,29 ±0,56s	17,5 ±0,27s	8,68 0,71s	10,61 ±0,59s	17,66 ±0,30s

TABLE A.2 – Résultats des simulations obtenus avec le modèle SSRWP.

A.2 Modèle de mobilité Gauss-Markov

Dans l'implantation du modèle Gauss-Markov de BonnMotion, la nouvelle vitesse et la direction de déplacement d'un nœud sont calculées à partir des anciennes valeurs. Nous utilisons les valeurs des écart-types de la vitesse et de la direction (angle) données par défaut : respectivement, 0,5ms^{-1} et 0,39°. La graine aléatoire utilisée est 539763.

Le tableau A.3 présente les résultats de l'analyse des traces de mouvements générées avec le modèle Gauss-Markov. Le tableau A.4 présente les résultats des simulations obtenus avec le modèle Gauss-Markov.

Paramètre	Valeurs		
Portée de transmission	170m		
Nombre de nœuds	10	25	50
Degré moyen d'un nœud	0,73	1,55	3,77
Nombre moyen de partitions réseaux	6,78	10,35	4,84
Vitesse moyenne d'un nœud	0,99ms^{-1}	0,99ms^{-1}	1.00ms^{-1}

TABLE A.3 – Résultats de l'analyse des traces de mouvements générées avec le modèle Gauss-Markov.

A.3 Modèle de mobilité Manhattan pour les piétons

La grille de Manhattan représente le plan d'une ville, par exemple Manhattan, dans lequel nous pouvons spécifier un certain nombre de blocs d'habitations selon l'axe des abscisses et l'axe des ordonnées. Nous fixons à 20 le nombre de blocs selon l'axe des abscisses et à 10 le nombre de blocs suivant l'axe des ordonnées. Pour le temps de pause maximal et la probabilité de pause, nous gardons les valeurs données par défaut dans l'implantation du modèle Manhattan de BonnMotion : respectivement, 120s et 0. De même, pour l'écart-type de la vitesse des nœuds, la probabilité qu'un nœud change de vitesse et la probabilité qu'un nœud change de direction, nous utilisons les valeurs par défaut de BonnMotion : respectivement, 0,2ms^{-1}, 0,2 et 0,5. En outre, nous utilisons la graine aléatoire donnée par défaut lors de la génération des traces de mouvements : 1346846506404.

Le tableau A.5 présente les résultats de l'analyse des traces de mouvements générés avec le modèle Manhattan pour les piétons. Le tableau A.6 présente les résultats des simulations obtenus avec le modèle Manhattan pour les piétons.

A.4 Modèle de mobilité Manhattan pour les véhicules

Les différences par rapport au modèle Manhattan pour les piétons sont les suivantes. La surface de la simulation est fixée à 7km × 5km. Comme la surface de la simulation est plus

Paramètre	Valeurs								
α	2			3			5		
THRESHOLD	1	5	10	1	5	10	1	5	10
MAXHB	5	10	20	5	10	20	5	10	20

10 nœuds									
A	27	14	15	8	5	5	1	1	1
K_{moy}	2,52 ±0,24	2,43 ±0,40	2,53 ±0,42	3,25 ±0,32	3,6 ±0,48	3,6 ±0,48	6 –	6 –	6 –
D_{moy}	241s	328s	374s	245s	447s	462s	74s	247s	140s
${}_{part}timeout_{moy}$	3,28 ±0s	15,6 ±1,16s	21,74 ±1,98s	4,48 ±0s	20,5 ±1,29s	25,13 ±1,87s	26,06 ±0,53s	27,24 ±0,54s	30,74 ±0,53s

25 nœuds									
A	123	45	22	46	30	15	35	19	11
K_{moy}	3,45 ±0,30	3,91 ±0,56	3,68 ±0,74	4,30 ±0,42	4,76 ±0,64	4,6 ±0,74	6,31 0,31	6,26 ±0,52	6 ±0,46s
D_{moy}	129s	260s	424s	159s	220s	280S	80S	117s	176s
${}_{part}timeout_{moy}$	3,18 ±0,11s	10,56 ±0,85s	17,71 ±0,45s	4,46 ±0,24s	12,38 ±1,25s	18,42 ±0,73s	13,82 ±1,34s	14,83 ±1,22s	19,47 ±0,85s

50 nœuds									
A	107	13	4	60	9	2	11	3	2
K_{moy}	3,38 ±0,31	3,31 ±1,26	3,25 1,89	5,56 ±0,69	6,44 ±2,29	4,5 ±2,93	7,18 ±1,21	11 ±8,84	7 1,96
D_{moy}	51s	298s	871s	47s	233s	540s	55s	271s	363s
${}_{part}timeout_{moy}$	3,27 ±0,01	8,95 ±0,27	16,98 ±0,07	4,38 ±0,06	9,20 ±0,36	17,07 ±0,14	7,42 ±0,69	9,93 ±0,44	17,14 ±0,13

TABLE A.4 – Résultats des simulations obtenus avec le modèle Gauss-Markov.

Paramètre	Valeurs		
Portée de transmission	170m		
Nombre de nœuds	10	25	50
Degré moyen d'un nœud	0,60	1,63	3,65
Nombre moyen de partitions réseaux	7,15	9,40	5,29
Vitesse moyenne d'un nœud	0,95ms^{-1}	0,96ms^{-1}	0,96ms^{-1}

TABLE A.5 – Résultats de l'analyse des traces de mouvements générées avec le modèle Manhattan pour les piétons.

importante, nous considérons trois scénarios avec des nombres de nœuds plus importants : 50, 100 et 150. La grille de Manhattan possède 35 et 33 blocs respectivement selon l'axe des abscisses et l'axe des ordonnées. La graine aléatoire utilisée est 423853.

