



Université Joseph Fourier — Master 2 Recherche - Systèmes et Logiciels

---

# Protocoles de diffusion totalement ordonnée pour les systèmes distribués synchrones

Projet réalisé par :
Willy MALVAULT

---

Soutenu le :
**18 juin 2007**

Équipe Sardes
Inria Rhône-Alpes - Laboratoire d'Informatique de Grenoble

---

Encadrant :
Vivien Quéma

JURY :

| | |
|---|---|
| M.Laurent Besacier | Membre du jury permanent |
| M.Rachid Echahed | Membre du jury permanent |
| M.Jean-Marc Vincent | Membre du jury permanent |
| Mme.Vania Marangozova-Martin | Examinatrice externe |
| M.Vivien Quéma | Encadrant |

**Résumé**

Dans le domaine des systèmes distribués, la diffusion totalement ordonnée est une primitive de communication très utilisée. En effet, cette primitive est très utile dans le cadre de la tolérance aux fautes, ou encore pour la construction de systèmes utilisant une mémoire partagée distribuée. Dans ce rapport, nous nous intéressons à la conception de primitives de diffusion totalement ordonnée pour les systèmes synchrones. Notre objectif est d'améliorer les performances (débit, latence et complexité en messages) des protocoles existants. Nous proposons deux protocoles : le premier protocole, appelé SCR, est optimal en débit et en complexité en messages, mais a une latence linéaire. Le second protocole, appelé SPA, est optimal en latence et en nombre de messages et garantit un débit proche de l'optimal.

**Mots clés :** *Diffusion totalement ordonnée, Systèmes distribués synchrones, Evaluation théorique de performances, Algorithmique distribuée, Débit, Latence.*

## Remerciements

Tout d'abord, je tiens à remercier vivement mon encadrant Vivien Quéma, pour son investissement dans mon projet de M2R, ainsi que pour sa disponibilité et pour sa confiance. Je remercie également Michaël Lienhardt pour sa relecture rigoureuse et ses conseils précieux, qui m'ont été utiles lors de la rédaction de ce rapport. Je remercie enfin Jean-Bernard et l'équipe SARDES de L'INRIA Rhônes-Alpes pour avoir apporté le financement nécessaire à ce projet, ainsi que pour leur accueil chaleureux et leur soutient au cours de ces quatres derniers mois.

# Table des matières

# Introduction

Ce document décrit les travaux réalisés lors de mon stage de Master 2 Recherche que j'ai effectué au sein du projet SARDES : "System Architecture for Reflexive Distributed EnvironmentS" de l'INRIA et du laboratoire LIG. Le projet SARDES focalise ses recherches sur des méthodes de construction d'intergiciels et d'architectures pour les systèmes distribués. Allant des systèmes pair à pair à grande échelle aux systèmes distribués sur puce, les domaines d'applications considérés par le projet SARDES sont nombreux et hétérogènes. Le travail présenté dans ce rapport appartient au domaine de l'algorithmique distribuée, qui étudie les modèles théoriques de systèmes distribués et propose des algorithmes permettant aux différents noeuds composant un tel système de collaborer à la réalisation de tâches variées. Ce travail considère un sous-ensemble des systèmes distribués : les *systèmes distribués synchrones*, dans lesquels des hypothèses de synchronie sont faites sur les temps de traitement des différentes tâches et sur les temps de communication entre les noeuds.

Dans le cadre des systèmes distribués synchrones, nous nous intéressons à une primitive de communication appelée *diffusion totalement ordonnée*. Une diffusion totalement ordonnée permet à l'ensemble des noeuds du système distribué de s'échanger des messages et garantit que tous les noeuds délivreront les messages dans le même ordre. La diffusion totalement ordonnée est particulièrement utilisée dans le domaine de la tolérance aux fautes, où elle est utilisée pour répliquer l'état de machines dans le but d'offrir un service hautement disponible. Elle est également utilisée par les Bases de Données réparties et les systèmes utilisant une mémoire partagée, afin d'assurer un ordre total sur les modifications réalisées par les différents noeuds du système sur un même fichier.

Dans ce projet de Master, notre objectif est de réaliser un protocole de diffusion totalement ordonnée plus performant que ceux existant aujourd'hui. Nous définissons la performance d'un protocole de diffusion totalement ordonnée à l'aide de trois métriques : la *latence* qui correspond au temps nécessaire à la réalisation de la diffusion d'un message dit "utile", la *complexité en messages* qui correspond au nombre de messages nécessaires pour effectuer une diffusion, et le *débit* qui correspond au nombre moyen de messages qu'un protocole permet de délivrer par unité de temps. Nous présentons deux protocoles : SCR et SPA. Le protocole SCR est optimal en débit et en complexité en messages, mais n'obtient qu'une latence linéaire en fonction du nombre de noeuds. Le protocole SPA est très légèrement moins bon en débit. En revanche, il est optimal en complexité en messages et en latence.

Ce rapport est organisé comme suit. Nous présentons, dans le premier chapitre, la définition d'un système distribué synchrone, puis la définition d'une diffusion totalement ordonnée, ainsi que les grandes classes de protocoles qui permettent de l'implanter. Nous expliquons également comment les performances d'une diffusion totalement ordonnée peuvent être évaluées de façon théorique. Dans le second chapitre, nous dressons l'inventaire des travaux proposant des protocoles de diffusion totalement ordonnée. Nous évaluons de façon théorique les performances des

diffusions que proposent ces protocoles et nous effectuons une synthèse, visant à comprendre la variation des performances de diffusion existant entre ces différents protocoles. Dans les quatrième et cinquièmes chapitres, nous présentons les protocoles SCR et SPA. Enfin, nous concluons ce rapport.

# Chapitre 1

# Systèmes synchrones et diffusion totalement ordonnée

Ce premier chapitre est consacré à la définition de notions utilisées par la suite dans ce rapport. Nous définissons tout d'abord ce que nous entendons par *système distribué synchrone*. Nous donnons ensuite la définition formelle d'une diffusion totalement ordonnée et décrivons les grandes classes de protocoles implantant une diffusion totalement ordonnée. Enfin, nous terminons ce chapitre en expliquant la façon dont les performances des protocoles implantant une diffusion totalement ordonnée seront évalués.

## 1.1 Système distribué synchrone

Un système distribué est constitué d'un ensemble de noeuds reliés par un système de communication. Ces noeuds ont des fonctions de traitements (processeurs), de stockage (mémoires) et de relation avec le monde extérieur (capteurs, actionneurs). Les différents noeuds peuvent être identifiés de manière unique. Par ailleurs, ils ne fonctionnent pas indépendamment, mais collaborent à une ou plusieurs tâches communes, ce qui a pour conséquence qu'une partie au moins de l'état global du système est partagé entre plusieurs noeuds. Nous nous intéressons aux systèmes distribués dans lesquels cette collaboration s'effectue par l'intermédiaire d'échanges de messages. Par ailleurs, le système de communication permettant ces échanges de messages peut prendre diverses formes : de réseaux locaux (LAN pour *Local Area Network*) à des réseaux de grande échelle, de type pair-à-pair (P2P). Ce système de communication est supposé parfait : les messages échangés entre deux noeuds qui ne sont pas en panne arrivent systématiquement à destination.

Dans le cadre de ce rapport, nous nous intéressons à une catégorie spéciale de systèmes distribués, dits *synchrones*. La particularité des systèmes distribués synchrones est la présence d'hypothèses temporelles sur le temps d'exécution des traitements effectués par les noeuds, ainsi que sur les délais de propagation des messages échangés par les noeuds. Plus précisément, nous qualifions un système distribué de synchrone lorsqu'il vérifie les deux propriétés suivantes [10] :

- **Calcul synchrone :** il existe une borne supérieure connue sur le temps d'exécution des traitements de chaque noeud. Un traitement est un ensemble d'actions. Parmi les actions qu'un noeud peut effectuer, citons la réception, le traitement, ou encore l'envoi de messages.

– **Communications synchrones :** il existe une borne supérieure connue sur le délai de transmission d'un message entre deux noeuds. C'est à dire que la période entre le moment où un noeud envoie un message et le moment où son destinataire le reçoit est toujours inférieure à cette borne.

Les noeuds d'un système distribué peuvent être l'objet de fautes. Il existe différents types de fautes [10, 13]. Dans le cadre de ce rapport, nous considérons quatre types de fautes :

– **Faute par panne franche :** un noeud commet une faute par panne franche lorsqu'il cesse de fonctionner. Un noeud fautif ne participe plus au système distribué. Il peut uniquement être redémarré sous un autre identifiant.
– **Faute par omission :** un noeud commet une faute par omission lorsqu'il oublie d'effectuer une certaine action (envoi de message, traitement, etc.).
– **Faute temporelle :** un noeud commet une faute temporelle lorsqu'il viole une des propriétés citées précédemment (calcul et communication synchrones).
– **Faute byzantine :** un noeud commet une faute byzantine lorsqu'il a un comportement arbitraire, malicieux ou non. Un exemple typique de faute byzantine est la modification indue du contenu d'un message.

Notons que ces fautes peuvent être classifiées : les fautes par panne franche sont un cas particulier des fautes par omission. Ces dernières sont un cas particulier des fautes temporelles, qui sont elles-mêmes des cas particuliers des fautes byzantines. Ainsi, un protocole tolérant les fautes byzantines tolérera tous les autres types de fautes.

Par ailleurs, du fait de la validité des deux propriétés mentionnées précédemment, il est possible de mettre en oeuvre différents services, utiles pour les protocoles distribués. Les deux services principaux sont les suivants :

– **Détecteur de panne parfait :** ce service permet de détecter les pannes franches des noeuds du système. De façon informelle, un détecteur de panne est dit parfait lorsque (1) il détecte toutes les pannes, et (2) il ne détecte jamais de panne n'ayant pas eu lieu. Par ailleurs, le détecteur peut garantir que toute panne sera détectée dans un laps de temps borné et connu.
– **Horloges synchronisées :** ce service consiste à synchroniser les horloges physiques des différents noeuds du système. Il permet de garantir que ces horloges ne différeront pas de plus d'une constante connue.

Nous allons maintenant nous intéresser à une méthode de communication particulière utilisée dans les systèmes distribués : la diffusion totalement ordonnée.

## 1.2 Diffusion totalement ordonnée

### 1.2.1 Diffusion

La diffusion (*broadcast*) permet d'effectuer un envoi de message(s) à un ensemble de noeuds. Elle est mise en œuvre à l'aide de deux primitives :

– **Broadcast**($m$) : cette primitive est invoquée par un noeud pour envoyer un message $m$ à un ensemble de noeuds. Par défaut, tous les noeuds du système sont destinataires (y compris l'émetteur). Néanmoins, cette interface peut être étendue dans certains systèmes pour prendre en paramètre un ensemble de destinataires.
– **Deliver**($m$) : cette primitive est invoquée pour livrer à chaque noeud destinataire un message $m$ préalablement reçu.

Notons qu'il est important de différencier la réception d'un message de sa livraison (ou délivrance) à l'aide la primitive *Deliver(m)*. En effet, des traitements peuvent être effectués entre ces deux instants. Un exemple de traitement est l'ordonnancement des messages reçus. La délivrance des messages reçus peut donc être retardée afin de respecter un ordre dans la délivrance des messages. Dans ce travail, nous nous intéressons à un type particulier de diffusion : la diffusion totalement ordonnée.

### 1.2.2 Ordre total dans une diffusion

La diffusion totalement ordonnée a été introduite par Leslie Lamport [12]. Depuis lors, elle a été l'objet de nombreux travaux [7]. De façon intuitive, une primitive de diffusion totalement ordonnée garantit que les messages diffusés sont délivrés dans le même ordre par chacun des noeuds. La Figure 1.1 illustre un exemple d'exécution de diffusion totalement ordonnée. Dans cet exemple, le système est composé de trois noeuds : $n_0$, $n_1$ et $n_2$. Le noeud $n_0$ diffuse un message $m_1$, puis le noeud $n_2$ diffuse un message $m_2$. Ces messages ne sont pas reçus dans l'ordre d'émission par tous les noeuds. Cependant, tous les noeuds délivrent les messages dans le même ordre.

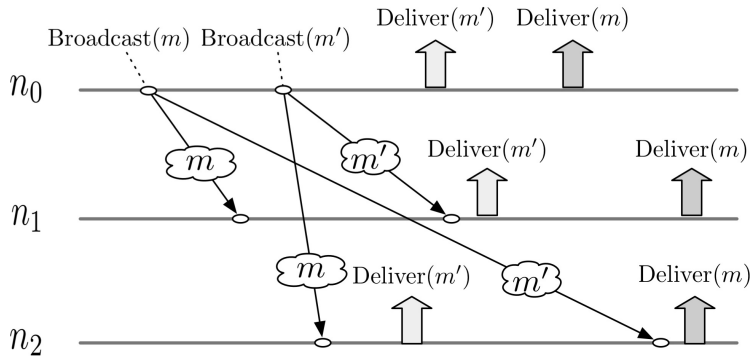


FIG. 1.1 – Exemple de diffusion totalement ordonnée.

De façon formelle, la diffusion totalement ordonnée est définie par les quatre propriétés suivantes :

- **Validité** : si un noeud correct diffuse un message $m$, alors il le délivrera au bout d'un temps fini.
- **Intégrité** : pour tout message $m$, chaque noeud $n$ délivre $m$ au plus une fois, et seulement si $m$ a été préalablement diffusé par un noeud.
- **Accord** : si un noeud correct délivre un message $m$, alors tous les noeuds corrects délivreront $m$ au bout d'un temps fini.
- **Ordre total** : s'il existe un noeud correct qui délivre un message $m$ avant un message $m'$, alors chaque noeud correct délivra $m'$ seulement aprés avoir délivré $m$.

### 1.2.3 Protocoles de diffusion totalement ordonnée

Dans cette section, nous présentons les grandes classes de protocoles de diffusion totalement ordonnée. Celles-ci sont au nombre de cinq [7].

**Protocoles utilisant un séquenceur fixe**

Dans cette catégorie de protocoles, un noeud particulier, appelé séquenceur, est responsable de l'ordonnancement des messages. Lorsqu'un noeud veut diffuser un message, il doit au préalable l'envoyer au séquenceur, qui affecte ensuite un numéro de séquence au message. Aucun autre noeud ne peut décider de l'ordre donné a un message. En cas de panne du séquenceur, un autre noeud est élu pour remplir ce rôle. Il existe trois variantes de cette classe de protocoles (voir Figure 1.2) :

- **Unicast-Unicast-Broadcast (UUB)** : dans cette variante, le noeud émetteur envoie son message $m$ au séquenceur qui lui attribut un numéro de séquence *seq(m)*. L'émetteur diffuse ensuite le couple *(m,seq(m))* aux noeuds destinataires.
- **Unicast-Broadcast (UB)** : dans cette variante, le noeud émetteur envoie son message $m$ au séquenceur qui lui attribut un numéro de séquence *seq(m)*. Le séquenceur diffuse ensuite le couple *(m,seq(m))* aux noeuds destinataires.
- **Broadcast-Broadcast (BB)** : dans cette variante, l'émetteur diffuse le message $m$ directement aux destinataires (y compris le séquenceur). Dans un second temps, le séquenceur diffuse aux destinataires le numéro de séquence *seq(m)* attribué au message $m$.

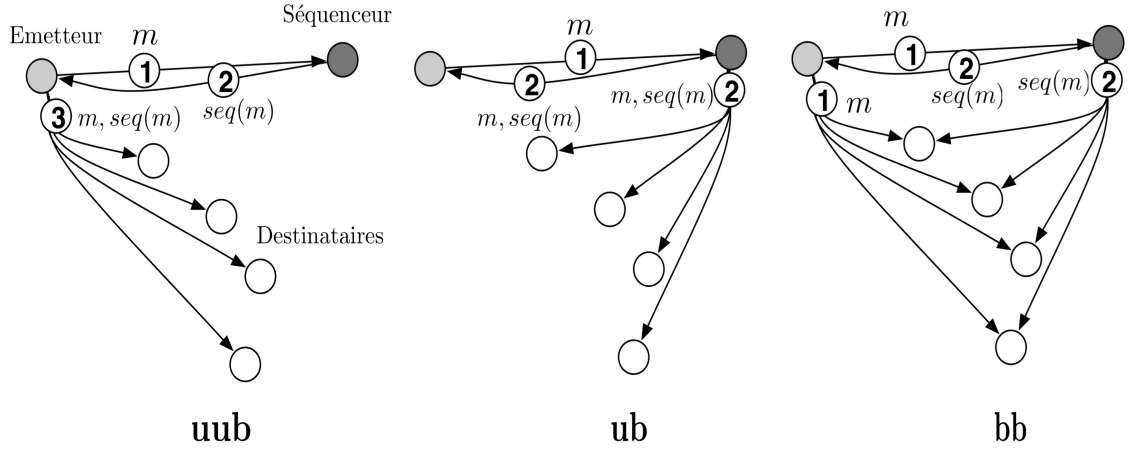


FIG. 1.2 – Les trois variantes du séquenceur fixe.

**Protocoles utilisant un séquenceur mobile**

Les protocoles utilisant un séquenceur mobile reprennent le principe des protocoles à séquenceur fixe, tout en permettant de partager le rôle de séquenceur entre plusieurs noeuds. Ce principe est illustré sur la Figure 1.3 : cette figure illustre le fait que le séquenceur est choisi parmi plusieurs noeuds. L'interêt d'utiliser un séquenceur mobile est de distribuer la charge exercée par les communications sur le séquenceur afin d'éviter de créer un gouleau d'étranglement. Le protocole exécuté par chaque séquenceur est cependant un peu plus compliqué que celui des séquenceurs fixes. Pour diffuser un message, un émetteur doit l'envoyer aux séquenceurs, qui font circuler un jeton portant le numéro de séquence à attribuer au prochain message reçu, ainsi qu'une liste de tous les messages auxquels un numéro de séquence à déja été attribué. Dés la réception du jeton, un séquenceur attribut un numéro de séquence

à tous les messages qu'il connait et qui n'en possèdent pas encore. Il diffuse ensuite les messages auxquels il a attribué un numéro de séquence aux noeuds du système, met à jour le jeton, puis transmet ce dernier au séquenceur suivant. Il est possible d'adapter les protocoles à séquenceur mobile selon les trois variantes présentées pour le séquenceur fixe. Cependant la grande majorité des protocoles à séquenceur mobile utilisent la variante "*Broadcast Broadcast*" (BB).

Notons que les protocoles à séquenceur fixe sont souvent préférés aux séquenceur mobile pour trois raisons principales :

- Le séquenceur fixe est beaucoup plus facile à réaliser, laissant moins de place aux erreurs d'implantation.
- La latence des diffusions obtenue par les protocoles à séquenceur fixe est souvent meilleure.
- Dans de nombreuses architectures distribuées, il y a souvent une machine plus puissante et moins propice aux pannes, qui est parfaitement adaptée à l'implantation d'un séquenceur fixe.

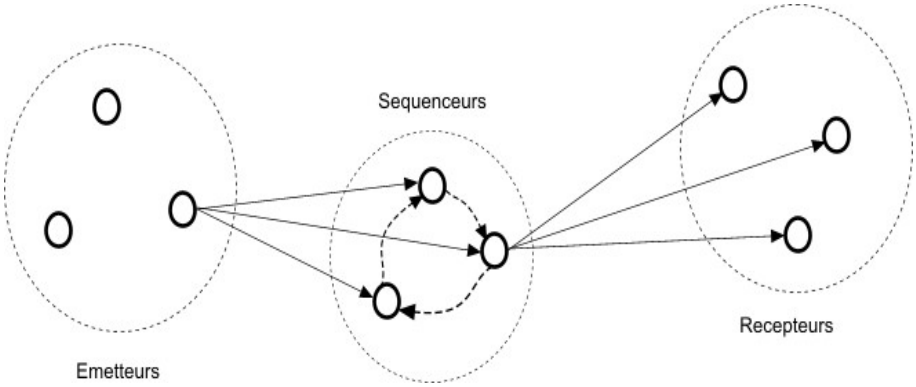


FIG. 1.3 – Protocole utilisant un séquenceur mobile.

### Protocoles basés sur un privilège

Dans les protocoles basés sur un privilège, un émetteur ne peut diffuser un message que lorsqu'il y est autorisé. L'ordre est déterminé par les noeuds émetteurs lorsqu'ils diffusent leurs messages. Ce privilège d'envoyer et d'ordonner des messages circule de façon exclusive (un seul noeud à la fois peut l'obtenir) parmi les noeuds émetteurs. Il est représenté par un jeton qui est porteur du dernier numéro de séquence utilisé. Le principe de cette classe de protocoles est illustré sur la Figure 1.4. Un jeton circule entre les émetteurs, portant le numéro de séquence à associer au prochain message à diffuser. Lorsqu'un émetteur veut diffuser un message, il doit attendre la réception du jeton, associer un numéro de séquence en cohérence avec celui du jeton, puis enfin mettre à jour le jeton et le transmettre à l'émetteur suivant. Les noeuds déstinataires délivrent ainsi les messages par ordre de numéro de séquence croissant.

### Protocoles utilisant l'historique des communications

Dans les protocoles basés sur l'historique des communications, les émetteurs peuvent diffuser des messages à tout instant. Chaque message porte une estampille horaire correspondant

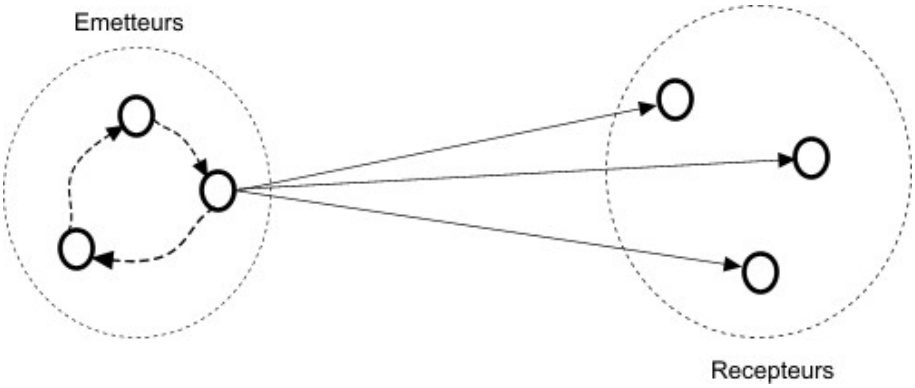


FIG. 1.4 – Protocole basé sur un privilège.

au moment où il a été diffusé. Les destinataires observent les messages reçus et leurs estampilles, et définissent un historique des communications qui leur permet de s'assurer lors de chaque livraison que l'ordre total est respecté. La Figure 1.5 illustre le principe de ces protocoles. Lorsqu'un noeud d'identifiant $n_i$ décide de diffuser un message $m$, il l'estampille avec son horloge locale $H$ puis diffuse l'ensemble $m, H, n_i$. Une fois les messages reçus par les destinataires, ils sont délivrés dans l'ordre de leurs estampilles. Dans le cas où deux messages possèdent la même estampille, l'ordre total est obtenu en utilisant l'ordre lexicographique sur l'identifiant des émetteurs.

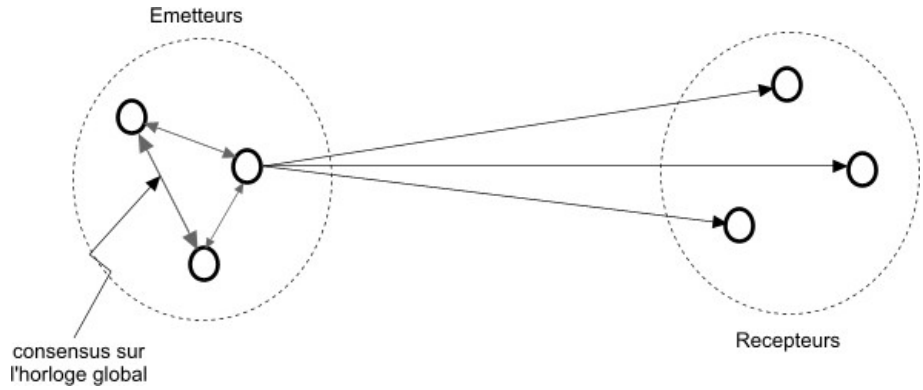


FIG. 1.5 – Protocole basé sur l'historique des communications.

### Protocoles utilisant un accord des destinataires

Dans cette classe de protocoles, l'ordre total se fait par un accord des destinataires, le plus souvent réalisé à l'aide d'un consensus. Le principe de ces protocoles est illustré sur la Figure 1.6. Il existe trois variantes principales pour ces protocoles : le consensus sur un numéro de séquence d'un message, le consensus sur un ensemble de message à délivrer et le consensus sur un ordre de message. Cette classe de protocoles ne sera pas détaillée davantage, car aucun protocole étudié dans ce rapport ne l'utilise.

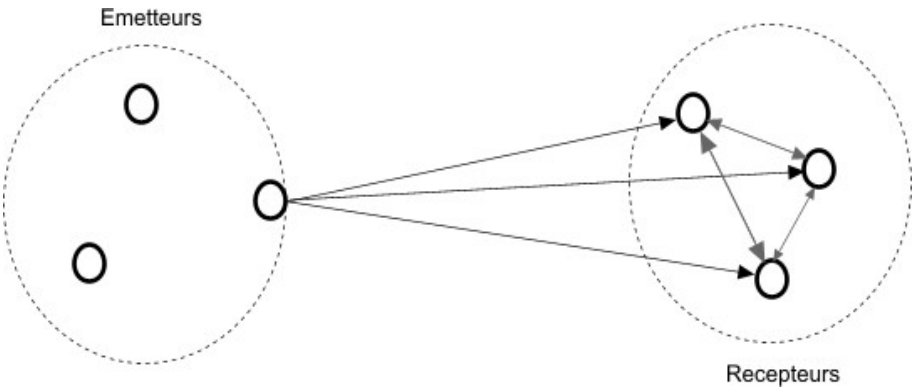


FIG. 1.6 – Protocole utilisant l'accord des destinataires.

## 1.3 Evaluation d'une diffusion

Dans le travail présenté dans ce rapport, nous nous intéressons à l'élaboration de protocoles de diffusion totalement ordonnée efficaces. Pour déterminer l'efficacité d'un protocole, il est nécessaire de définir des métriques, ainsi qu'un modèle dans lequel ces métriques sont évaluées. Nous commençons cette section par une définition de ce modèle et de ces métriques. Nous présentons ensuite la façon dont ces dernières seront évaluées dans ce rapport.

### 1.3.1 Modèle de communication et métriques de performances

Etant donné que nous nous intéressons aux systèmes synchrones, il est naturel d'utiliser un *modèle de ronde synchrone* qui peut être défini de la façon suivante :

1. Le temps est divisé en rondes de durée fixe et connue de tous les noeuds.
2. Un noeud peut envoyer *un* message à destination d'un ou plusieurs autres noeuds au début de chaque ronde.
3. Un noeud peut recevoir *un* message émis par un autre noeud à la fin de chaque ronde.

Dans ce modèle, nous considérons trois métriques de performance :

- **La latence**, qui correspond au nombre de rondes nécessaires pour effectuer une diffusion.
- **La complexité en messages**, qui correspond au nombre de messages qui sont nécessaires pour effectuer une diffusion.
- **Le débit**, qui correspond au nombre de messages que chaque noeud peut délivrer par ronde.

Il est important de noter que le débit est aussi important, si ce n'est plus, que la latence. En effet, il y a deux sources de latence qui doivent être considérées au sein d'un noeud : le temps nécessaire pour délivrer un message émis et le temps que ce message passe dans les files d'attente avant de pouvoir être émis. Si un noeud a peu de messages à diffuser, sa file d'attente est vide et la latence totale observée correspond à la latence du protocole de diffusion. Quand, au contraire, un noeud a un grand nombre de messages à diffuser simultanément, la latence totale observée devient dépendante du débit du protocole : plus le protocole permet de diffuser un grand nombre de messages par unité de temps, plus la file d'attente des messages à envoyer est petite.

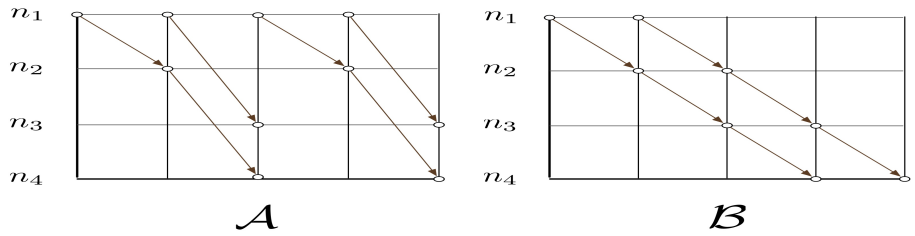


FIG. 1.7 – Deux exemples de protocole de diffusion.

Il est, par ailleurs, primordial de noter qu'un protocole optimisant la latence n'optimise pas nécessairement le débit. Considérons pour cela les deux protocoles de diffusion présentés sur la Figure 1.7.

- Dans le protocole $\mathcal{A}$ (diffusion en arbre), le noeud $n_1$ envoie tout d'abord le message à $n_2$. Dans la ronde suivante, le message est simultanément transmis de $n_2$ vers $n_4$ et de $n_1$ vers $n_3$.
- Dans le protocole $\mathcal{B}$ (diffusion en pipe-line), le noeud $n_1$ envoie tout d'abord le message à $n_2$. Dans la ronde suivante, $n_2$ transmet le message à $n_3$ qui le transmet, à son tour, à $n_4$ dans la ronde suivante.

En conséquence, le protocole $\mathcal{A}$ a une latence de 2 rondes, alors que le protocole $\mathcal{B}$ a une latence de 3 rondes. Le protocole $\mathcal{A}$ est donc meilleur pour la latence. Cependant, si l'on considère le débit, on constate que le protocole $\mathcal{A}$ permet de débuter une nouvelle diffusion de message toutes les 2 rondes, alors que le protocole $\mathcal{B}$ permet d'en débuter une toutes les rondes. En conséquence, le débit obtenu avec le protocole $\mathcal{B}$ est le double de celui obtenu avec le protocole $\mathcal{A}$.

Etant donné l'influence du débit sur la latence, nous nous consacrons en priorité, dans le travail présenté dans ce rapport, à concevoir des protocoles performants en débit.

### 1.3.2 Evaluation des métriques de performances

Notons tout d'abord que toutes les métriques de performances sont évaluées en considérant qu'aucune panne n'intervient. La latence et la complexité en messages d'une diffusion sont faciles à évaluer. En effet, il suffit d'étudier, dans le modèle proposé, la diffusion d'un seul message. La latence correspond au nombre de rondes nécessaires, alors que la complexité correspond au nombre de messages échangés. La latence optimale est d'une ronde, et la complexité en messages optimale est de $(N-1)$ messages, $N$ étant le nombre de noeuds dans le système. Cette dernière valeur s'explique par le fait qu'un message diffusé doit au moins atteindre les $(N-1)$ noeuds destinataires.