Le tableau A.7 présente les résultats de l'analyse des traces de mouvements générées avec

Paramètre	Valeurs								
α	2			3			5		
THRESHOLD	1	5	10	1	5	10	1	5	10
MAXHB	5	10	20	5	10	20	5	10	20

10 nœuds									
A	18	13	11	4	4	5	2	2	2
K_{moy}	2,38 ±0,28	2,54 ±0,36	2,82 ±0,58	3,25 ±0,49	3,25 ±0,49	3,6 ±0,78	5,5 ±0,98	5,5 ±0,98	5,5 ±0,98
D_{moy}	251s	448s	557s	395s	302s	355s	442s	436s	412s
${}_{part}timeout_{moy}$	3,28 ±0s	16,42 ±0,93s	23,53 ±0,68s	4,48 ±0	21,74 ±0,19	26,75 ±0,52	25,64 ±0,20	26,84 ±0,21	30,32 ±0,20

25 nœuds									
A	133	43	20	81	26	9	29	9	4
K_{moy}	4,32 ±0,39	3,58 ±0,53	3,15 ±0,78	5,61 ±0,52	5,04 ±0,77	5 ±0,80	6,93 ±0,65	7,11 ±1,28	5,5 ±0,57
D_{moy}	96s	262s	367s	79s	90s	141s	76s	121s	182s
${}_{part}timeout_{moy}$	3,28 ±0,11	11,28 ±0,87	17,96 ±0,50	4,48 ±0	13,93 ±1,25	19,15 ±0,69	14,30 ±1,51	15,63 ±1,27	20,25 ±0,78

50 nœuds									
A	104	25	6	34	19	2	18	2	1
K_{moy}	3,86 ±0,60	3,44 ±0,67	3,16 ±1,28	4,50 ±0,86	4,52 ±0,78	3,5 ±0,98	6,28 ±0,59	7 ±1,96	9 –
D_{moy}	69s	221s	323s	87s	170s	623s	46s	145s	98s
${}_{part}timeout_{moy}$	3,28 ±0s	8,88 ±0,19s	16,98 ±0,07s	4,37 ±0,06s	9,25 ±0,36s	17,07 ±0,11s	7,86 ±0,82s	10,26 ±0,59s	17,39 ± 0,27s

TABLE A.6 – Résultats des simulations obtenus avec le modèle Manhattan pour les piétons.

le modèle Manhattan pour les véhicules. Le tableau A.8 présente les résultats des simulations obtenus avec le modèle Manhattan pour les véhicules.

Paramètre	Valeurs		
Portée de transmission	170m		
Nombre de nœuds	50	100	150
Degré moyen d'un nœud	0,10	0,24	0,37
Nombre moyen de partitions réseaux	47,34	88,30	123,83
Vitesse moyenne d'un nœud	8,88ms^{-1}	8,88ms^{-1}	8,88ms^{-1}

TABLE A.7 – Résultats de l'analyse des traces de mouvements générées avec le modèle Manhanttan pour les véhicules.

Paramètre	Valeurs								
α	2			3			5		
THRESHOLD	1	5	10	1	5	10	1	5	10
MAXHB	5	10	20	5	10	20	5	10	20

50 nœuds									
A	176	104	60	19	11	8	1	–	–
K_{moy}	2,07 ±0,04	2,08 ±0,05	2,07 ±0,06	3 ±0	3 ±0	3 ±0	5 ±	–	–
D_{moy}	176s	309s	420s	322s	684S	772s	12s		
${}_{part}timeout_{moy}$	3.28 ±0s	16,8 ±0s	27,57 ±0,51s	4,48 ±0s	22,3 ±0s	30,65 ±0s	26,87 ±0,25s	28,10 ±0,01s	31,58 ±0,01s

100 nœuds									
A	777	366	197	168	67	31	1	1	2
K_{moy}	2,17 ±0,03	2,12 ±0,04	2,11 ±0,05	3,17 ±0,07	3,15 ±0,96	3,06 ±0,88	5 –	5 –	5 ±0
D_{moy}	81s	137s	230s	106s	131s	193s	541	530	1220s
${}_{part}timeout_{moy}$	3.28 ±0s	16,79 ±0,01s	23,81 ±0,41s	4,48 ±0s	22,29 ±0,02s	28,1 ±0,33s	26,65 ±0,08s	27,77 ±0,11s	31,03 ±0,16s

150 nœuds									
A	1762	588	250	531	132	71	11	7	4
K_{moy}	2,29 ±0,03	2,15 ±0,03	2,12 ±0,04	3,38 ±0,06	3,12 ±0,06	3,08 ±0,06	5,10 ±138s	5,28 ±0,55	5 ±0
D_{moy}	53s	106s	179s	56s	118s	145s	138s	187s	294s
${}_{part}timeout_{moy}$	3,28 ±0s	16,33 ±0,14s	21,34 ±0,29s	4,48 ±0s	21,6 ±0,18s	25,22 ±0,32s	25,97 ±0,16s	26,87 ±0,16s	29,51 ±0,24s

TABLE A.8 – Résultats des simulations obtenus avec le modèle Manhattan pour les véhicules.

Bibliographie

[Aguilera and Toueg, 1997] Aguilera, M.-K., C. W. and Toueg, S. (1997). Heartbeat : A Timeout-Free Failure Detector for Quiescent Reliable Communication. In *Proceeding of the Workshop Distributed Algorithms (WDAG)*, pages 126–140. 56, 108

[Aguilera et al., 2003] Aguilera, M., Delporte-Gallet, C., Fauconnier, H., and Toueg, S. (2003). On implementing Omega with weak reliability and synchrony assumptions. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing*, pages 306–314, New York, USA. 35

[Aguilera, 2004] Aguilera, M.-K. (2004). A Pleasant Stroll through the Land of Infinitely Many Creatures. *SIGACT News*, 35(2) :36–59. 42

[Aguilera, 2010] Aguilera, M.-K. (2010). Stumbling over Consensus Research : Misunderstandings and Issues. In *Replication*, pages 59–72. 4, 20

[Aguilera et al., 1999] Aguilera, M.-K., Chen, W., and Toueg, S. (1999). Using the Heartbeat Failure Detector for Quiescent Reliable Communication and Consensus in Partitionable Networks. *Theoretical Computer Science*, 220(1) :3–30. 43, 85, 92, 108, 110

[Aguilera et al., 2000] Aguilera, M.-K., Chen, W., and Toueg, S. (2000). On Quiescent Reliable Communication. *SIAM Journal on Computing*, 29(6) :2040–2073. 28, 35, 82, 92

[Aguilera et al., 2004] Aguilera, M.-K., Delporte-Gallet, C., Fauconnier, H., and Toueg, S. (2004). Communication-Efficient Leader Election and Consensus with Limited Link Synchrony. In *Proceeding of the 23rd ACM Symposium on Principles of Distributed Computing*, pages 328–337, St. John's, Newfoundland, Canada. 48

[Alekeish and Ezhilchelvan, 2011] Alekeish, K. and Ezhilchelvan, P. (2011). Consensus in Sparse, Mobile Ad Hoc Networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(3) :467–474. 42

[Amir et al., 1997] Amir, Y., G.V., C., Dolev, D., and Vitenberg, R. (1997). Efficient State Transfer in Partitionable Environments. In *Proceeding of the 2nd European Research Seminar on Advances in Distributed Systems*, pages 183–192. 23