Il est, en revanche, beaucoup plus complexe d'évaluer le débit d'un protocole. En effet, l'évaluation du débit nécessite de considérer des diffusions simultanées de messages. Nous ne calculerons donc pas les débits de façon exacte, mais ferons des approximations réalistes. Par ailleurs, nous ne considérerons que les deux cas suivants :

- *Un seul noeud a une infinité de messages à émettre.* Dans ce cas, nous évaluons une borne supérieure sur le débit en étudiant le nombre minimal de rondes qui séparent deux émissions de message. Cette méthode appliquée aux protocoles $\mathcal{A}$ et $\mathcal{B}$ (Figure 1.7) donnent des débits respectifs de $\frac{1}{2}$ et 1. Notons que le débit optimal dans ce cas est de

1. En effet, le noeud émetteur ne peut pas émettre plus d'un message par ronde.

– *Tous les noeuds ont une infinité de messages à émettre.* Dans ce cas, nous évaluons une borne supérieure sur le débit de la façon suivante : $\frac{Complexite\ en\ messages_{opt}}{Complexite\ en\ messages} * D_{opt}$, où $D_{opt}$ représente le débit optimal que peut atteindre une diffusion dans laquelle tous les noeuds ont un message à emettre et $Complexite\ en\ messages_{opt}$ représente la complexité en messages optimale. Comme nous l'avons expliqué précédemment, cette complexité optimale est égale à $(N-1)$ messages. Par ailleurs, le débit optimal est $D_{opt} = \frac{N}{N-1}$. En effet, pour diffuser un message de façon optimale, $(N-1)$ transferts de messages sont nécessaires. Or $N$ transferts de messages peuvent être effectués durant une ronde. Une borne supérieure sur le débit qu'une diffusion peut atteindre est donc donnée par $\frac{N}{Complexite\ en\ messages}$.

## 1.4 Synthèse

Dans ce chapitre, nous avons défini ce que nous appelons un système distribué synchrone. Nous avons également présenté la définition formelle (à l'aide de quatre propriétés) d'une diffusion totalement ordonnée, ainsi que les grandes classes de protocoles implantant une telle diffusion. Nous avons, enfin, présenté les métriques de performances qui seront utilisées dans la suite de ce rapport.

# Chapitre 2

# Etat de l'art

Dans ce chapitre, nous dressons un état de l'art des protocoles qui ont été proposés pour effectuer des diffusions de messages totalement ordonnées dans le cadre des systèmes distribués synchrones. Parmi les cinq classes de protocoles présentées dans le chapitre précédent, seules deux classes ont été utilisées dans le cadre des systèmes distribués synchrones : l'utilisation de privilèges et l'utilisation d'historiques sur les communications. Nous présentons les différents protocoles appartenant à ces deux classes. Pour chaque protocole, nous démarrons par une présentation des hypothèses qui sont faites pour garantir son fonctionnement. Nous présentons ensuite le protocole et finissons par une étude théorique de ses performances. Nous concluons ce chapitre par une synthèse de l'état de l'art.

## 2.1 Protocoles basés sur un privilège

Dans cette section, nous présentons les protocoles existant qui reposent sur l'utilisation d'un privilège.

### 2.1.1 RTCAST

*RTCAST (Lightweight Multicast for Real-Time Process Groups)* [2] est le premier protocole basé sur la notion de privilège, que nous présentons. Il implante une diffusion totalement ordonnée conçue pour les applications nécessitant des garanties strictes sur les temps de diffusion (appelés systèmes temps-réels distribués à contraintes temporelles fortes).

***Hypothèses :***

- *Modèle de communication :* le protocole utilise un modèle de rondes synchrone.
- *Topologie du réseau :* le réseau d'interconnexion est supposé complètement connecté[1].
- *Identification des noeuds :* chaque noeud est identifié de manière unique, et il existe un ordre lexicographique sur les identifiants des noeuds.

***Présentation du protocole :***

`Emission des messages :` le privilège de la diffusion est modèlisé par un jeton unique (Figure 2.1). Le jeton circule sur un anneau virtuel comprenant l'ensemble des noeuds du système. Seul le possesseur du jeton peut diffuser des messages à l'aide d'une primitive

[1] Chaque noeud possède une connexion directe avec chaque autre noeud du système.

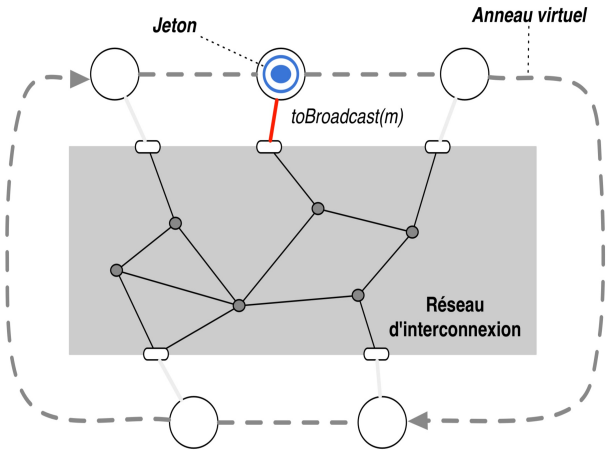


FIG. 2.1 – Illustration du principe de RTCAST.

`toBroadcast(m)`. Chaque noeud garde le jeton pendant une durée prédéterminée (*"token holding time"*), même s'il ne souhaite pas diffuser de message, puis le transmet à son successeur sur l'anneau.

`Ordonnancement total des messages :` chaque noeud délivre les messages dans l'ordre dans lequel il les reçoit. En raison de l'exclusion mutuelle de l'accés au réseau d'interconnexion modélisé par le jeton, il est garanti que l'ordre total est respecté.

***Analyse théorique des performances :***
`Latence :` dans RTCAST, lorsqu'un noeud diffuse un message, il est le seul à avoir accés au canal de communication (puisqu'il a le privilège). Il peut donc envoyer le message directement à chaque noeud du système. La latence de la diffusion est donc d'une ronde. Un exemple où chaque noeud garde le jeton pendant une durée équivalente à celle d'une ronde, et où le noeud d'identifiant $n_1$ ne souhaite pas diffuser de message est illustré sur la Figure 2.2.

`Complexité en messages :` la complexité en messages du protocole RTCAST est de $(N-1)$.

`Débit :` supposons qu'un seul noeud parmi $N$ possède un nombre infini de messages à diffuser. Pendant une période de $N$ rondes consécutives, il n'y aura qu'une seule diffusion. Le débit de RTCAST lorsqu'un seul noeud a une infinité de messages à diffuser est donc de $\frac{1}{N}$. Lorsque tous les noeuds ont une infinité de messages à diffuser, le débit est borné par $\frac{N}{N-1}$.

### 2.1.2 Protocole de Gopal et Toueg

Le protocole de diffusion totalement ordonnée de Gopal et Toueg [8] repose, lui aussi, sur l'utilisation d'un privilège. La caractéristique de ce protocole est de tolérer, outre les pannes franches de machines, les fautes par omission (décrites dans le chapitre précédent).

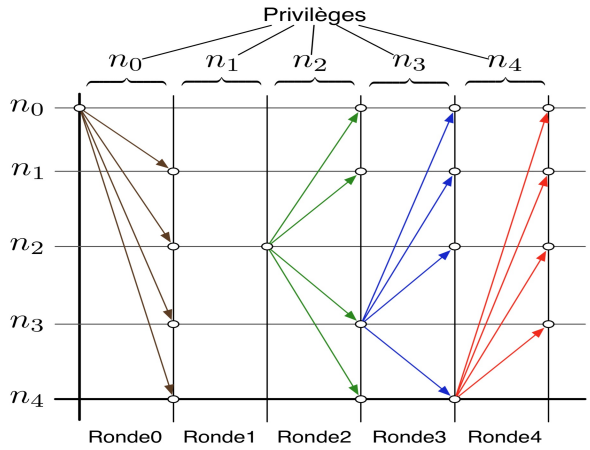


FIG. 2.2 – Analyse du débit dans RTCAST.

***Hypothèses :***

- *Modèle de communication :* le protocole utilise un modèle de rondes synchrone.
- *Topologie du réseau :* le réseau d'interconnexion est supposé complètement connecté.
- *Identification des noeuds :* il existe un noeud particulier, appelé "émetteur". Chaque noeud sait distinguer s'il est l'émetteur ou non. Le mécanisme permettant de changer l'émetteur (i.e. transmettre le privilège) n'est pas décrit dans [8].

***Présentation du protocole :***

`Emission des messages :` seul le noeud émetteur $E$ est autorisé à diffuser des séquences de messages. Chaque message dans une séquence est diffusé avec un numéro de séquence $numSeq(m)$.

`Réception des messages :` lorsqu'un noeud non-émetteur reçoit un couple $[m, numSeq(m)]$, il stocke ce couple dans un tampon local *received*, puis il acquitte la réception de cet ensemble en diffusant un message d'acquittement $[ACK, numSeq(m)]$ à tous les noeuds du système (Figure 2.3[2]).

`Ordonnancement total des messages :` lorsqu'un noeud a reçu une majorité des acquittements $[ACK, numSeq(m)]$, il transfert le couple $[m, numSeq(m)]$ stocké dans le tampon *received* vers un tampon appelé *valide*. A la fin de chaque ronde, chaque noeud délivre le message (s'il existe) stocké dans son tampon valide et ayant le numéro de séquence $numSeq(m) - 2$. Ce retard imposé dans la remise des messages permet de détecter les omissions dans la séquence de message à diffuser, et il assure l'ordre total de remise des messages en délivrant les messages dans l'ordre de leur numéro de séquence.

***Analyse théorique des performances :***

`Latence :` le protocole étudié nécessite que chaque noeud émette un acquittement de tous les messages qu'il reçoit. Par ailleurs, un message ne peut être mis dans le tampon *valide*

[2]Par soucis de clarté, nous avons représenté des rondes au cours desquelles des noeuds reçoivent plus d'un message.

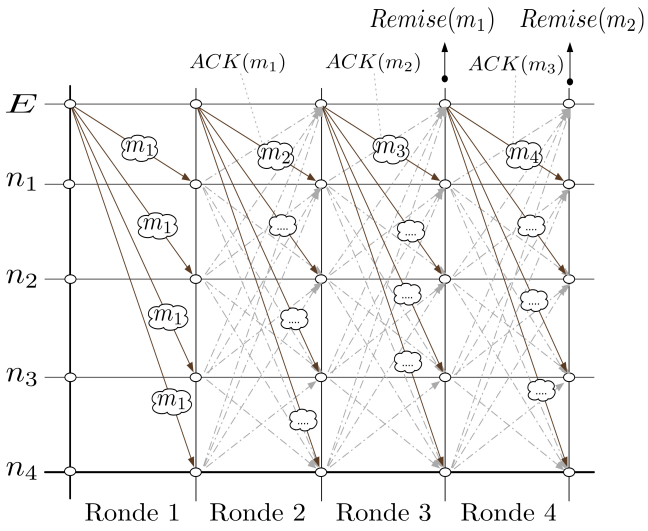


FIG. 2.3 – Exemple d'exécution du protocole de Gopal and Toueg.

que lorsqu'une majorité d'acquittements a été reçue. Dans le modèle de communication que nous utilisons pour évaluer les performances, $\lceil \frac{N-1}{2} \rceil$ rondes seront nécessaires avant qu'une majorité d'acquittements soient reçus (Figure 2.4). Si l'on considère que ce nombre de rondes est supérieur à 2 (i.e. $N \geq 4$), le retard imposé sera d'ores et déjà respecté et la latence sera de $1 + \lceil \frac{N-1}{2} \rceil$ rondes. Dans le cas contraire, la latence sera de $2 + \lceil \frac{N-1}{2} \rceil = 3$ rondes.

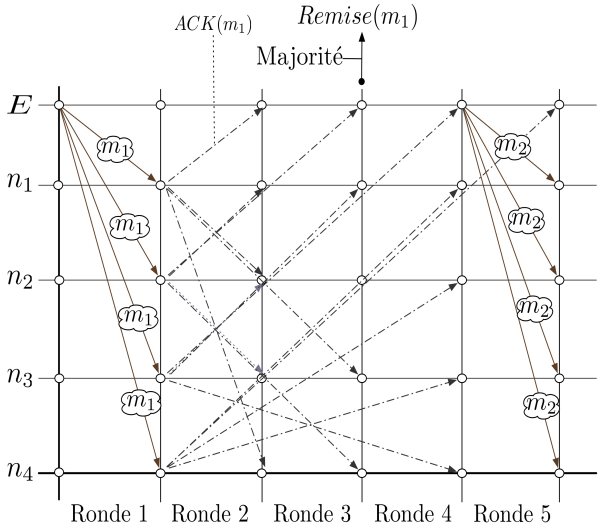


FIG. 2.4 – Analyse des performances du protocole de Gopal et Toueg dans le modèle de communication décrit en Section 1.3.

`Complexité en messages :` après l'envoi initial du message à l'ensemble ($(N-1)$ messages), chaque noeud envoie un acquittement aux autres noeuds (Figure 2.4). Par conséquent la com-

plexité en messages est égale à $(N-1)*(N-1)+(N-1) = N*(N-1)$ messages.

`Débit :` dans le cas où un seul noeud a un nombre infini de message à envoyer, le noeud émetteur ne peut initier une nouvelle diffusion que toutes les $(N-1)$ rondes. En effet, les $N-2$ rondes succédant à une diffusion sont utilisées par les autres noeuds pour émettre des acquittements (Figure 2.4). Par conséquent, le débit obtenu par le protocole lorsqu'un seul noeud a un nombre infini de messages à diffuser est de $\frac{1}{N-1}$. Dans le cas où $N$ noeuds on une infinité de messages à diffuser, le débit est borné par $\frac{N}{N*(N-1)} = \frac{1}{N-1}$.

### 2.1.3 MARS

Le protocole MARS [11] est le dernier exemple de protocole basé sur un privilège que nous étudions dans le cadre de cet état de l'art. Ce protocole permet de gérer un groupe de noeuds. Plus exactement, il permet de connaître l'ensemble des noeuds actifs, i.e. qui ne sont pas en panne, dans le système à un moment donné. Ce protocole n'assure pas de diffusion totalement ordonnée des messages échangés ; néanmoins, une telle diffusion peut être implantée de façon triviale en modifiant le protocole de gestion de groupe.

***Hypothèses :***

- *Modèle de communication :* le protocole utilise un modèle de rondes synchrone.
- *Topologie du réseau :* le réseau d'interconnexion est supposé complètement connecté.
- *Identification des noeuds :* chaque noeud est identifié de manière unique.

***Présentation du protocole :***

`Principe du TDMA :` ce protocole implante un protocole de gestion de groupe en utilisant le principe du TDMA (*Time Divisible Multiple Access*). Le principe du TDMA est le suivant : étant donné un système composé de $N$ noeuds, un cycle complet de TDMA est divisé en $N$ créneaux. Un créneau est dénoté $t^{i,k}$, où $i$ est un entier représentant le numéro du cycle et $k$ un entier représentant le numéro du créneau. Lorsque $k$ atteint la valeur $N$, un nouveau cycle commence. Un créneau par cycle est alors alloué à chaque noeud du système, durant lequel ce noeud a le privilège de diffuser un message : le créneau $t^{i,k}$ est associé au noeud $n_k$ où $0 \leq k \leq N$.

`Principe du protocole :` on apelle vue du système, l'ensemble des noeuds corrects appartenant à un groupe (dans cette étude, tous les noeuds du système définissent le groupe). Chaque vue est estampillée avec le numéro de cycle auquel elle correspond. Ainsi, on note $V_{i,k}$ la vue du système qu'a le noeud $n_k$ lors du cycle $i$. Afin d'avoir une vue globale et identique du système pour tous les noeuds, un consensus est réalisé. Pour ce faire, chaque noeud $n_k$ diffuse sa vue $V_{i,k}$ du système pendant le créneau $t^{i,k}$. En plus de participer au consensus en donnant sa propre vue du système, le noeud $n_k$ informe les autres noeuds qu'il n'est pas en panne en réalisant cette diffusion. A la fin d'un cycle, tous les noeuds ont reçu toutes les vues des autres noeuds et peuvent alors décider quelle est la vue globale du système. Le consensus est ainsi réalisé.

`Ordonnancement total des messages :` le protocole présenté précédemment peut être étendu pour effectuer des diffusions totalement ordonnées de messages. Pour ce faire, il suffit d'autoriser le noeud $n_k$ à associer un message $m$ à sa vue $V_{i,k}$, lorsqu'il la diffuse dans le cadre

du protocole de gestion de groupe. A l'issue du consensus (à la fin du cycle), si le noeud $n_k$ est considéré comme correct, alors son message $m$ pourra être délivré par tous les noeuds. C'est donc à la fin de chaque cycle, lorsque tous les noeuds se sont exprimés, que l'ont décide si un noeud est correct et si son message peut être délivré. En délivrant les messages par ordre des identifiants de noeud, on peut garantir que les séquences de messages délivrées respecteront l'ordre total. Le principe de ce protocole est représenté sur la Figure 2.5 : tous les messages émis lors du cycle 0 sont délivrés lors du créneau $t^{1,0}$. Sur cette figure, on considère qu'un créneau est équivalent à une ronde, dans notre modèle d'évaluation de performance décrit en section 1.3.

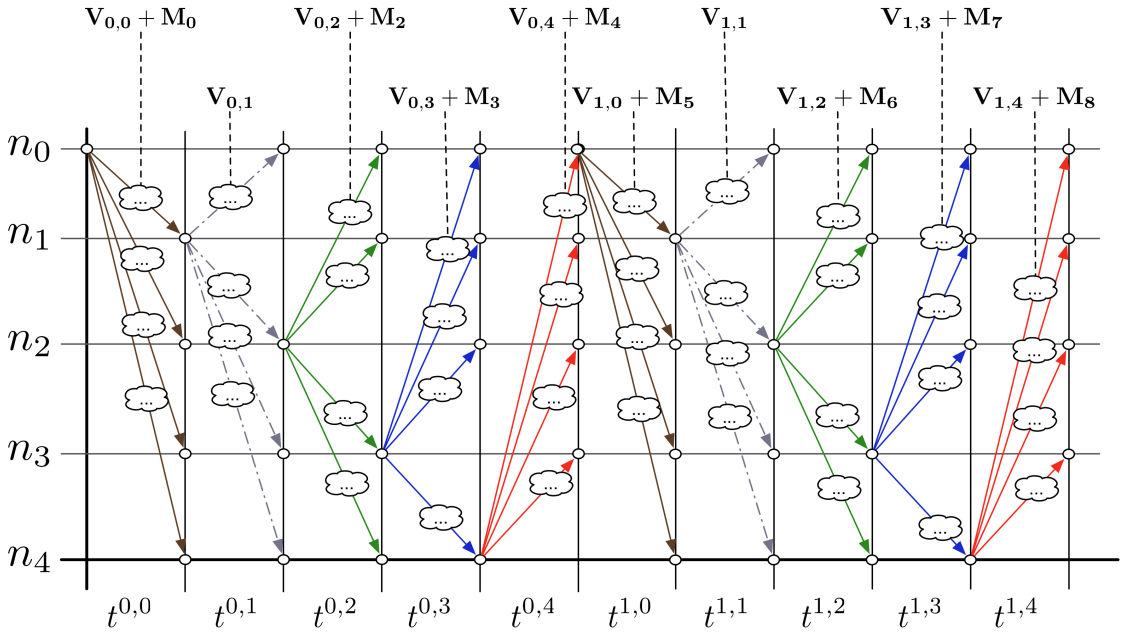


FIG. 2.5 – Exemple d'exécution du protocole MARS.

***Analyse théorique des performances :***

`Latence :` dans ce protocole, la latence d'une diffusion dépend de la place du noeud $n_i$ dans l'ordonnancement des privilèges réalisé par le TDMA. La latence maximale sera donc d'un cycle TDMA, soit $N$ rondes pour le premier noeud qui diffusera un message dans le cycle. La latence minimum sera de 1 ronde, pour le dernier noeud qui diffusera un message pendant un cycle TDMA.

`Complexité en messages :` la complexité en messages du protocole MARS est de $(N-1)$.

`Débit :` supposons qu'un seul noeud parmi $N$ possèdent un nombre infini de messages à diffuser. Pendant une période de $N$ rondes consécutives, il n'y aura qu'une seule diffusion.

Le débit de MARS lorsqu'un seul noeud a une infinité de messages à diffuser est donc de $\frac{1}{N}$. Lorsque tous les noeuds ont une infinité de messages à diffuser, le débit est borné par $\frac{N}{N-1}$.

## 2.2 Protocoles basés sur l'historique des communications

Dans cette section, nous présentons les protocoles existant qui reposent sur l'utilisation de l'historique des communications pour ordonner les messages reçus.

### 2.2.1 HAS

HAS [6] est une collection de protocoles de diffusion totalement ordonnée. Les auteurs définissent un modèle de protocole appliqué à trois types de fautes : les fautes par omission qui donne la variante HAS-$\mathcal{O}$, les fautes temporelles qui donne HAS-$\mathcal{T}$ et les fautes byzantines qui donne HAS-$\mathcal{B}$. Cependant, le principe de diffusion de HAS reste le même pour ces trois protocoles, les différences intervenant uniquement dans le cas où des pannes se produisent. Comme nous l'avons expliqué plus tôt, nous ne nous intéressons qu'au cas d'exécution sans panne dans notre étude ; nous ne distinguerons donc pas les différentes variantes de HAS dans ce travail.

***Hypothèses :***

- `Modèle de communication :` le modèle de communication est synchrone. Les déviations entre les horloges synchronisées sont bornées par une constante $\epsilon$. On suppose disposer d'une borne supérieure $\delta$ sur le temps de transfert d'un message entre deux noeuds directement connectés.
- `Topologie du réseau :` le réseau d'interconnexion est connecté, c'est à dire que tout noeud peut atteindre tout autre noeud (éventuellement via plusieurs sauts).
- `Identification des noeuds :` chaque noeud est identifié de manière unique et il existe un ordre total sur les identifiants des noeuds.

***Présentation du protocole :***

`Emission des messages :` un noeud d'identifiant $n_i$ initie la diffusion d'un message $m$ en l'envoyant à ses voisins[3] avec la primitive `toBroadcast(`$m$`)`. Il associe à $m$ son identifiant $n_i$ ainsi qu'une estampille horaire locale $T_m$ correspondant au moment où la diffusion de $m$ a été initiée. Le message $m$ est donc identifié de manière unique par l'association $[m, P_i, T_m]$ (voir Figure 2.6 pour le temps $T_0$).

`Réception des messages :` pour la propagation d'un message, le principe d'innondation est utilisé : quand un noeud $n_j$ reçoit un message $m$ pour la première fois, il le stocke puis le transmet une fois à tous ses voisins à l'exception du voisin par lequel il a reçu $m$ (voir Figure 2.6 pour le temps $T_0 + \lambda$). Ensuite, si $n_j$ reçoit de nouveau le message $m$ par un autre voisin, il ne le retransmet pas. En utilisant ce principe de diffusion, il est prouvé dans [6] que tous les noeuds reçoivent $m$ au moins une fois et que la diffusion de $m$ se termine dans un temps fini.

[3] Deux noeuds sont voisins quand ils partagent un lien du réseau d'interconnexion.

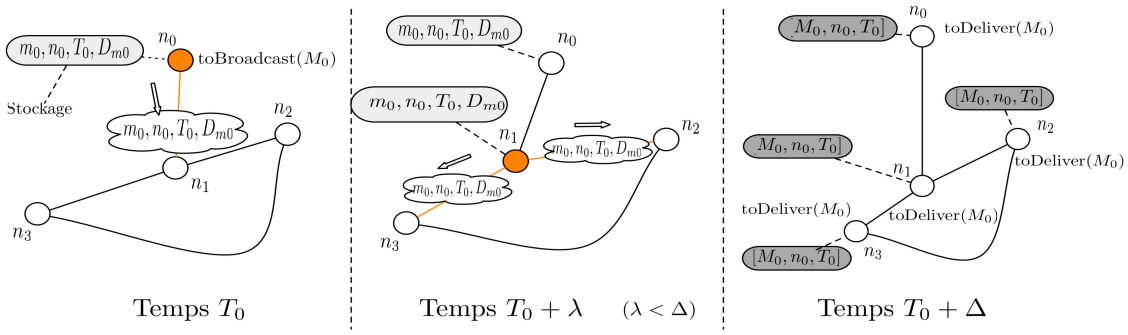


FIG. 2.6 – Principes du protocole HAS.

`Ordonnancement total des messages :` chaque message stocké est associé à une date $D_m$ de livraison. Cette date est calculée de la façon suivante : chaque noeud connaît une constante $\Delta$ qui correspond à la borne supérieure sur le temps qu'une diffusion peut prendre dans le système. La date $D_m$ de remise d'un message $m$ est telle que : $D_m = T_m + \Delta + \epsilon$, où $T_m$ est l'estampille horaire du message $m$ et $\epsilon$ est la déviation maximale entre les horloges de deux noeuds du système. Notons enfin que la valeur de la constante $\Delta$ dépend de la constante $\delta$ et de la topologie du réseau.

Sur chaque noeud, une tâche est chargée de parcourir en permanence l'ensemble des messages stockés, afin de vérifier si certains doivent être délivrés. Si la date de livraison $D_m$ d'un message est inférieure à la valeur de l'horloge locale, alors il faut délivrer le message $m$ à l'aide de la primitive `toDeliver(`$m$`)` (voir Figure 2.6 pour le temps $T_0 + \Delta$). Si deux messages possèdent la même estampille, ils sont délivrés par ordre croissant des identifiants des émetteurs, assurant ainsi l'ordre total sur la remise des messages diffusés dans le système.

***Analyse théorique des performances :***

`Latence :` la latence du protocole HAS dépend de la topologie du réseau. Supposons que $\epsilon = 0$ et que $\delta = 1$ ronde. Dans le meilleur cas (réseau totalement connecté), la latence est d'une ronde. Dans le pire des cas (noeuds connectés en pipe-line), la latence est de N rondes.

`Complexité en messages :` la complexité en messages du protocole HAS dépend également de la topologie du réseau. Nous supposons un réseau totalement connecté. Dans un tel réseau, après l'envoi initial du message à l'ensemble de ses voisins ($(N-1)$ envois), un message est retransmis $(N-2)$ fois par chaque noeud (excepté l'émetteur). En effet, aprés l'émission du message $m$, les noeuds non-initiateurs retransmettent $m$ à tous leurs voisins (à l'exception de l'émetteur). Par ailleurs, cette retransmission est initiée au même moment sur chaque noeud (au début de la deuxième ronde). Le mécanisme de limitation de l'inondation consistant à ne pas retransmettre un message déja reçu sur un même lien ne peut donc pas fonctionner (Figure 2.7). Par conséquent la complexité en messages est égale à $(N-1) * (N-2) + (N-1) = (N-1)^2$ messages.

`Débit :` le débit du protocole HAS dépend de la topologie du réseau. Considérons le cas d'un réseau totalement connecté. Dans le cas où un seul noeud a un nombre infini de messages à diffuser, le débit est de $\frac{1}{N-1}$. En effet, comme il est représenté sur la Figure 2.7, l'émetteur

ne peut émettre un nouveau message que toutes les $N-1$ rondes. Dans le cas où $N$ noeuds ont un nombre infini de messages à diffuser, le débit est borné par $\frac{N}{(N-1)^2}$.

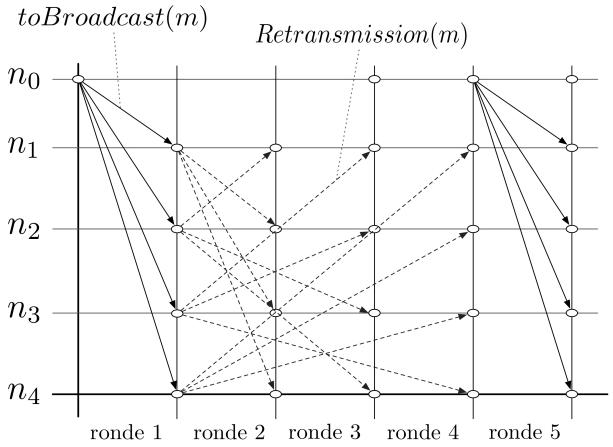


FIG. 2.7 – Analyse des performances du protocole HAS dans le modèle de communication décrit en Section 1.3.

### 2.2.2 ABP

Le protocole ABP (*Atomic Broadcast Protocol*) [3] est le second protocole basé sur la notion d'historique de communications, que nous présentons. Il utilise un mécanisme de vues synchrones pour établir la diffusion totalement ordonnée de messages et pour traiter les pannes potentielles.

***Hypothèses :***

- `Modèle de communication :` le modèle de communication est synchrone. On suppose disposer d'une borne supérieure $\delta$ sur le temps de transfert d'un message entre deux noeuds directement connectés. Pour des raisons de simplification, nous supposerons que $\delta = 1$ ronde.
- `Topologie du réseau :` le réseau d'interconnexion est supposé complètement connecté.
- `Identification des noeuds :` chaque noeud est identifié de manière unique et il existe un ordre total sur les identifiants des noeuds.

***Présentation du protocole :***

`Vue synchrone :` le mécanisme de vue synchrone est utilisé dans les systèmes distribués dynamiques (avec départ et arrivée de noeuds) pour identifier de manière unique les différents états du système au cours de son évolution. L'état du système est représenté par l'ensemble des noeuds considérés comme valides. Une vue synchrone est donc composée d'un ensemble $\mathcal{N}$ de noeuds et est identifiée par une estampille unique $\mathcal{E}$. Lorsque l'ensemble $\mathcal{N}$ est modifié (ajout d'un nouveau noeud, panne ou départ d'un noeud), une nouvelle vue est créée. Les estampilles utilisées dans les vues synchrones sont basées sur les horloges logiques de Lamport [12] : soit $\mathcal{V}_k$ la vue d'estampille $\mathcal{E}_k$ représentant l'état $\mathcal{N}_k$ du système à un instant donné. Lorsqu'un changement d'état du système survient, une nouvelle vue $\mathcal{V}_{k+1}$ d'estampille $\mathcal{E}_{k+1}$

est créée puis diffusée à l'ensemble des noeuds du système. Les estampilles utilisées étant des horloges logiques de Lamport, chaque fois qu'une nouvelle vue est créée la valeur de l'estampille croît strictement ($\mathcal{E}_{k-1} > \mathcal{E}_k > \mathcal{E}_{k+1}$).

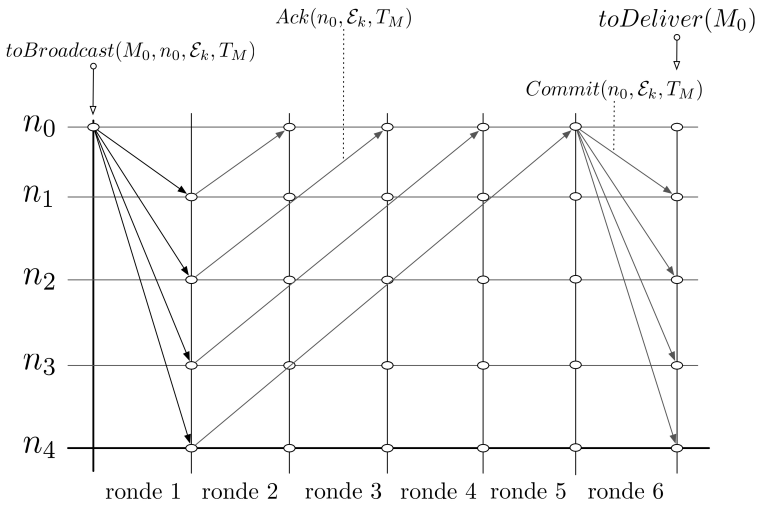


FIG. 2.8 – Exemple d'exécution du protocole ABP.

`Emission (phase 1) :` lorsqu'un noeud veut diffuser un message $m$, il diffuse l'association $[m, n_i, \mathcal{E}_k, T_m]$ à tous les noeuds. L'identifiant $n_i$ est celui du noeud émetteur, $T_m$ est l'estampille associée à $m$ et $\mathcal{E}_k$ est l'estampille de la vue courante (Figure 2.8). Le couple $[n_i, t_m]$ constitue donc un identifiant unique du message $m$.