[Anceaume et al., 1995] Anceaume, E., Charron-Bost, B., Minet, P., and Toueg, S. (1995). On the Formal Specification of Group Membership Services. Technical Report TR 95-1534, Department of Computer Science, Cornell University, Ithaca, New-York (USA). 5, 8, 25

[Arantes et al., 2010] Arantes, L., Sens, P., Thomas, G., Conan, D., and Lim, L. (2010). Partition Participant Detector with Dynamic Paths in MANETs. In *Proceedings of the 9th IEEE International Symposium on Network Computing and Applications*, Cambridge, MA, USA. 44, 110

[Aschenbruck et al., 2010] Aschenbruck, N., Ernst, R., Gerhards-Padilla, E., and Schwamborn, M. (2010). BonnMotion : A mobility scenario generation and analysis tool. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, pages 51 :1–51 :10. 116, 124, 133

[Aschenbruck et al., 2007] Aschenbruck, N., Gerhards-Padilla, E., Gerharz, M., Frank, M., and Martini, P. (2007). Modelling Mobility in Disaster Area Scenarios. In *Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems*, pages 4–12. 127, 134

[Aschenbruck and Schwamborn, 2010] Aschenbruck, N. and Schwamborn, M. (2010). Synthetic Map-based Mobility Traces for the Performance Evaluation in Opportunistic Networks. In *Proceedings of the Second International Workshop on Mobile Opportunistic Networking*, pages 143–146, New York, USA. 124, 134

[Attiya and Bar-Noy, 1995] Attiya, H. and Bar-Noy, A.and Dolev, D. (1995). Sharing Memory Robustly in Message-Passing Systems. *Journal of the ACM*, 42 :124–142. 110

[Babaoğlu et al., 2001] Babaoğlu, O., Davoli, R., and Montresor, A. (2001). Group Communication in Partitionable Systems : Specification and Algorithms. *IEEE Transactions on Software Engineering*, 27(4) :308–336. vi, 4, 5, 8, 9, 16, 17, 20, 21, 22, 24, 26, 27, 29, 31, 108, 109, 163

[Babaoğlu et al., 1998] Babaoğlu, O., Davoli, R., Montresor, A., and Segala, R. (1998). System Support for Partition-Aware Network Applications. *ACM SIGOPS Operating Systems Review*, 32 :41–56. 22

[Badarneh and Kadoch, 2009] Badarneh, O.-S. and Kadoch, M. (2009). Multicast Routing Protocols in Mobile Ad Hoc Networks : A Comparative Survey and Taxonomy. *EURASIP J. Wirel. Commun. Netw.*, 2009 :26 :1–26 :42. 120

[Bagrodia et al., 1998] Bagrodia, R., Takai, M., Chen, Y.-A., Zeng, X., and Martin, J. (1998). Parsec : A Parallel Simulation Environment for Complex Systems. *IEEE Computer*, 31 :77–85. 139

[Bai et al., 2003] Bai, F., Sadagopan, N., and Helmy, A. (2003). IMPORTANT : A framework to systematically analyze the Impact of Mobility on Performance of RouTing protocols for Adhoc NeTworks. In *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications*, pages 825–835. 125, 134

[Baldoni et al., 2009a] Baldoni, R., Bonomi, S., Kermarrec, A.-M., and Raynal, M. (2009a). Implementing a Register in a Dynamic Distributed System. *proceedings of the 29th IEEE International Conference on Distributed Computing Systems*, 0 :639–647. 110, 111

[Baldoni et al., 2009b] Baldoni, R., Bonomi, S., and Raynal, M. (2009b). Regular Register : an Implementation in a Churn Prone Environment. In *Proceeding of the 16th International Colloquium on Structural Information and Communication Complexity*, pages 15–29. 111

[Bar-noy and Kessler, 1994] Bar-noy, A. and Kessler, I. (1994). Mobile Users : To Update or not to Update ? In *Proceedings of the 13th Proceedings IEEE Networking for Global Communications*, pages 570 – 576, Toronto, Canada. 120

[Basile et al., 2003] Basile, C., Killijian, M.-O., and Powell, D. (2003). A Survey of Dependability Issues in Mobile Wireless Networks. Technical report, LASS CNRS Toulouse, France. 1

[Bettstetter et al., 2004] Bettstetter, C., Hartenstein, H., and Pérez-Costa, X. (2004). Stochastic Properties of the Random Waypoint Mobility Model. *Wireless Network*, 10 :555–567. 120

[Bettstetter et al., 2003] Bettstetter, C., Resta, G., and Santi, P. (2003). The Node Distribution of the Random Waypoint Mobility Model for Wireless Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 2 :257–269. 120

[Birman and Joseph, 1987] Birman, K.-P. and Joseph, T.-A. (1987). Exploiting Virtual Synchrony in Distributed Systems. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, pages 123–138, Austin, USA. 22

[Blakely and Lowekamp, 2004] Blakely, K. and Lowekamp, B. (2004). A structured group mobility model for the simulation of mobile ad hoc networks. In *Proceedings of the second international workshop on Mobility management & wireless access protocols*, pages 111–118, New York, USA. 127

[Boichat et al., 2003a] Boichat, R., Dutta, P., Frølund, S., and Guerraoui, R. (2003a). Deconstructing Paxos. *ACM SIGACT News Distributed Computing Column*, 34(1) :47–67. 49, 55, 56, 58, 59, 66, 67, 74, 108

[Boichat et al., 2003b] Boichat, R., Dutta, P., Frølund, S., and Guerraoui, R. (2003b). Reconstructing Paxos. *ACM SIGACT News Distributed Computing Column*, 34 :42–57. 49, 55, 67

[BonnMotion, 2011] BonnMotion (2011). *BonnMotion—a Mobility Scenario Generation and Analysis Tool*. University of Bonn. 134

[Boulkenafed et al., 2005] Boulkenafed, M., Sacchetti, D., and Issarny, V. (2005). Using Group Management to Tame Mobile Ad Hoc Networks. In *Mobile Information Systems, IFIP TC 8 Working Conference on Mobile Information Systems*, pages 245–260. 4, 5, 45, 112

[Brewer, 2000] Brewer, E.-A. (2000). Towards robust distributed systems (abstract). In *Proceedings of the 19th annual ACM Symposium on Principles of Distributed Computing*, page 7. 2

[Briesemeister and Hommel, 2006] Briesemeister, L. and Hommel, G. (2006). Localized Group Membership Service for Ad Hoc Networks. In *International Conference on Parallel Processing Workshops*, pages 94–100. 4