`Acquittement (phase 1) :` lorsqu'un noeud $n_j$ reçoit un ensemble $[m, n_i, \mathcal{E}_k, T_m]$, il s'assure que la vue $V_k$ est cohérente et que le noeud $n_i$ est considéré comme valide dans cette vue. Ensuite, s'il considère l'émetteur comme valide, il stocke le message $m$ dans une variable locale *Received*. Il acquitte alors le message $[m, n_i, \mathcal{E}_k, T_m]$ en envoyant un ensemble $[ACK, n_j, [n_i, \mathcal{E}_k, T_m]]$ au noeud $n_i$ (Figure 2.8). Si le noeud $n_i$ reçoit des acquittements positifs de la part de tous les noeuds, il déclenche la phase de confirmation pour la diffusion du message $m$.

`Confirmation (phase 2) :` le noeud $n_i$ confirme la validité de la diffusion du message $m$ en diffusant à tous les noeuds l'ensemble $[COMMIT, [n_i, \mathcal{E}_k, T_m]]$. Il stocke ensuite l'ensemble $[m, n_i, \mathcal{E}_k, T_m]$ dans un tampon local *toDeliver*. Lorsqu'un noeud $n_j$ reçoit un ensemble $[COMMIT, [n_i, \mathcal{E}_k, T_m]]$, il sort le message $m$ correspondant à l'identifiant $[n_i, t_m]$ de sa variable locale *Received* puis stocke de nouveau l'ensemble $[m, n_i, \mathcal{E}_k, T_m]$ dans le tampon *toDeliver*.

`Ordonnancement total des messages :` sur chaque noeud, un démon est chargé de parcourir en permanence l'ensemble des messages stockés dans le tampon *toDeliver*, afin de délivrer les messages qui s'y trouvent à l'aide de la primitive `toDeliver(`$m$`)`. Les messages sont délivrés en suivant l'ordre des estampilles $T_m$. Si deux messages possèdent la même

estampille, ils sont alors délivrés par ordre croissant des identifiants des émetteurs. La procédure de remise de messages assure ainsi l'ordre total sur les séquences de messages qu'elle délivre.

***Analyse théorique des performances :***
`Latence :` le protocole étudié nécessite que chaque noeud émette un acquittement de tous les messages qu'il reçoit. Par ailleurs, un message ne peut être mis dans la variable *toDeliver* que lorsque tous les acquittements ont été reçus. Dans le modèle de communication que nous utilisons pour évaluer les performances, $(N-1)$ rondes sont nécessaires avant que tous les acquittements soient reçus. Ensuite, la diffusion de la confirmation nécessite une ronde pour s'éffectuer. Supposons que $\delta = 1$ ronde, alors la latence d'une diffusion dans le protocole ABP est donc de $(1+N-1+1)$ soit $(N+1)$ rondes.

`Complexité en messages :` la complexité en messages de ce protocole est de $(N-1)$ messages pour la diffusion initiale, auxquels s'ajoutent $(N-1)$ messages d'acquittements et $(N-1)$ messages de confirmation. Soit une complexité en messages de $3*(N-1)$ messages.

`Débit :` le débit du protocole ABP, dans le cas où un seul noeud a un nombre infini de messages à diffuser est de $\frac{1}{N+1}$. En effet, en raison de la latence retardée par la présence d'acquittements et de confirmation, un noeud ne peut diffuser un message, au mieux, qu'une fois toutes les $(N+1)$ rondes. Dans le cas où $N$ noeuds ont un nombre infini de messages à diffuser, le débit est borné par $\frac{N}{3*(N-1)}$.

### 2.2.3 Atom

Le protocole Atom [4] est également basé sur l'utilisation d'un historique des communications. Sa particularité est de ne pas dégrader la latence des diffusions en fonction des départs et des arrivées de noeuds dans le système. Ceci est réalisé à l'aide d'un consensus appelé $CUP$ sur un ensemble à priori uncertain de noeuds participants.

***Hypothèses :***
- `Modèle de communication :` le protocole utilise un modèle de rondes synchrone. Il existe une borne supérieure $\delta$ sur le temps de transfert d'un message entre deux noeuds directement connectés. Nous admettons pour cette étude que $\delta = 1$ ronde.
- `Topologie du réseau :` le réseau d'interconnexion est supposé complètement connecté.
- `Identification des noeuds :` chaque noeud est identifié de manière unique et il existe un ordre total sur les identifiants des noeuds.

***Présentation du protocole :***
`Principe des créneaux :` dans ce protocole, des créneaux temporels sont définis avec une durée $\Delta$ fixe connue de tous les noeuds, ils sont désignés à l'aide de la notation $C_k$, où $C_0$ est le créneau initial. Un timer est armé sur chaque noeud lorsqu'un créneau commence. Au bout de $\Delta$ unité de temps, une primitive $end\_slot(C_k)$ notifie le noeud local de la fin du créneau courant $(C_k)$, et le créneau suivant $(C_{k+1})$ commence. Notons bien, qu'un créneau peut être équivalent à plusieurs rondes.

`Emission des messages :` lorsque l'application utilisant ce protocole décide de diffuser un message à l'aide de la primitive $toBroadcast(m)$, le noeud $n_i$ stocke le message $m$ dans un tampon $Tamp_k$. Si l'application décide de diffuser un autre message $m'$ pendant le même créneau $C_k$, $m'$ sera ajouté au tampon $Tamp_k$. Il en est de même pour tous les messages qui sont diffusés sur le noeud $n_i$ pendant ce créneau. Lorsque la fin du créneau $C_k$ approche[4], $n_i$ diffuse à tous les noeuds l'ensemble $[Tamp_k, n_i, C_k]$. Cette diffusion déclenche alors une instance $CUP(C_k, n_i)$ d'un consensus particulier ($CUP$ [4]) fonctionnant sans connaître à priori l'ensemble des noeuds appartenant au système. Ce consensus est utilisé pour décider de la validité de l'ensemble $[Tamp_k, n_i, C_k]$, dans un contexte où des noeuds peuvent joindre ou quitter le système à tous moment et où des pannes peuvent survenir. Comme tous les consensus, il impose que tous les noeuds du système diffusent leur avis sur l'objet du consensus.

`Réception des messages :` lorsqu'un noeud reçoit un ensemble $[Tamp_k, n_i, C_k]$, il le stocke dans un tampon local *received*, puis diffuse son avis sur la validité de cet ensemble dans le cadre du consensus $CUP(C_k, n_i)$.

`Ordonnancement total des messages :` ce protocole fait l'hypothèse qu'une fois l'événement $end\_slot(C_k)$ déclenché, tous les consensus $CUP(C_k, X)$ initiés pendant le créneau $C_k$ sont terminés. Chaque noeuds délivre alors les messages contenus dans les différents tampons en suivant l'odre des identifiants des émetteurs. Les messages sont extraits depuis les tampons de façon déterministe (par exemple dans l'ordre FIFO[5]), de façon à garantir l'ordre total sur la remise des messages.

***Analyse théorique des performances :***

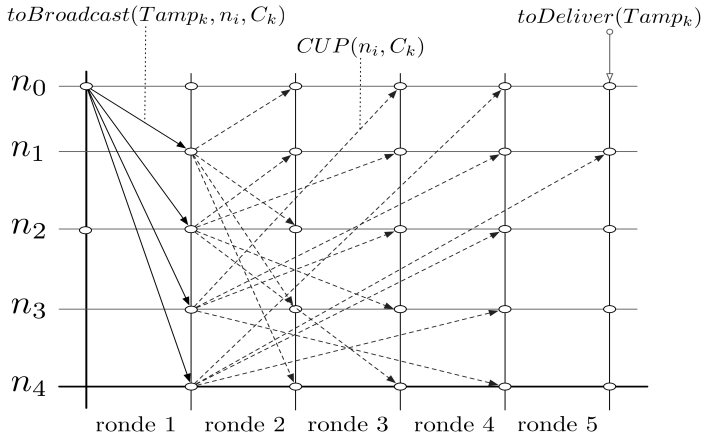


FIG. 2.9 – Analyse des performances du protocole Atom dans le modèle de rondes défini en section 1.3.

`Latence :` lorsqu'un noeud diffuse un message avec Atom, il déclenche automatiquement une instance du consensus $CUP$. $(N-1)$ messages relatifs au consensus sont alors diffusés sur le

[4]Un noeud dispose d'un autre timer qui se déclenche pour lui signifier que les messages doivent être émis.
[5]First In First Out.

réseau de communication pour chaque message qu'un noeud souhaite diffuser (Figure 2.9). Aucun noeud n'est autorisé à délivrer le message avant la fin du consensus. La latence d'une diffusion dans Atom est donc de $(1 + N - 1)$, soit $N$ rondes.

`Complexité en messages :` la complexité en messages du protocole Atom est de $(N-1) + (N-1) * (N-1) = N * (N-1)$ messages.

`Débit :` le débit du protocole Atom, dans le cas où un seul noeud a un nombre infini de messages à diffuser est de $\frac{1}{N-1}$ en raison de la présence des messages du consensus $CUP$. Dans le cas où $N$ noeuds ont un nombre infini de messages à diffuser, le débit est borné par $\frac{N}{N*(N-1)} = \frac{1}{N-1}$.

### 2.2.4 Quick Atomic Broadcast

Dans ce travail, P.Berman et A.Bharali [5] se sont focalisés sur la réalisation d'une diffusion atomique rapide tolérant jusqu'à $f$ fautes, $f$ étant un nombre entier prédéterminé. Une diffusion atomique est considérée comme rapide (*"quick"*) lorsque sa latence dépend du nombre de fautes effectivement rencontrées pendant son exécution, plutôt que du nombre maximum de fautes qu'elle peut tolérer. Le protocole proposé utilise un consensus tolérant les fautes byzantines afin de réaliser la diffusion totalement ordonnée. Notons enfin que l'intérêt principal de ce protocole est de traiter les défaillances de façon rapide et efficace. Cependant, nous ne nous intéresserons pas aux détails du consensus qui traite ces défaillances, afin de nous focaliser sur le fonctionnement sans pannes du protocole.

***Hypothèses :***

- `Modèle de communication :` le protocole utilise un modèle de rondes synchrone.
- `Topologie du réseau :` le réseau d'interconnexion est supposé complètement connecté.
- `Identification des noeuds :` chaque noeud est identifié de manière unique et il existe un ordre total sur les identifiants des noeuds.

***Présentation du protocole :***

`Emission des messages :` un noeud $n_i$ associe à chaque message $m$ qu'il diffuse les informations suivantes : son identifiant $n_i$ et la date $T_m$ d'initiation de la diffusion. Le couple $[n_i, T_m]$ identifie de manière unique le message $m$. Ensuite, $n_i$ diffuse l'ensemble $[m, n_i, T_m]$ à tous les noeuds, initiant par cette action un consensus $C_{byz}[n_i, T_m]$. Ce consensus est chargé d'établir l'accord sur la validité du message $m$. Notons qu'il tolère les fautes byzantines.

`Réception des messages :` le consensus $C_{byz}[n_i, T_m]$ impose que chaque noeud correct diffuse son accord sur la validité de la diffusion de l'ensemble $[m, n_i, T_m]$ lorsqu'il le reçoit. Le détail du protocole réalisant le consensus $C_{byz}[n_i, T_m]$ est disponible dans l'article complet [5], où il est prouvé que l'accord est atteint avec une latence de $(\varphi + N - 1)$ rondes, où $\varphi$ est le nombre de pannes rencontrées pendant la réalisation du consensus. Une fois que le consensus $C_{byz}[n_i, T_m]$ s'est achevé, son résultat (validé ou non) est stocké dans un historique $H$. Si l'accord est négatif, alors tous les noeuds suppriment le message $m$ et ignorent sa diffusion. Si l'accord est positif, l'ensemble $[m, n_i, T_m]$ est stocké localement par tous les noeuds dans

une variable tampon *received*.

`Ordonnancement total des messages :` une fois par ronde, chaque noeud parcourt sa variable *received* et délivre les messages dont le résultat du consensus est contenu dans l'historique $H$. Chaque message $m$ ainsi délivré est supprimé de la variable *received*. Les messages sont délivrés en suivant l'ordre des estampilles $T_m$ contenues dans le résultat des consensus qui correspondent à ces messages. Si deux messages possèdent la même estampille, alors la délivrance suit l'ordre des identifiants des noeuds émetteurs. L'ordre total est alors assuré pour la délivrance des messages.

***Analyse théorique des performances :***

Une analyse dans notre modèle de ronde d'un exemple d'exécution du protocole *Quick atomic broadcast*, dans un système composé de 5 noeuds est illustré sur la Figure 2.10.

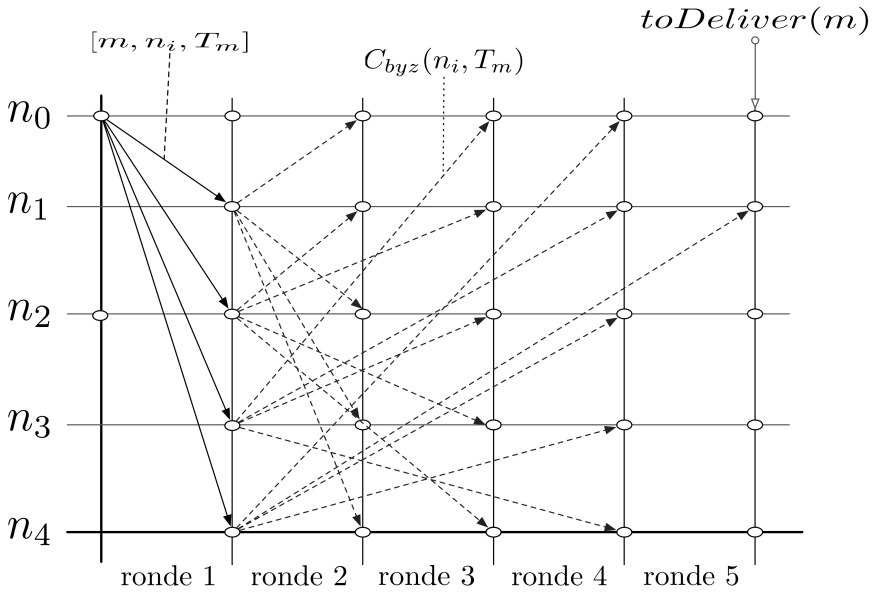


FIG. 2.10 – Analyse des performances du protocole "Quick atomic broadcast" dans le modèle de ronde défini en section 1.3.

`Latence :` comme il est prouvé dans ce protocole, un consensus à besoin de $(N-1)$ rondes lorsqu'aucune panne ne se produit. La latence d'une diffusion est donc de $(1+N-1) = N$ rondes.

`Complexité en messages :` la complexité en message est de $((N-1)*(N-1)+N-1) = N*(N-1)$ messages dans ce protocole en raison des messages utilisés lors du consensus $C_{byz}$ initié lors de chaque diffusion.

`Débit :` dans le cas où un seul noeud désire diffuser des messages, le débit théorique de diffusion dans ce protocole est de $\frac{1}{N}$, à cause des messages liés au consensus $C_{byz}$, qui empêchent un noeud de diffuser des messages avec une fréquence supérieure à une fois toutes les $N$ rondes. Dans le cas où $N$ noeuds ont un nombre infini de messages à diffuser, le débit est borné par $\frac{N}{N*(N-1)} = \frac{1}{(N-1)}$.

## 2.3 Synthèse

Cette section propose une synthèse de l'état de l'art. Pour chaque protocole, nous rappelons sa latence, sa complexité en messages, ainsi que son débit théorique lorsqu'un seul noeud possède un nombre infini de messages à diffuser ($D_1$) et lorsque tous les noeuds ont une infinité de messages à diffuser ($D_N$).

**Protocoles basés sur un historique des communications**

| ***Nom*** | ***Latence*** | ***Complexité en messages*** | $D_1$ | $D_N$ |
|---|---|---|---|---|
| HAS | $N$ | $(N-1)^2$ | $\frac{1}{N-1}$ | $\frac{N}{(N-1)^2}$ |
| ABP | $N+1$ | $3(N-1)$ | $\frac{1}{N+1}$ | $\frac{N}{3(N-1)}$ |
| Atom | $N$ | $N(N-1)$ | $\frac{1}{N-1}$ | $\frac{1}{N-1}$ |
| Quick atomic broadcast | $N$ | $N(N-1)$ | $\frac{1}{N}$ | $\frac{1}{N-1}$ |

Ce récapitulatif montre que les protocoles utilisant un historique des communications fournissent une diffusion ayant une latence linéaire en fonction du nombre de noeuds et des débits inversement proportionnels au nombre de noeuds. Les performances de ces protocoles sont mauvaises, ce qui s'explique souvent par le fait qu'ils utilisent des acquittement et/ou procèdent à des consensus, ce qui augmente la complexité en messages et de ce fait réduit les performances des diffusions.

**Protocoles basés sur l'utilisation d'un privilège**

| ***Nom*** | ***Latence*** | ***Complexité en messages*** | $D_1$ | $D_N$ |
|---|---|---|---|---|
| RTCAST | 1 | $N-1$ | $\frac{1}{N}$ | $\frac{N}{N-1}$ |
| Gopal et Toueg | $1+\lceil\frac{N-1}{2}\rceil$ | $N(N-1)$ | $\frac{1}{N-1}$ | $\frac{1}{N-1}$ |
| MARS | $1 \leq l \leq N$ | $N-1$ | $\frac{1}{N}$ | $\frac{N}{N-1}$ |

Le récapitulatif des performances des protocoles basés sur un privilège montre que les diffusions de ces protocoles garantissent un très bon débit lorsque tous les noeuds ont une infinité de messages à diffuser. Par ailleurs, certains de ces protocoles obtiennent une latence intéressante d'une ronde, ce qui est optimal. Cependant tous ces protocoles ont un débit de diffusion inversement proportionnel au nombre de noeuds lorsqu'un seul noeud diffuse une infinité de messages. Ce mauvais débit s'explique par le fait que le privilège de la diffusion est attribué aussi bien aux noeuds désirant diffuser des messages qu'aux noeuds ne souhaitant pas diffuser de messages.

Cette synthèse nous montre que les protocoles basés sur l'utilisation d'un privilège fournissent des diffusions plus performantes que celles proposées par les protocoles utilisant un historique des communications. Néanmoins, ces performances ne sont pas satisfaisantes (en termes de débit) du fait qu'elles ne sont bonnes que lorsque tous les noeuds ont des messages

à diffuser. Notre objectif est de proposer un protocole garantissant une diffusion performante dans tous les cas d'exécution. Notre première priorité est que le protocole garantisse un débit aussi élevé que possible (car le débit conditionne la latence quand le système est chargé). Nous souhaitons néanmoins également que le protocole garantisse une latence faible.

# Chapitre 3

# Le protocole SCR

Dans ce chapitre, nous présentons SCR, un protocole que nous avons élaboré avec pour objectif d'améliorer les performances des protocoles existants. Plus exactement, notre but est de garantir un bon débit quel que soit le nombre de noeuds ayant une infinité de messages à diffuser. Ce protocole fonctionne en organisant les noeuds du système sous forme d'un anneau virtuel. L'utilisation de la topologie en anneau pour la diffusion totalement ordonnée a initialement été introduite dans FSR [9], un protocole conçu pour les systèmes asynchrones. Tout comme dans FSR, chaque noeud communique uniquement avec son successeur sur l'anneau, ce qui permet d'obtenir un débit très élevé. En revanche, les hypothèses de synchronie faites sur le système permettent de modifier la façon dont l'ordonnancement est réalisé, ce qui se traduit par un gain en latence variant de 1 à N rondes, selon la position du noeud émetteur sur l'anneau.

Ce chapitre est organisé de la façon suivante : nous introduisons tout d'abord le protocole FSR. Nous décrivons ensuite le protocole SCR. Enfin, nous concluons par une analyse des performances de SCR.

## 3.1 Le protocole FSR

### 3.1.1 Présentation du protocole

Le protocole FSR (*Fix Sequencer Ring*) [9] garantit une diffusion totalement ordonnée basée sur l'utilisation d'un séquenceur fixe. Ce protocole étant conçu pour fonctionner sur des systèmes distribués asynchrones, il ne suppose aucune borne temporelle sur les temps de transmission et de traitement des messages. FSR utilise un séquenceur pour attribuer des numéros de séquence aux messages et un anneau virtuel pour les diffuser. Contrairement aux protocoles traditionnels utilisant un séquenceur fixe, les noeuds dans FSR envoient leurs messages uniquement à leur successeur sur l'anneau. Ces messages sont ensuite acheminés jusqu'au séquenceur.

Le fonctionnement du protocole FSR est illustré sur la Figure 3.1. Le noeud $n_i$ initie la diffusion d'un message $m$ en le transmettant à son successeur $n_{i+1}$ (message $m_1$). Ce dernier le stocke et le transmet à son successeur, et ainsi de suite jusqu'à ce que ce message atteigne le séquenceur ($n_0$). Comme dans tous les protocoles utilisant un séquenceur fixe, le séquenceur assigne des numéros de séquence croissants aux différents messages, assurant ainsi un ordre total sur leur livraison. Le couple constitué du message $m$ et de son numéro de séquence $seq(m)$ ($m_2$ sur la Figure 3.1) est ensuite transmis aux successeurs de $n_0$ jusqu'au noeud $n_{i-1}$.

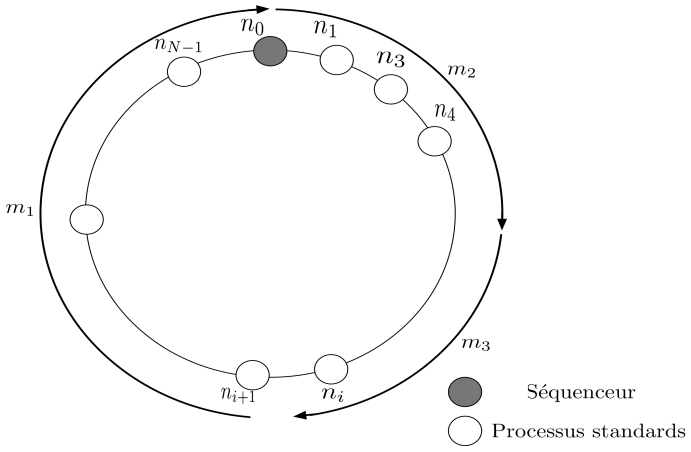


FIG. 3.1 – Illustration du principe de FSR

Chaque successeurs de $n_0$ délivre le message après l'avoir transmis à son successeur. Enfin, un dernier message $m_3$ comprenant uniquement le numéro de séquence de $m$ est transmis de $n_{i-1}$ à $n_{N-1}$, afin que tous les noeuds ayant déjà reçu le message, mais ne l'ayant pas délivré puisse le délivrer. Notons que ce dernier message ne contient qu'un numéro de séquence et peut donc être accolé à un autre message à diffuser. En conséquence, la partie "utile" d'un message ne fait le tour de l'anneau qu'une seule fois.

### 3.1.2 Performances du protocole

`Latence :` la latence d'une diffusion varie selon la position des noeuds. La latence $L_i$ d'une diffusion initiée par un noeud $n_i$ est $L_i = 2N - i - 1$.

`Complexité en messages :` la complexité du protocole FSR est de $(N-1)$ messages car les messages ne véhiculant qu'un numéro de séquence peuvent être accolés à des messages "utiles".

`Débit :` dans le cas où un seul noeud a un nombre infini de messages à diffuser, le protocole FSR garantit un débit de 1. Ce résultat, illustré sur la Figure 3.2 vient du fait que chaque noeud peut délivrer un message à chaque ronde. Dans le cas où $N$ noeuds ont une infinité de messages à diffuser, le débit est borné par $\frac{N-1}{N-1} * D_{opt} = D_{opt}$.

## 3.2 Le protocole SCR

Ce protocole a pour principe d'utiliser les hypothèses de synchronie, et en particulier la synchronisation parfaite des horloges fournie par les modèles de systèmes distribués synchrones. Tout comme FSR, il repose sur une diffusion des messages à l'aide d'un anneau virtuel. En revanche, l'ordonnancement diffère : chaque noeud utilise une horloge physique pour estampiller les messages (d'où le nom "Synchronized Clock Ring"). L'hypothèse de synchronie du système permet de connaître le temps auquel le message aura fini le tour d'anneau. Grâce à ces horloges synchronisées, ce protocole implante une diffusion totalement ordonnée ayant une meilleur latence que FSR.

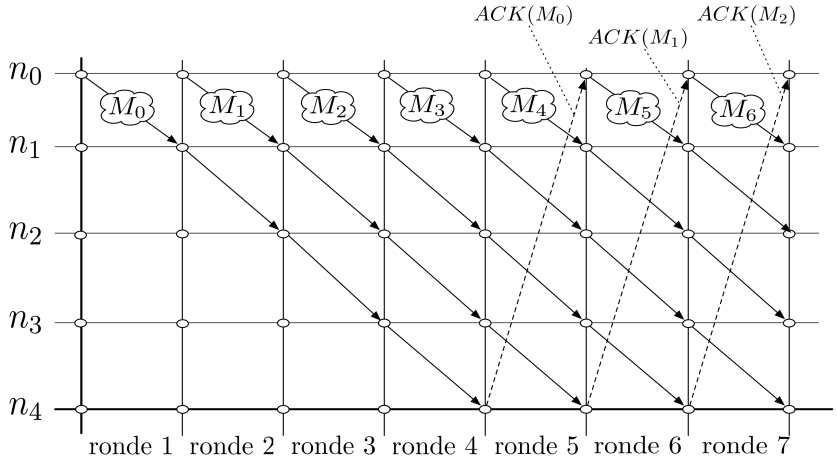


FIG. 3.2 – Débit du protocole FSR.

### 3.2.1 Présentation du protocole

***Hypothèses :***

- `H1. Modèle de communication :` le protocole utilise un modèle de ronde synchrone semblable à celui défini dans la section 1.3. Les horloges des différents noeuds sont parfaitement synchronisées.
- `H2. Topologie du réseau :` un anneau virtuel est construit par dessus le réseau d'interconnexion.
- `H3. Unicité des identifiants :` chaque noeud possède un identifiant unique et il existe un ordre léxicographique sur les identifiants des différents noeuds. Ces identifiants respectent la topologie en anneau : le noeud possédant l'identifiant $n_i$ a pour successeur sur l'anneau le noeud d'identifiant $n_{(i+1) \mod N}$.
- `H4. Pannes franches :` les noeuds du système ne subissent que des pannes franches.
- `H5. Fréquence de panne :` une seule panne peut arriver pendant une même ronde.
- `H6. Mécanisme de vue :` le système dispose d'un mécanisme de vue synchrone utilisant un détecteur de panne parfait. La vue d'un noeud est une structure de donnée qui contient l'ensemble des identifiants des noeuds du systèmes qui sont dans l'anneau. Ce mécanisme fournit la vue initiale "*vue_initiale*" du système et déclenche la procédure `changement_vue(...)` lorsqu'un changement d'état du système est detecté (départ, arrivée ou panne d'un noeud). Ce mécanisme est capable d'informer tous les noeuds du système d'un changement de vue (panne y compris) ayant eut lieu lors d'une ronde $r$ au début de la ronde $r + 1$.
- `H7. Notification de début de ronde :` chaque noeud dispose d'un mécanisme lui notifiant le début d'une nouvelle ronde à l'aide de la primitive `debutDeRonde()`.

On distingue deux types de tâches réalisées par le protocole : l'exécution des procédures qui a lieu au début de chaque ronde et les traitements des événements déclenchés par le système, qui sont pris en charge par des traitants d'événements. Plus particulièrement, le

protocole SCR réagit à trois types d'évennements : la réception d'un message, qui est prise en charge par le traitant `Receive(...)`, la notification d'un changement de vue qui est prise en charge par le traitant `changementDeVue(...)` et la notification du début d'une nouvelle ronde, prise en charge par le traitant `debutDeRonde()`. Le protocole est initialisé sur chaque noeud à l'aide de la procédure `initialize(`*vue_initiale*`)`. Cette procédure à pour but de fournir la valeur de la ronde courante et la vue courante du système à un noeud lorsqu'il se joint au système. Nous allons maintenant détailler les différentes tâches réalisées par le protocole. Le pseudo-code du protocole est disponible sur la Figure 3.3.

`Emission des messages :` lorsqu'un noeud d'identifiant $n_i$ veut diffuser un message $m_i$ en invoquant la procédure `toBroadcast(`$m_i$`)`, ce message est stocké dans une file d'attente nommée $aDiffuser$ (ligne 15). Au début d'une ronde, chaque noeud vérifie s'il doit transmettre un message, qui aurait été reçu lors de la ronde précédente. Si ce n'est pas le cas, il peut diffuser le premier message de sa file d'attente $aDiffuser$ (ligne 51). Comme pour FSR, un noeud à besoin de $N-1$ rondes pour être transmis à tous les noeuds de l'anneau, si aucune panne ne se produit. Si le noeud $n_i$ diffuse le message $m_i$ lors d'une ronde $r$, alors il est sûr que ce message aura été reçu par tous les noeuds du système lors de la ronde $r+N-1$. Par conséquent, le message $m_i$ est estampillé avec une date de livraison $D_{liv}(m_i)$ valant $r+N-1$. Le noeud $n_i$ envoie ensuite l'ensemble $[m_i, n_i, D_{liv}(m_i)]$ à son successeur (le noeud $n_{(i+1) \mod N}$) (ligne 52). Enfin, $n_i$ stocke l'ensemble $[m_i, n_i, D_{liv}(m_i)]$ dans une variable tampon *enAttente*, jusqu'à sa date de livraison.

`Réception des messages :` lorsqu'un noeud reçoit un message, le traitant `Receive(`$[n_j, D_{liv}(m_j), m_j]$`)` est invoqué. Ce traitant vérifie que le message $m_j$ n'a pas déja été reçu, si ce n'est pas le cas alors il le stocke dans sa variable tampon *enAttente* jusqu'à sa date de livraison (ligne 20). Ensuite il vérifie s'il doit transmettre le message à son tour : si le noeud $n_i$ est le prédecesseur du noeud $n_j$ ($n_i = n_{(j-1) \mod N}$), alors il ne retransmet pas le message, car celui a achevé son tour d'anneau. Par contre s'il n'est pas le prédecesseur du noeud $n_j$, il stocke le message $m_j$ dans une variable temporaire $aTransmettre$ afin de retransmettre ce message au début de la ronde suivante (ligne 24).

`Retransmission des messages :` au début de chaque ronde, chaque noeud vérifie s'il doit retransmettre un message : c'est le cas si sa variable $aTransmettre$ est non-nulle. Il transmet alors le message contenu dans la variable $aTransmettre$ à son sucesseur (ligne 47). Si la variable $aTransmettre$ est nulle, il peut diffuser un message selon la procédure détaillée dans le paragraphe `Emission des messages`.

`Ordonnancement total des messages :` la livraison des messages est réalisée par la procédure *tryDeliver*(), qui est appelée à chaque début de ronde. Cette procédure parcourt l'ensemble de la variable tampon *enAttente* et délivre les messages dont la ronde de livraison correspond à la ronde courante, tout en suivant l'ordre des identifiants des émetteurs (lignes 29 à 36). De cette façon l'ordre total sur la livraison des messages est préservé.