[Bui-Xuan et al., 2003] Bui-Xuan, B.-M., Ferreira, A., and Jarry, A. (2003). Computing Shortest, Fastest, and Foremost Journeys in Dynamic Networks. *International Journal of Foundations of Computer Science*, 14(2) :267–285. 43

[Camp et al., 2002] Camp, T., Boleng, J., and Davies, V. (2002). A Survey of Mobility Models for Ad Hoc Network Research. *Wireless Communication & Mobile Computing : Special Issue on Mobile Ad Hoc Networking : Research, Trend and Application*, 2 :483–502. 117, 118, 120, 121, 122, 125, 126, 127, 134

[Cavin et al., 2004] Cavin, D., Sasson, Y., and Schiper, A. (2004). Consensus with Unknown Participants or Fundamental Self-Organization. In *Proceedings of the 3rd International Conference on AD-HOC Networks & Wireless*, number 3158 in Lecture Notes in Computer Science, pages 135–148, Vancouver, British Columbia, Canada. Springer-Verlag. 43, 109

[Chaintreau et al., 2005] Chaintreau, A., Hui, P., Crowcroft, J. adn Diot, C., Gass, R., and Scoot, J. (2005). Pocket Switched Networks and Human Mobility in Conference Environments. Research Report ucam-cl-tr-617, Computer Laboratory, University of Cambridge. 117

[Chaintreau et al., 2007] Chaintreau, A., Hui, P., Crowcroft, J., Diot, C., Gass, R., and Scott, J. (2007). Impact of Human Mobility on Opportunistic Forwarding Algorithms. *IEEE Transactions on Mobile Computing*, 6 :606–620. 120

[Chandra et al., 1996a] Chandra, T.-D., Hadzilacos, V., and Toueg, S. (1996a). The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4). 48, 59

[Chandra et al., 1996b] Chandra, T.-D., Hadzilacos, V., Toueg, S., and Charron-Bost, B. (1996b). On the Impossibility of Group Membership. In *Proceedings of the 15th ACM Symposium on Principles of Distributed Computing*, Philadeplhia, USA. 5, 8, 48, 61

[Chandra and Toueg, 1996] Chandra, T.-D. and Toueg, S. (1996). Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2) :225–267. 16, 38, 47, 48, 82, 110

[Chen and Kotz, 2000] Chen, G. and Kotz, D. (2000). A Survey of Context-Aware Mobile Computing Research. Technical report, Hanover, NH, USA. 3

[Chen, 2006] Chen, W. (2006). Abortable Consensus and Its Application to Probabilistic Atomic Broadcast. Technical Report MSR-TR-2006-135, Microsoft Research. 49

[Chiang, 1998] Chiang, C. (1998). *Wireless Network Multicasting*. PhD thesis, University of California. 123, 134

[Chockler et al., 2001] Chockler, G.-V., Keidar, I., and Vitenberg, R. (2001). Group Communication Specifications : A Comprehensive Study. *ACM Computing Surveys*, 33(4) :427–469. v, 4, 5, 7, 8, 9, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 31, 44, 61, 108, 109, 110, 163

[CMU Monarch project, 1999] CMU Monarch project (1999). The CMU Monarch Projects Wireless and Mobility Extension to ns. 138

[Conan et al., 2008] Conan, D., Sens, P., Arantes, L., and Bouillaguet, M. (2008). Failure, Disconnection and Partition Detection in Mobile Environment. In *Proceedings of the 7th IEEE International Symposium on Network Computing and Applications*, pages 119–127. 110

[Coutaz et al., 2005] Coutaz, J., Crowley, J., Dobson, S., and Garlan, D. (2005). The disappearing computer : Context is Key. *Communications of the ACM*, 48(3) :49–53. 3

[Davies, 2000] Davies, V.-A. (2000). Evaluating mobility models within an ad hoc network. Master's thesis, Colorado School of Mines. 118, 125, 127

[De et al., 2005] De, P., Raniwala, A., Sharma, S., and Chiueh, T. (2005). Design Considerations for a Multihop Wireless Network Testbed. In *IEEE Communication Magazine*. 116

[Delporte-Gallet et al., 2001] Delporte-Gallet, C., Fauconnier, H., Toueg, S., and Aguilera, M.-K. (2001). Stable Leader Election. In *Proceedings of the 15th International Conference on Distributed Computing*, pages 108–122. Springer-Verlag. 113, 114, 166

[Dolev et al., 1987] Dolev, D., Dwork, C., and Stockmeyer, L. (1987). On the Minimal Synchronism Needed for Distributed Consensus. *Journal of the ACM*, 34(1). 4, 48

[Dricot and Doncker, 1992] Dricot, J.-M. and Doncker, P.-D. (1992). High-Accuracy physical layer model for wireless network simulations in ns-2. In *Proceedings of the International Workshop on Wireless Ad-Hoc Networks*, pages 249–253. 138, 139

[Dunkels et al., 2004] Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462. 138

[Dwork et al., 1988] Dwork, C., Lynch, N.-A., and Stockmeyer, L. (1988). Consensus in the Presence of Partial Synchrony. *Journal of the ACM*, 35(2) :288–323. 4, 35, 48

[Fall et al., 2000] Fall, K., Varadhan, K., and the VINT project (2000). *The ns Manual*. 138

[Feeley et al., 2004] Feeley, M., Hutchinson, N., and Ray, S. (2004). Realistic Mobility for Mobile Ad Hoc Network Simulation. *Ad-Hoc, Mobile, and Wireless Networks*, pages 324–329. 118

[Fernández et al., 2006] Fernández, A., Jiménez, E., and Raynal, M. (2006). Eventual Leader Election with Weak Assumptions on Initial Knowledge, Communication Reliability, and Synchrony. In *Proceeding of the 7th IEEE International Conference on Dependable Systems and Networks*, pages 166–178, Philadelphia, PA, USA. 110

[Filali et al., 2006a] Filali, M., Issarny, V., Mauran, P., Padiou, G., and Quéinnec, P. (2006a). Maximal Group Membership in Ad Hoc Networks. In *6th International Conference on Parallel Processing and Applied Mathematics*, pages 51–58, Poznan, Poland. 4, 5, 111, 112

[Filali et al., 2006b] Filali, M., Issarny, V., Mauran, P., Padiou, G., and Quéinnec, P. (2006b). Maximal Group Membership in Ad Hoc Networks. In *Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics*, pages 51–58, Berlin, Heidelberg. Springer-Verlag. 5

[Fischer et al., 1985] Fischer, M.-J., Lynch, N.-A., and Paterson, M. (1985). Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2) :374–382. 48