### 3.2.2 Gestion des pannes franches

Les pannes sont détectées par le détecteur de pannes parfait utilisé par le mécanisme de vue synchrone. Un exemple de panne est dessiné sur la Figure 3.4.

```
1: procedure initialize(vue_initiale)
2:    enAttente ← ∅
3:    aDiffuser ← ∅
4:    N ← vue_initiale.nbNoeud
5:    numRonde ← vue_initiale.numRondeCourante
6:    m_temp ← ⊥
7:    vue_courrante ← vue_initiale
8:    reprise ← faux
9:    aReprendre ← ⊥
10:   aTransmettre ← ⊥
11:   successeur ← n_((i+1) mod N)
12:   debutDeRonde()
13: fin

14: procedure toBroadcast(m)
15:   aDiffuser ← aDiffuser ∪ {m}
16: fin

17: lorsque Receive([n_j, D_liv(m_j), m_j]) faire
18:   si ¬reprise alors
19:     si m_j ∉ enAttente alors
20:       enAttente ← enAttente ∪ {m_j}
21:       si n_j = successeur alors
22:         aTransmettre ← ⊥
23:       sinon
24:         aTransmettre ←< [n_j, D_liv(m_j), m_j] >
25:       fin si
26:     fin si
27:   fin si
28: fin lorsque

29: procedure tryDeliver()
30:   pour k de 0 à N faire
31:     si ∃[n_k, D_liv(m_k), m_k] ∈ enAttente tel que D_liv(m_k) = numRonde alors
32:       toDeliver(m_k)
33:       enAttente ← enAttente \{[n_k, D_liv(m_k), m_k]}
34:     fin si
35:   fin pour
36: fin

37: lorsque debutDeRonde() faire
38:   tryDeliver()
39:   numRonde ← numRonde +1
40:   si reprise alors
41:     si successeur ≠ vue_courante.successeur alors
42:       envoie aReprendre à successeur
43:     fin si
44:     successeur ← vue_courante.successeur
45:     reprise ← faux
46:   sinon si aTransmettre ≠ ⊥ ∧ ¬reprise alors
47:     envoie aTransmettre à successeur
48:     aReprendre ← aTransmettre
49:     aTransmettre ← ⊥
50:   sinon si aTransmettre = ⊥ ∧ ¬reprise ∧ aDiffuser ≠ ∅ alors
51:     m_temp ← aDiffuser.getFirst()
52:     envoie < [n_i, numRonde + N − 1, m_temp] > à successeur
53:     aReprendre ←< [n_i, numRonde + N − 1, m_temp] >
54:     enAttente ← enAttente ∪ {[n_i, numRonde + N − 1, m_temp]}
55:   fin si
56: fin lorsque

57: lorsque changementDeVue(nouvelle_vue) faire
58:   vue_courrante ← nouvelle_vue
59:   N ← nouvelle_vue.nbNoeud
60:   reprise ← vrai
61:   pour chaque < [n_k, D_liv(m_k), m_k] >∈ enAttente faire
62:     remplacer < [n_k, D_liv(m_k), m_k] > par < [n_k, D_liv(m_k) + 1, m_k] > dans enAttente
63:   fin pour
64:   successeur ← n_((i+1) mod N)
65: fin lorsque
```



FIG. 3.3 – Pseudo-code du protocole SCR, exécuté sur le noeud $n_i$.

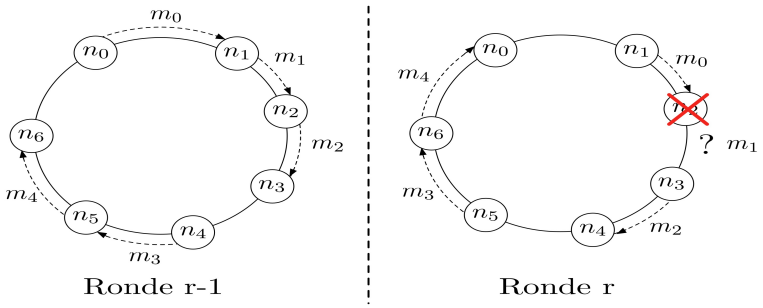


FIG. 3.4 – Détection d'une panne.

Sur cet exemple, le noeud $n_2$ tombe en panne lors de la ronde $r$ alors qu'il était chargé de transmettre le message $m_1$ à son successeur $n_3$. Il est impossible de savoir si $n_2$ est tombé en panne avant ou aprés avoir envoyé le message $m_1$. Nous considérons donc systématiquement que le noeud $n_3$ n'a pas reçu ce message. Une retransmission du message $m_1$ est donc nécessaire. Conformément à nos hypothèses, le mécanisme de vue déclenche le traitant `changement_vue()` avant la fin de la ronde $r$, sur tous les noeuds du système. Ce traitant est chargé, dans un premier temps, de récupérer la nouvelle vue du système et de signaler la panne en activant une variable *reprise* (lignes 58 à 60). Lors de la ronde $r+1$, le noeud $n_1$ (prédécesseur du noeud $n_2$ ayant subit une panne) ré-émet le message $m_1$ potentiellement perdu[1] (Figure 3.5). Les autres noeuds ne font rien (lignes 40 à 45). Par ailleurs, le transfert de tous les autres messages est "figé" lors de la ronde $r+1$, à cause de la retransmission. Le traitant de changement de vue a donc pour rôle de retarder les dates de livraisons de tous les messages contenus dans les tampons *enAttente* de chaque noeud (lignes 61 à 63). Ceci permet de garantir que tous les messages sont bien reçus par tous les noeuds avant leur date de livraison, même si des pannes sont survenues pendant leur acheminement. Les pannes franches ne mettent donc pas en péril le protocole (dont le comportement reste correct). En revanche, chaque panne franche a pour effet d'augmenter d'une ronde la latence des diffusions en cours.

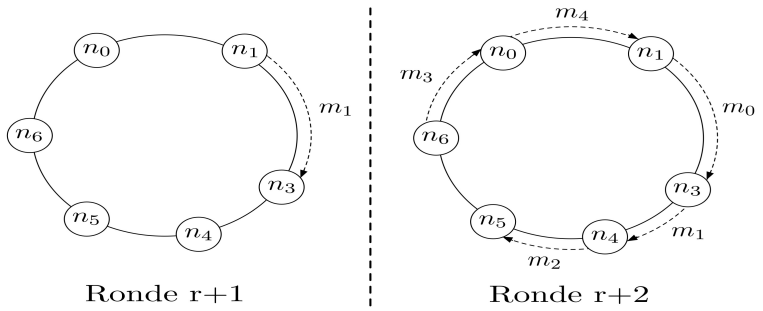


FIG. 3.5 – Recouvrement aprés panne.

[1] Afin que cette ré-émission soit possible, chaque noeud sauvegarde systématiquement le dernier message qu'il a transmis dans une variable *aReprendre* (ligne 48 et 53).

Notons que ce mécanisme permet de traiter le départ volontaire d'un noeud, en considérant ce départ comme une panne. De plus, si un nouveau noeud arrive dans le système, son intégration dans l'anneau peut également être traitée comme une panne. Le noeud recevra les messages retransmis et participera au transfert des messages dés la ronde suivant son arrivée. Ce mécanisme de traitement de panne s'applique donc à tous les changements de vue supportés dans le protocole SCR.

## 3.3 Performances

Dans cette section, nous présentons les performances théoriques du protocole SCR. Nous étudions ensuite l'équité du protocole.

### 3.3.1 Performances théoriques

`Latence :` la latence $L$ d'une diffusion initiée par un noeud est $L = N - 1$. Comme nous l'avons vu ce nombre est égal au nombre de rondes nécessaires pour qu'un message soit reçu par tous les noeuds du système.

`Complexité en messages :` la complexité du protocole SCR est de $(N - 1)$ messages.

`Débit :` dans le cas où un seul noeud a un nombre infini de messages à diffuser, le protocole SCR garantit un débit de 1, car le noeud diffusant des messages peut initier une nouvelle diffusion lors de chaque ronde. Ce résultat est similaire à celui de FSR illustré sur la Figure 3.2. Dans le cas où $N$ noeuds ont une infinité de messages à diffuser, le débit est borné par $\frac{N-1}{N-1} * D_{opt} = D_{opt}$.

Ces différents résultats ont été confirmés par des simulations réalisées à l'aide du simulateur Peersim [1].

### 3.3.2 Etude de l'équité

Les performances garanties par le protocole SCR correspondent aux attentes formulées : SCR améliore significativement le débit des protocoles existants. En revanche, le protocole SCR soulève un problème d'équité qui n'existait pas dans les protocoles de l'état de l'art. L'équité est une notion utilisée dans de nombreux domaines de l'informatique (systèmes distribués, allocation de ressource sur grille de calcul, ordonnancement de tâches dans un système d'exploitation, etc.) dont le principe est simple : lorsqu'une ressource doit être partagée entre plusieurs entités, une répartition équitable consiste à leur allouer la même quantité de cette ressource.

Dans notre contexte, les entités sont les noeuds du système et la ressource à partager est l'accés à la primitive de diffusion de messages. La contrainte d'équité que nous voulons exprimer peut donc être définit comme suit : soit un système distribué dans lequel un nombre $k$ de noeuds possédent un nombre infini de messages à diffuser. Si $n * k$ diffusions ont été réalisées, alors chaque noeud aura diffusé $n$ messages, et ces messages auront été délivrés par tous les noeuds du système.

Le protocole SCR n'est pas équitable. Plus grave, il peut engendrer des cas de famine : certains noeuds désirant diffuser des messages peuvent ne pas y parvenir. Ceci se produit

lorsqu'un (ou plusieurs) noeud(s) innonde(nt) l'anneau de messages durant un temps infini (Figure 3.6). Un tel cas de figure est représenté sur la Figure 3.6 : le noeud $n_0$ possède un nombre infini de messages à diffuser. Il inonde donc l'anneau avec ses messages ($M_0$ à $M_4$ sur la figure). Si les autres noeuds (à l'exception de n5) ont des messages à diffuser, ils ne peuvent pas le faire car ils doivent prioritairement faire suivre les messages de $n_0$. En effet, comme on le remarque sur la Figure 3.6, seul le noeud précédant $n_0$ sur l'anneau ($n_5$) peut disposer d'un créneau pour diffuser des messages. Si $n_5$ n'a jamais de message à diffuser, alors $n_0$ n'aura jamais de message à faire suivre et pourra de ce fait initier une nouvelle diffusion lors de chaque ronde. Les autres noeuds ne pourront donc jamais diffuser leurs messages.

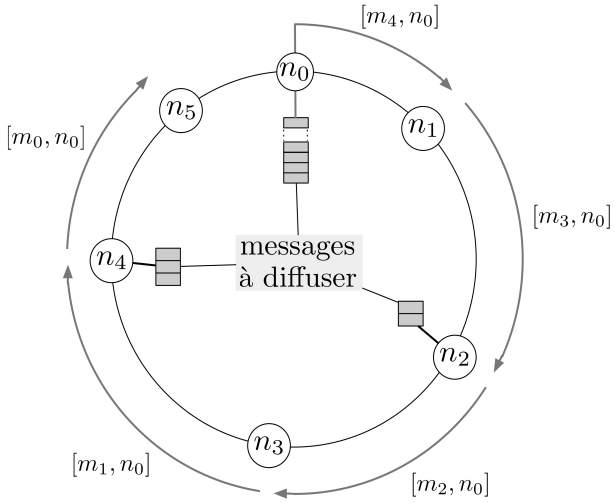


FIG. 3.6 – Famine dans SCR.

Ce phénomène de famine est du au fait que l'on privilégie le relais d'un message par rapport à une nouvelle diffusion. Cette priorité est essentielle pour garantir le fonctionnement du protocole. Effectivement, si les noeuds sont autorisés à retarder la transmission des messages pour initier leur diffusion, il n'est plus possible de connaître a priori la date de délivrance des messages.

## 3.4 Synthèse

Dans ce chapitre, nous avons présenté SCR, un protocole basé sur une organisation en anneau virtuel des noeuds du système. SCR est inspiré du protocole FSR qui a été conçu pour des systèmes asynchrones. L'utilisation d'un anneau virtuel permet à SCR de garantir un débit supérieur aux protocoles existants. Néanmoins, SCR a un inconvénient majeur par rapport aux protocoles existants : il n'est pas équitable. C'est la raison pour laquelle nous avons proposé un second protocole qui est présenté dans le chapitre suivant.

# Chapitre 4

# Le protocole SPA

Dans ce chapitre, nous présentons SPA, un protocole de diffusion totalement ordonnée que nous avons élaboré avec pour objectif d'améliorer SCR sur deux points : la latence et l'aspect équitable. Nous inspirant des travaux existants étudiés dans l'état de l'art, il nous a paru intéressant d'abandonner la topologie en anneau pour utiliser une topologie de réseau complètement connectée. Effectivement, les protocoles de diffusion totalement ordonnée basés sur l'utilisation d'un privilège ont pour la plupart une latence d'une ronde [2, 11], ce qui est impossible dans le cas où une topologie en anneau est utilisée.

La particularité du protocole SPA est qu'il propose un algorithme d'ordonnancement des privilèges ; d'où son nom : "Scheduled Privilege Algorithm". Ceci le distingue des protocoles étudiés dans l'état de l'art dans lesquels le passage du privilège d'un noeud à un autre est systématique et connu a priori. Ce chapitre débute par une présentation intuitive du protocole. Nous en expliquons ensuite le fonctionnement en détail. Nous évaluons ensuite les performances du protocole de façon théorique et à l'aide de simulations.

## 4.1 Présentation du protocole

Dans le protocole SPA, un seul noeud possède le privilège de diffuser un message pour une ronde donnée. Afin de bien comprendre la façon dont les privilèges sont organisés dans ce protocole, nous utiliserons la notion de tour dans la description de son fonctionnement : un tour est formé de $N$ rondes consécutives. Chaque tour est noté sous la forme $T_i$. Les rondes composant le tour $T_i$ sont notées sous la forme $r_{i,j}$, où $r_{i,0}$ est la ronde initiant le tour $T_i$ et $r_{i,N-1}$ la ronde le cloturant.

### 4.1.1 Utilisation du privilège

Dans un premier temps, nous proposons d'étudier l'ordonnancement classique qui revient à donner à chaque noeud le privilège de diffusion une fois par tour. Cet ordonnancement attribue le privilège au noeud $n_j$ lors des rondes $r_{X,j}$. L'attribution du privilège assure que la répartition du privilège est équitable. L'ordre total est assuré en livrant simplement les messages dés leur réception. En effet, l'utilisation du privilège garantit un accès en exclusion mutuelle aux canaux de communication, assurant que chaque diffusion est atomique[1]. Le

[1]Le message d'une diffusion est livré à tous les noeuds du système avant qu'une autre diffusion ne commence.

principe de cet ordonnancement est illusté sur la Figure 4.1, pour une exécution de deux tours dans un système composé de cinq noeuds.

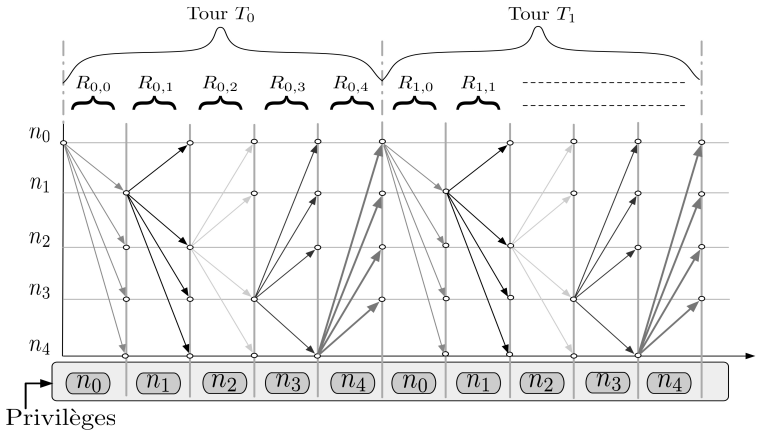


FIG. 4.1 – Ordonnancement classique du privilège.

Comme nous l'avons vu dans le chapitre consacré à l'état de l'art, un tel ordonnancement n'est pas optimal en débit lorsqu'un seul noeud possède une infinité de messages à diffuser. Ceci est dû au fait que le privilège de la diffusion est systématiquement attribué à tous les noeuds pendant un même tour, sans se soucier de leur souhait de diffusion. Les rondes pendant lesquelles le privilège est attribué aux noeuds silencieux[2] s'écoulent sans qu'aucune diffusion ne s'exécute, alors que d'autres noeuds pourraient profiter de ce créneau pour diffuser leurs messages quand ils en ont.