[Forman and Zahorjan, 1994] Forman, G.-H. and Zahorjan, J. (1994). The Challenges of Mobile Computing. *Computer*, 27 :38–47. 1

[Franceschetti and Bruck, 2001] Franceschetti, M. and Bruck, J. (2001). A group membership algorithm with a practical specification. *IEEE Transactions on Parallel and Distributed Systems*, 12(11) :1190–1200. 5, 8

[Friedman and Tcharny, 2009] Friedman, R. and Tcharny, G. (2009). Evaluating failure detection in mobile ad-hoc networks. *International Journal of Pervasive Computing and Communications*, 5(4) :476–496. 149

[Gafni and Lamport, 2003] Gafni, E. and Lamport, L. (2003). Disk paxos. *Distributed Computing*, 16 :1–20. 49

[García et al., 2009] García, J.-C., Beyer, S., and Galdámez, P. (2009). A Stability Criteria Membership Protocol for Ad Hoc Networks. In *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems*, pages 690–707, Berlin, Heidelberg. Springer-Verlag. 4, 5, 8, 45

[Garetto and Leonardi, 2006] Garetto, M. and Leonardi, E. (2006). Analysis of Random Mobility Models with Partial Differential Equations. In *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 73–84, New York, USA. 120

[Gilbert and Lynch, 2002] Gilbert, S. and Lynch, N.-A. (2002). Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *SIGACT News*, 33(2) :51–59. 2

[Greve et al., 2011] Greve, F., Sens, P., Arantes, L., and V., S. (2011). A Failure Detector for Wireless Networks with Unknown Membership. In *Proceedings of the 17th Proceedings of the 17th international conference on Parallel processing*, pages 27–38, Berlin, Heidelberg. Springer-Verlag. 42, 109, 110

[Greve and Tixeuil, 2007] Greve, F. and Tixeuil, S. (2007). Knowledge Connectivity vs. Synchrony Requirements for Fault-Tolerant Agreement in Unknown Networks. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 82–91. 43, 109

[Guerin, 1987] Guerin, R. (1987). Channel Occupancy Time Distribution in a Cellular Radio System. In *IEEE Transactions on Vehicular Technology*, pages 89–99. 120, 134

[Guerraoui and Rodrigues, 2006] Guerraoui, R. and Rodrigues, L. (2006). *Introduction to Reliable Distributed Programming*. Springer-Verlag, NJ, USA. 9, 10, 49

[Guerraoui and Schiper, 1997] Guerraoui, R. and Schiper, A. (1997). Consensus : The big misunderstanding. In *Proceedings of the 6th IEEE Computer Society Workshop on Future Trends in Distributed Computing Systems*, pages 183–188, Tunis, Tunisia. IEEE Computer Society. 4, 20

[Guerraoui and Schiper, 2001] Guerraoui, R. and Schiper, A. (2001). The Generic Consensus Service. *IEEE Transactions on Software Engineering*, 27(1) :29–41. 6, 48, 61

[Haas, 1997] Haas, Z. (1997). A New Routing Protocol For The Reconfigurable Wireless Networks. In *In Proceeding of the IEEE International Conference on Universal Personal Communications*, pages 562–566, San Diego, CA , USA. 122

[Hähner and Dudkowski, 2007] Hähner, J. and Dudkowski, D. (2007). Quantifying Network Partitioning in Mobile Ad Hoc Networks. In *Proceedings of the International Conference on Mobile Data Management*, pages 174–181. 2

[Heidemann et al., 2001] Heidemann, J., Bulusu, N., Elson, J., Intanagonwiwat, C., Lan, K., Xu, Y., Ye, W., Estrin, D., and Govindan, R. (2001). Effects of Detail in Wireless Network Simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 3–11. 116, 137

[Hermant and Le Lann, 2002] Hermant, J.-F. and Le Lann, G. (2002). Fast Asynchronous Uniform Consensus in Real-Time Distributed systems. *IEEE Transactions on Computers*, 51 :931–944. 42

[Hogie et al., 2006] Hogie, L., Bouvry, P., and Guinand, F. (2006). An Overview of MANETs Simulation. *Electronic Notes Theorical Computer Science*, 150 :81–101. 116, 137, 140, 141

[Hong et al., 1999] Hong, X., Gerla, M., Pei, G., and Chiang, C.-C. (1999). A group Mobility Model for Ad Hoc Wireless Networks. In *Proceedings of the 2nd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 53–60. 127, 134

[Hu and Dittmann, 2009] Hu, L. and Dittmann, L. (2009). Heterogeneous Community-Based Mobility Model for Human Opportunistic Network. In *Proceedings of the 2009 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 465–470, Washington, DC, USA. 119

[Hui et al., 2005] Hui, P., Chaintreau, A., Scott, J., Gass, R., Crowcroft, J., and Diot, C. (2005). Pocket Switched Networks and Human Mobility in Conference Environments. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 244–251, New York, NY, USA. 117

[Johnson and Maltz, 1996] Johnson, D.-B. and Maltz, D. A. (1996). Dynamic Source Routing in Ad Hoc Wireless Networks. *Mobile Computing*, pages 153–181. 120, 134

[Karagiannis et al., 2007] Karagiannis, T., Le Boudec, J.-Y., and Vojnović, M. (2007). Power Law and Exponential Decay of Inter Contact Times between Mobile Devices. In *Proceedings of the 13th annual ACM international conference on Mobile computing and networking*, pages 183–194, New York, NY, USA. 120

[Khazan, 2004] Khazan, R.-I. (2004). Group membership : A novel Approach and the First Single-Round Algorithm. In *Proceedings of the 23 ACM Symposium on Principles of Distributed Computing*, pages 347–356, New York, NY, USA. ACM. 4, 16

[Kleijnen, 1999] Kleijnen, J.-P.-C. (1999). VALIDATION OF MODELS : STATISTICAL TECHNIQUES AND DATA AVAILABILITY. In *Proceedings of the 31st conference on Winter simulation : Simulation*, pages 647–654, New York, NY, USA. 117

[Ko-Steven et al., 2008] Ko-Steven, Y., Hoque, I., and Gupta, I. (2008). Using Tractable and Realistic Churn Models to Analyze Quiescence Behavior of Distributed Protocols. In *Proceedings of the 2008 Symposium on Reliable Distributed Systems*, pages 259–268. 110

[Köpke et al., 2008] Köpke, A., Swigulski, M., Wessel, K., Willkomm, D., Haneveld, P.-T. K., Parker, T.-E.-V., Visser, O.-W., Lichte, H.-S., and Valentin, S. (2008). Simulating wireless and mobile networks in OMNeT++ the MiXiM vision. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems*

& workshops, pages 71 :1–71 :8, ICST, Brussels, Belgium, Belgium. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering. 140, 147

[Kraaier and Killat, 2005] Kraaier, J. and Killat, U. (2005). The Random Waypoint City Model - User Distribution in a Street-Based Mobility Model for Wireless Network Simulations. In *Proceedings of the 3rd ACM international workshop on Wireless mobile applications and services on WLAN hotspots*, pages 100–103, New York, USA. 124

[Kurkowski et al., 2005] Kurkowski, S., Camp, T., and Colagrosso, M. (2005). MANET Simulation studies : The Incredibles. *SIGMOBILE Mobile Computing and Communications Review*, 9(4) :50–61. 127, 139

[Kvaksrud, T.I., 2008] Kvaksrud, T.I. (2008). Range Mesurements in an Open Field Environment. 146

[Lamport, 1978] Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7) :558–565. 4, 11

[Lamport, 1986] Lamport, L. (1986). On interprocess communication - part i and ii. *Distributed Computing*, 1(2) :77–101. 110

[Lamport, 1998] Lamport, L. (1998). The Part Time Parliament. *ACM Transactions on Computer Systems*, 16(2) :133–169. 6, 49, 55

[Lamport, 2001] Lamport, L. (2001). Paxos Made Simple. *ACM SIGACT News Distributed Computing Column*, 32(4) :18–25. 6, 49, 51, 53, 54, 55

[Lamport, 2006] Lamport, L. (2006). Fast Paxos. *Distributed Computing*, 19(2) :79–103. 49

[Lamport and Massa, 2004] Lamport, L. and Massa, M. (2004). Cheap Paxos. In *Proceedings of the 2004 IEEE International Conference on Dependable Systems and Networks*, page 307. 49

[Lassila et al., 2005] Lassila, P., Hyytiä, E., and Koskinen, H. (2005). Connectivity properties of random waypoint mobility model for ad hoc networks. In *Proceedings of MedHoc-Net, Île de Porquerolles*, pages 159–168. 120

[Le Boudec and Vojnovic, 2006] Le Boudec, J.-Y. and Vojnovic, M. (2006). The Random Trip Model : Stability, Stationary Regime, and Perfect Simulation. *IEEE/ACM Transaction on Networking*, 14 :1153–1166. 121

[Lee et al., 2009] Lee, K., Hong, S., Joon, S.-K., Rhee, I., and Chong, S. (2009). Slaw : A New Human Mobility Model. In *Proceedings of the 28th Conference on Computer Communications*, pages 855–863. 134

[Liang and Haas, 2003] Liang, B. and Haas, Z.-J. (2003). Predictive Distance-Based Mobility Management for Multidimensional PCS Networks. *IEEE/ACM Transactions on Networking*, 11 :718–732. 122, 134

[Lim et al., 2006] Lim, S., Yu, C., and Das, C. (2006). Clustered Mobility Model for Scale-Free Wireless Networks. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, pages 231–238. 129, 132

[López and Manzoni, 2001] López, M. S. and Manzoni, P. (2001). ANEJOS : a Java based simulator for ad hoc networks. *Future Generation Computer Systems*, 17(5) :573–583. 126, 134

[MacDougall, 1987] MacDougall, M.-H. (1987). *Simulating Computer Systems : Techniques and Tools*. MIT Press, Cambridge, MA, USA. 137

[McCanne and Floyd, 1989] McCanne, S. and Floyd, S. (1989). ns Network Simulator. 138

[Merritt and Taubenfeld, 2000] Merritt, M. and Taubenfeld, G. (2000). Computing with Infinitely Many Processes Under Assumptions on Concurrency and Participation. In *Proceedings of the 14th International Symposium on Distributed Computing*, pages 164–178. 33

[Milic et al., 2005] Milic, B., Milanovic, N., and Malek, M. (2005). Prediction of Partitioning in Location-Aware Mobile Ad Hoc Networks. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, pages 306–3, Washington, DC, USA. 3, 4

[MiXiM project, 2012] MiXiM project (2012). http://mixim.sourceforge.net. 140

[Mostefaoui et al., 2005] Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., and El Abbadi, A. (2005). From Static Distributed Systems to Dynamic Systems. In *Proceeding of the 24th IEEE Symposium on Reliable Distributed Systems*, pages 109–118, Florianpolis (Brazil). 28, 38, 42, 109

[Musolesi and Mascolo, 2007] Musolesi, M. and Mascolo, C. (2007). Designing Mobility Models Based on Social Network Theory. *ACM SIGMOBILE Mobile Computing and Communications Review*, 11 :59–70. 117, 130, 131, 132

[Navidi et al., 2004] Navidi, Q., Camp, T., , and Bauer, N. (2004). Improving the Accuracy of Random Waypoint Simulations Through Steady-State Initialization. In *Proceedings of the 15th International Conference on Modeling and Simulation*, pages 319–326. 134

[Navidi and Camp, 2004] Navidi, W. and Camp, T. (2004). Stationary Distributions for the Random Waypoint Mobility Model. *IEEE Transactions on Mobile Computing*, 3 :99–108. 120

[Neis and Zipf, 2009] Neis, P. and Zipf, A. (2009). OpenLS Route Service. 124

[Nesterenko and Schiper, 2007] Nesterenko, M. and Schiper, A. (2007). On Properties of the Group Membership Problem. Technical Report TR-KSU-CS-2007-01, Kent State University. 109

[OMNeT++ project, 2012] OMNeT++ project (2012). http://www.omnetpp.org/. 140

[OpenStreetMap Community, 2009] OpenStreetMap Community (2009). OpenStreetMap - The Free WiKi World Map. 124

[Österlind et al., 2006] Österlind, F., Dunkels, A., Eriksson, J., Finne, N., and Voigt, T. (2006). Cross-Level Sensor Network Simulation with COOJA. In *Proceeding of the First IEEE International Workshop on Pratical Issues in Building Sensor Network Applications*. 138

[Papageorgiou et al., 2009] Papageorgiou, C., Birkos, K., Dagiuklas, T., and Kotsopoulos, S. (2009). Simulating Mission Critical Mobile Ad Hoc Networks. In *Proceedings of the 4th ACM workshop on Performance monitoring and easurement of heterogeneous wireless and wired networks*, pages 143–150, New York, USA. 128

[Pelov, 2009] Pelov, A. (2009). *MOBILITY MODELS FOR WIRELESS NETWORKS*. PhD thesis, Université de Starbourg. 118

[Pérez-Costa et al., 2003] Pérez-Costa, X., Bettstetter, C., and Hartenstein, H. (2003). Toward a Mobility Metric for Reproducible & Comparable Results in Ad Hoc Networks Research. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7 :58–60. 120

[Piórkowski et al., 2008] Piórkowski, M., Sarafijanovic-Djukic, N., and Grossglauser, M. (2008). On Clustering Phenomenon in Mobile Partitioned Networks. In *Proceeding of the 1st ACM SIGMOBILE workshop on Mobility models*, pages 1–8, New York, USA. 132

[Pleisch et al., 2008] Pleisch, S., Rütti, O., and Schiper, A. (2008). On the Specification of Partitionable Group Membership. In *Proceedings of the 7th European Dependable Computing Conference*, pages 37–45, Kaunas, Lithuania. 5, 8, 22, 25, 26, 27, 35, 109

[Pomportes et al., 2011] Pomportes, S., Tomasik, J., and Vèque, V. (2011). A Composite Mobility Model for Ad Hoc Networks in Disaster Areas. *Journal on Electronics and Communications*, 1(1) :62–68. 128

[Rhee et al., 2011] Rhee, I., Shin, M., Hong, S., Lee, K., Kim, S.-J., and Chong, S. (2011). On the Levy-Walk Nature of Human Mobility. *IEEE/ACM Transactions on Networking*, 19 :630–643. 128

[Roman et al., 2001] Roman, G.-C., Huang, Q., and Hazemi, A. (2001). Consistent Group Membership in Ad Hoc Networks. In *Proceedings of the 23rd ACM International Conference on Software Engineering*, pages 381–388. 3, 4, 5, 8

[Royer et al., 2001] Royer, M.-E., Melliar-Smith, P.-M., and Moser, L.-E. (2001). An Analysis of the Optimum Node Density for Ad Hoc Mobile Networks. In *In Proceedings of the IEEE International Conference on Communications*, pages 857–861. 121, 134

[Sastry and Pike, 2007] Sastry, S. and Pike, S. (2007). Eventually Perfect Failure Detectors Using ADD Channels. In *In Proceedings of the 5th international Symposium on Parallel and Distributed Processing and Applications (2007)*, pages 483–496, Nigara Falls, Canada. 35, 44, 110

[Scalable Network Technologies, 2008] Scalable Network Technologies, I. (2008). *Qualnet user manual*. 139

[Schiper, 2004] Schiper, A. (2004). Group Communication : Where are we today and future challenges. In *Proceeding of the 4th IEEE International Symposium on Network Computing and Applications*, pages 109–117, Washington, DC, USA. 6, 9, 35, 48

[Schiper, 2006] Schiper, A. (2006). Dynamic group communication. *Distributed Computing*, 18(5) :359–374. 6, 8, 48

[Schiper and Toueg, 2006] Schiper, A. and Toueg, S. (2006). From Set Membership to Group Membership : A Separation of Concerns. *IEEE Transactions on Dependable and Secure Computing*, 3(1) :2. 5, 8

[Schneider, 1990] Schneider, F.-B. (1990). Implementing Fault-Tolerant Services Using the State Machine Approach : A Tutorial. *ACM Computing Surveys*, 22(4) :299–319. 53

[Sekercioglu et al., 2003] Sekercioglu, Y.-A., Varga, A., and Egan, G.-K. (2003). Parallel simulation made easy with omnet++. In *Proceedings of the 15th European Simulation Symposium*. 141

[Spyropoulos et al., 2006] Spyropoulos, T., Psounis, K., and Raghavendra, C.-S. (2006). Performance Analysis of Mobility-assisted Routing. In *Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 49–60, New York, USA. 130, 132

[Srivastava et al., 2008] Srivastava, V., Hilal, A., Thompson, M.-S., Chattha, J.-N., Mackenzie, A.-B., and Dasilva, L.-A. (2008). Characterizing Mobile Ad Hoc Networks — The MANIAC Challenge Experiment. In *Proceedings of 3rd ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and CHaracterization (WiNTECH)*. 1, 2

[Tian et al., 2002] Tian, J., Haehner, J., Becker, C., Stepanov, I., and Rothermel, K. (2002). Graph-Based Mobility Model for Mobile Ad Hoc Network Simulation. In *Proceedings of the 35th Annual Simulation Symposium*, pages 337–, Washington, DC, USA. IEEE Computer Society. 123

[Toley, 1999] Toley, V. (1999). Load reduction in ad hoc networks using mobile servers. Master's thesis, Colorado Shcool of Mines. 122

[Tucci-Piergiovanni and Baldoni, 2010] Tucci-Piergiovanni, S. and Baldoni, R. (2010). Eventual Leader Election in Infinite Arrival Message-Passing System Model with Bounded Concurrency. In *Proceedings of the 2010*, pages 127–134, Washington, DC, USA. IEEE Computer Society. 42, 43, 110

[Varga, 2001] Varga, A. (2001). The OMNeT++ Discrete Event Simulation System. In *Proceedings of the European Simulation Multiconference*. 140

[Varga, 2006] Varga, A. (2006). TWiki. Simulation. OMNeTppComparison. 137

[Varga and Pongor, 1997] Varga, A. and Pongor, G. (1997). Flexible Topology Description Language for Simulation Programs. In *Proceedings of the 9th European Simulation Symposium*. 141

[Wang and Baochun, 2002] Wang, K. and Baochun, L. (2002). Group Mobility and Partition Prediction in Wireless Ad-Hoc Networks. In *Proceedings of the IEEE International Conference on Communications*, pages 1017–1021. 3, 4, 127

[Yeo et al., 2006] Yeo, J., Kotz, D., and Henderson, T. (2006). CRAWDAD : A Community Resource for Archiving Wireless Data at Dartmouth. *ACM SIGCOMM Computer Communication Review*, 36(2) :21–22. 117

[Yoon et al., 2003] Yoon, J., Liu, M., and Noble, B. (2003). Random Waypoint Considered Harmful. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1312–1321. 121

[Zeng et al., 1998] Zeng, X., Bagrodia, R., and Gerla, M. (1998). GloMoSim : A Library for Parallel Simulation of Large-scale Wireless Networks. In *Proceedings of the 12th Workshop on Parallel and Distributed Simulation*, pages 154–161. 139

[Zhang et al., 2009] Zhang, Y., Low, C.-P., Ng, J., and Wang, T. (2009). An Efficient Group Partition Prediction Scheme for MANETs. In *Proceedings of the 2009 IEEE conference on Wireless Communications & Networking Conference*, pages 2816–2821. 3, 4

[Zhou et al., 2004] Zhou, B., Xu, K., and Gerla, M. (2004). Group and Swarm Mobility Models for Ad Hoc Network Scenarios Using Virtual Tracks. In *Proceeding of Military Communications Conference*, pages 289–294. 128, 129

[Zonoozi and Dassanayake, 1997] Zonoozi, M.-M. and Dassanayake, P. (1997). User Mobility Modeling and Characterization of Mobility Patterns. *IEEE Journal on Selected Areas in Communications*, 15(7) :1239–1252. 120

Résumé

Dans les réseaux mobiles spontanés (en anglais, *Mobile Ad hoc NETworks* ou MANETs), la gestion de groupe partitionnable est un service de base permettant la construction d'applications réparties tolérantes au partitionnement, c'est-à-dire dans lesquelles plusieurs groupes de processus évoluent concurremment et indépendamment les uns des autres. En deux décennies, plusieurs spécifications de ce service ont été proposées. Toutefois, aucune de ces spécifications ne satisfait les deux exigences antagonistes suivantes : *1)* elle doit être assez forte pour fournir des garanties utiles aux applications réparties dans les systèmes partitionnables ; *2)* elle doit être assez faible pour être résoluble (implantable). Dans cette thèse, nous proposons une spécification et une implantation de la gestion de groupe partitionnable en environnement réseaux très dynamiques tels que les MANETs.

Après une étude approfondie des problèmes dans les spécifications de la littérature et une identification des caractéristiques des MANETs, nous proposons un modèle de système qui caractérise la stabilité dans les MANETs. Les nœuds mobiles (processus) qui restent dans une partition pendant une période de temps assez longue sont dits stables. Un critère de stabilité basé sur le comptage des battements de cœur est proposé afin de sélectionner les nœuds les plus stables, c'est-à-dire les nœuds faisant partie d'une éventuelle partition stable. Puisque les processus stables et instables peuvent coexister dans une partition, nous définissons une condition de stabilité de α processus stables dans une partition pour assurer la progression et la terminaison des exécutions, et ceci même si la partition n'est pas complètement stable.

Ensuite, nous adaptons pour les systèmes partitionnables l'approche Paxos à base de consensus Synod. Cette adaptation résulte en la spécification d'un consensus abandonnable $\mathcal{AC}$ construit au-dessus d'un détecteur ultime des α participants d'une partition $\diamondsuit\mathcal{PPD}$ et d'un registre ultime par partition $\diamondsuit\mathcal{RPP}$. $\diamondsuit\mathcal{PPD}$ garantit la vivacité dans une partition même si la partition n'est pas complètement stable tandis que $\diamondsuit\mathcal{RPP}$ préserve la sûreté dans la même partition. Le rôle de $\diamondsuit\mathcal{PPD}$ est de capturer le compromis entre l'accord et la progression en détectant ultimement la condition de stabilité et en fournissant un leader parmi ces processus. $\diamondsuit\mathcal{RPP}$ fournit la mémoire répartie (ou registre) partagée dans la même partition.

Enfin, la gestion de groupe partitionnable est résolue en la transformant en une séquence d'instances de $\mathcal{AC}$, où chaque instance de $\mathcal{AC}$ est exécutée par les processus participants dans la vue successeur potentielle. Quand la décision est retournée par le consensus abandonnable est un ensemble de processus αSet muni d'un identifiant, ces processus et l'identifiant constituent la vue successeur. La décision peut être la valeur spécifique $(\perp, \perp)$ signifiant que l'instance de consensus a été abandonnée car la condition de stabilité n'est pas satisfaite. Chacun des modules $\diamondsuit\mathcal{PPD}$, $\diamondsuit\mathcal{RPP}$, $\mathcal{AC}$ et gestion de groupe partitionnable est implanté et prouvé. Par ailleurs, nous analysons les performances de $\diamondsuit\mathcal{PPD}$ par simulation.

Mots-clés: MANETs, systèmes partitionnables, gestion de groupe partitionnable, consensus

abandonnable.

Abstract

In Mobile Ad hoc NETworks or MANETs, partitionable group membership is a basic service for building partition-tolerant applications—*i.e.,* severals groups of processes evolve concurrently and independently of each other. Since two decades, severals partitionable group membership specifications have been proposed. However, none of them satisfies the two following antagonistic requirements : *1)* it must be strong enough to simplify the design of partition-tolerant distributed applications in partitionable systems ; *2)* it must be weak enough to be implantable. This thesis studies partitionable group membership in very dynamic network environment such as MANETs.

After an in-depth study of the problems in the specifications proposed in the literature and identifying the characteristics of MANETs, we propose a solution to the problem of partitionable group membership by adapting Paxos for such systems. To this means, we develop a system model that characterises stability in MANETs. Mobile nodes (processes) that stay in a partition during a period that lasts enough are said to be stable. A stability criterion based on heartbeats counting is also proposed to select the most stable nodes—*i.e.,* nodes that could be part of a potential stable partition. Since stable and unstable nodes can coexist in the context of a partition, we define a weak stability condition of α stable processes in a partition which guarantees the progress and termination of executions even if the partition is not completely stable.

Then, the adaptation of Paxos results in a specification of abortable consensus $\mathcal{AC}$ which is constructed on the eventual α partition-participants detector $\diamondsuit\mathcal{PPD}$ and the eventual register per partition $\diamondsuit\mathcal{RPP}$. $\diamondsuit\mathcal{PPD}$ guarantees liveness in a partition even if the partition is non completely stable whereas $\diamondsuit\mathcal{RPP}$ ensures safety in the same partition. The role of $\diamondsuit\mathcal{PPD}$ is to make trade-offs between agreement and progress by eventually detecting the stability condition, and eventually providing the leader among them. $\diamondsuit\mathcal{RPP}$ provides a distributed storage (or register) in a partition.

Finally, partitionable group membership is solved by transforming it into a sequence of abortable consensus instances $\mathcal{AC}$, where each $\mathcal{AC}$ instance is executed by the participant nodes in the potential successor view. When the decision returned by the abortable consensus is a set of processes α-Set associated with an identifier, these processes and the identifier constitute the successor view. The decision could be a specific value $(\perp, \perp)$ meaning that the consensus has aborted because the stability condition is not satisfied. Each of the modules $\diamondsuit\mathcal{PPD}$, $\diamondsuit\mathcal{RPP}$, $\mathcal{AC}$, and partitionable group membership is implanted and proved. Next, we analyse the performances of $\diamondsuit\mathcal{PPD}$ by simulation.

Keywords: MANETs, dynamic partitionable systems, partitionable group membership, abortable consensus.