Partant de ce constat, nous proposons dans la suite de ce rapport un nouveau mécanisme d'ordonnancement du privilège qui est utilisé dans SPA. Ce mécanisme tient compte des souhaits de diffusion de tous les noeuds du système. Il permet aux noeuds désirant diffuser plusieurs messages par tour de disposer des créneaux inutilisés, s'il en existe. Ce principe est représenté sur la Figure 4.2 : le système est composé de cinq noeuds, dont deux silencieux ($n_3$ et $n_4$). Les privilèges des rondes $r_{i,3}$ et $r_{i,4}$ sont attribués aux noeuds $n_2$, puis $n_1$, optimisant de ce fait le débit des diffusions exécutées pendant le tour $T_i$. Cet ordonnancement du privilège permet d'atteindre un débit de 1, mais il nécessite de savoir dynamiquement quels noeuds ont des messages à diffuser. Nous allons donc maintenant introduire le calcul de l'ordonnancement du privilège. Ce calcul est effectué au début de chaque tour.

### 4.1.2 Calcul de l'ordonnancement du privilège

Nous présentons un mécanisme permettant de calculer, au début de chaque tour, un ordonnancement des privilèges pour chacune des rondes du tour. En plus d'optimiser le débit de la diffusion, l'ordonnancement des privilèges doit être équitable. Par exemple, si tous les noeuds désirent diffuser des messages au début du tour, alors une ronde doit être attribuée à chacun d'eux. Afin de réaliser cet ordonnancement, le protocole associe une variable d'état *souhait* à chaque noeud. Cette variable contient le nombre de messages que ce noeud désire

[2]Un noeud silencieux est un noeud qui ne diffuse pas de message.

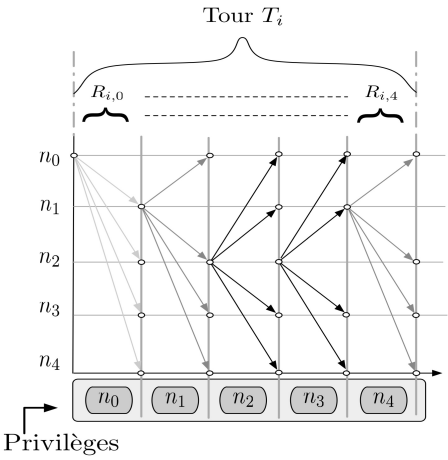


FIG. 4.2 – Ordonnancement du privilège dans le protocole SPA.

diffuser. Afin que le mécanisme d'ordonnancement tienne compte des souhaits de diffusion, chaque noeud diffuse la valeur de sa variable *souhait* au moins une fois par tour. A la fin du tour, tous les noeuds ont donc la même connaissance des souhaits de tous les autres noeuds du système et l'ordonnancement des privilèges du tour suivant peut s'effectuer. Dans le cas où aucun noeud n'est silencieux, chacun peut ajouter cette information au message qu'il diffuse. Cependant, un problème se pose lorsque certains noeuds sont silencieux : si ces noeuds ne souhaitent pas diffuser de message pendant un certains temps, ils perdent le privilège de la diffusion et ne peuvent donc pas communiquer la valeur de leur variable de souhait aux autres noeuds. Ce problème est grave, car priver les noeuds de communiquer leur variable *souhait* revient à les priver d'obtenir le privilège de diffuser, ce qui se traduit en une violation de la propriété d'équité.

Pour remédier à ce problème, nous avons introduit, dans le protocole SPA, la notion de *co-émetteur* : un co-émetteur est un noeud silencieux qui ne possède aucun privilège pendant un tour donné. Néanmoins, ce co-émetteur est autorisé à envoyer un message contenant sa variable *souhait* à un noeud possédant plusieurs privilèges pendant ce même tour. En effet, en observant les traces d'exécution du protocole SPA (Figure 4.2), il apparait que pour chaque ronde, le noeud diffusant un message ne reçoit rien[3]. Le co-émetteur peut donc lui transmettre la valeur de sa variable *souhait*. Notons que le noeud recevant cette information de souhait doit impérativement disposer d'un second privilège avant la fin du tour, de façon à pouvoir diffuser cette information à tous les autres noeuds avant la fin du tour. Pour garantir cela, un ordonnancement des "co-privilèges" est calculé, en se basant sur l'ordonnancement des privilèges. Un noeud possédant plus d'un privilège par tour est dit "super-émetteur". Le co-privilège d'une ronde $r$ doit être alloué de façon à ce que : (1) le noeud bénéficiant du co-privilège soit silencieux dans le tour considéré, (2) le noeud recevant le message émis par le co-émetteur doit être un super-émetteur et posséder un privilège de diffusion pendant une autre ronde du même tour et ultérieure à la ronde $r$.

Le principe de l'ordonnancement du privilège et du co-privilège est illustré sur la Fi-

[3]On rappelle que dans notre modèle de ronde, la réception d'un message qu'un noeud s'envoie à lui même n'est pas comptée dans sa capacité de réception.

gure 4.3. Sur cet exemple, le système est composé de cinq noeuds, dont deux super-émetteurs ($n_0$ et $n_1$) et deux co-émetteurs ($n_3$ et $n_4$). Le noeud $n_3$ possède le co-privilège à la ronde 1 à destination du noeud $n1$, alors que le noeud $n_4$ le possède à la ronde 2 à destination du noeud $n_2$. Ainsi, tous les noeuds, y compris les noeuds silencieux, peuvent diffuser la valeur de leur variable *souhait* à tous les autres noeuds, avant la fin du tour $T_i$. Le mécanisme d'ordonnancement peut donc tenir compte des souhaits de chaque noeud pour le tour $T_{i+1}$. En l'occurrence, dans le cas présenté, l'ensemble des noeuds souhaitaient diffuser un message, ce qui se traduit par l'allocation d'une ronde à chaque noeud.

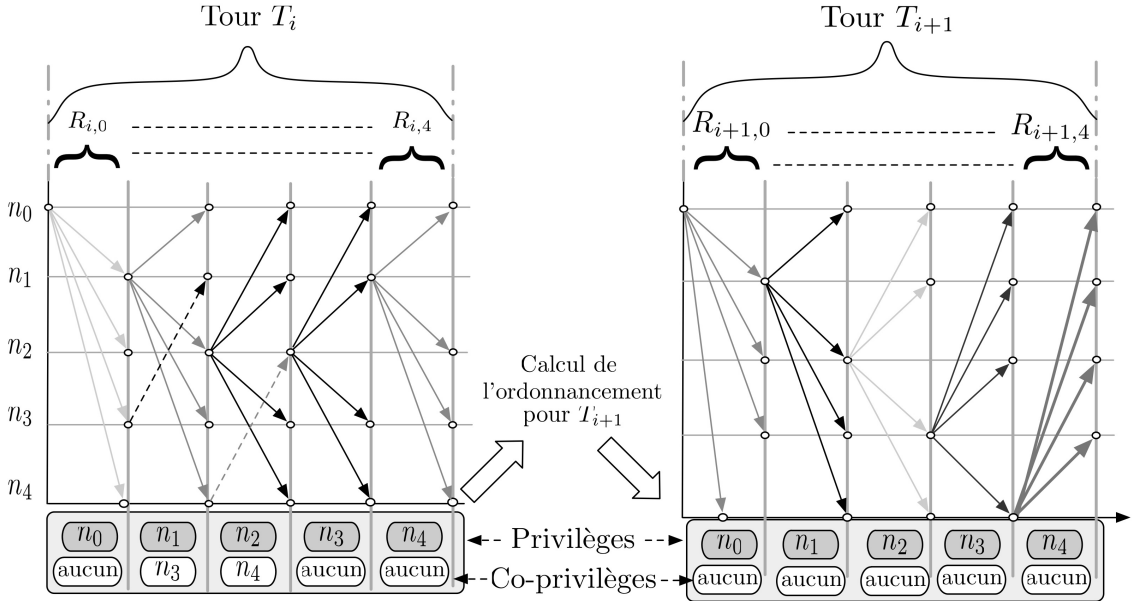


FIG. 4.3 – Ordonnancement du privilège et du co-privilège dans le protocole SPA.

### 4.1.3 Respect de l'équité

Comme nous le mentionnions en introduction de ce chapitre, l'un des objectifs que nous nous sommes fixés lors de la conception du protocole SPA est de garantir l'équité de l'allocation du privilège entre les noeuds. Lorsque le nombre de noeud silencieux dans le système est non nul pendant plusieurs tours, les privilèges supplémentaires doivent être répartis de façon équitable entre les noeuds non-silencieux. Afin de respecter l'équité, le protocole SPA utilise une variable nommée *historique* qui permet de maintenir à jour un historique des privilèges qui sont alloués à chaque noeuds au cours des ordonnancements successifs. Cette variable est incrémentée chaque fois qu'un noeud reçoit un privilège supplémentaire, i.e. lorsqu'il reçoit un privilège qui aurait été attribué à un noeud silencieux. Le protocole fait en sorte que chaque noeud se voit allouer un nombre égal de privilèges supplémentaires. Notons qu'il est possible de spécifier un ensemble de noeuds auxquels les privilèges supplémentaires seront systématiquement alloués.

### 4.1.4 Gestion des pannes

Comme dans le cas de SCR, nous faisons l'hypothèse qu'uniquement des pannes franches peuvent survenir. Par ailleurs, nous faisons l'hypothèse de disposer d'un détecteur de pannes parfait. Contrairement au protocole SCR, les pannes peuvent être traitées de façon très simple. En effet, les messages ne sont pas transmis de noeuds en noeuds comme dans une structure en anneau, ce qui facilite grandement la reprise sur panne. Lorsqu'à la fin d'une ronde, le détecteur de panne notifie l'occurrence d'une pannes aux autres noeuds, ceux-ci reprennent immédiatement l'ordonnancement du privilège initial dans lequel chaque noeud dispose d'une ronde. Après un tour, les mécanismes de co-privilèges se ré-activeront. Notons que par soucis de clarté, ce mécanisme n'est pas décrit dans le pseudo-code du protocole.

## 4.2 Implantation du protocole SPA

Dans cette section, nous présentons les détails du protocole SPA. Le pseudo-code est représenté sur la Figure 4.4. Nous débutons par la liste des hypothèses. Nous présentons ensuite les variables d'états utilisées par le protocole. Enfin, nous en décrivons le pseudo-code. La preuve du protocole est donnée en Annexe.

### 4.2.1 Hypothèses

- `H1. Modèle de communication :` le protocole SPA utilise un modèle de rondes synchrone. Les horloges des différents noeuds sont supposées parfaitement synchronisées.
- `H2. Topologie du réseau :` le réseau d'interconnexion est supposé complètement connecté.
- `H3. Unicité des identifiants :` chaque noeud possède un identifiant unique et il existe un ordre léxicographique sur les identifiants des différents noeuds.
- `H4. Pannes franches :` les noeuds du système ne subissent que des pannes franches.
- `H5. Mécanisme de vue :` le système dispose d'un mécanisme de vue synchrone utilisant un détecteur de pannes parfait. Comme dans le cas de SCR, ce mécanisme fournit la vue initiale *vueInitiale* du système.
- `H6. Notification de début de ronde :` chaque noeud dispose d'un mécanisme lui notifiant le début d'une nouvelle ronde à l'aide de la primitive `debutDeRonde()`.

### 4.2.2 Variables d'états et initialisation

Dans cette section, nous présentons toutes les variables d'états présentes sur chaque noeud, ainsi que leur fonction :

- `aEnvoyer :` liste contenant les messages à diffuser.
- `silencieux :` liste contenant les identifiants des noeuds déclarés silencieux pour le tour courant.
- `superEmetteurs :` liste contenant les identifiants des super-émetteurs du tour courant.
- $N$ : nombre de noeuds dans le système.
- `numRonde :` numéro de la ronde courante.
- `debutTourCourant :` numéro de la ronde correspondant au début du tour courant.
- `debutDeTour :` variable booléenne qui indique si la ronde courante correspond au début d'un nouveau tour.

– `decalage` : décalage en nombre de ronde entre la ronde courante et le début du tour courant.
– `nbEmetteur` : nombre de noeuds non-silencieux du tour courant.
– `reçu` : variable tampon servant à stocker les messages reçus.
– `souhait` : tableau de taille N contenant tous les souhaits de tous les noeuds. L'équivalent de la variable *souhait* d'un noeud $n_i$ présenté précédement correspond à la valeur $souhait[i]$.
– `privilège` : tableau de taille $N$ associant à chaque ronde du tour courant l'identifiant du noeud qui à le privilège de la diffusion pendant cette même ronde.
– `coPrivilège` : tableau de taille $N$ associant à chaque ronde du tour courant l'identifiant du noeud qui à le co-privilège.
– `historique` : tableau de taille $N$ associant à chaque noeud le nombre de privilèges supplémentaires qu'il a eut.

### 4.2.3 Traitants d'événements et procédures

Deux types de tâches sont réalisées par le protocole SPA : l'exécution des procédures qui ont lieux au début de chaque ronde et les traitements des événements déclenchés par le système. Deux types d'événements peuvent survenir : la réception d'un message — prise en charge par le traitant `Reception(...)` — et la notification du début d'une nouvelle ronde — prise en charge par le traitant `debutDeRonde()`. Nous allons maintenant détailler les différentes tâches réalisées par le protocole.

`Initialisation` : l'initialisation du protocole est réalisée à l'aide de la procédure initialisation(*vueInitiale*), où *vueInitiale* est une variable contenant des informations sur la composition initiale du système (lignes 1 à 13). Une procédure `initTable()` initialise l'ordonnancement initial des privilèges de façon à ce que tous les noeuds possèdent strictement un privilège lors du premier tour (lignes 14 à 21).

`Début d'une nouvelle ronde` : lorsque l'événement `debutDeRonde()` est déclenché, chaque noeud met à jour les valeurs de la ronde courante et vérifie si un nouveau tour commence en exécutant la procédure `nouvelleRonde()` (lignes 44 à 54). Ensuite, chaque noeud vérifie s'il a reçu un message à délivrer en exécutant la procédure `remise()`, qui livrera ce message à l'aide de la primitive `toDeliver(m)` (lignes 55 à 60). Comme il a été montré lors de l'introduction du principe de SPA, la remise de message n'effectue aucun traitement spécifique, puisque l'utilisation des privilèges assure l'ordre total. Si un nouveau tour commence, alors les procédures d'ordonnancement du privilège et du co-privilège (`ordonnerPrivilege()` et `ordonnerCoPrivilege()`) sont invoquées. Enfin, la procédure `envoie()` est invoquée afin de diffuser un message ou d'envoyer une information si le noeud en a le privilège.

`Emission des messages` : lorsqu'un noeud d'identifiant $n_i$ veut diffuser un message $m_i$ en invoquant la procédure `toBroadcast(`$m_i$`)`, ce message est stocké dans une file d'attente nommée *aEnvoyer* (ligne 23). Ensuite, lorsque la procédure `envoie()` est invoquée au début de chaque ronde, le noeud $n_i$ vérifie s'il possède le co-privilège associé à la ronde courante, si c'est le cas, il envoie le contenu de sa variable souhait associée à son identifiant au noeud possédant le privilège(lignes 62 à 64). Si $n_i$ possède le privilège de la diffusion, alors il diffuse

l'ensemble $[n_i, m, S_i]$, où $n_i$ est son identifiant, $S_i$ sa variable *souhait* et $m$ le premier message contenu dans la liste *aEnvoyer*, s'il en existe un. Si la liste *aEnvoyer* est vide, alors un message vide est envoyé à la place de $m$ (lignes 65 à 73).

`Réception des messages :` le traitant d'événement reception($n_j$,m,$S_j$) stocke l'ensemble $[n_j$,m,$S_j]$ dans la variable *reçu*, puis met à jour les variables *souhait* correspondant au noeud $n_j$ (lignes 34 à 43).

`Ordonnancement du privilège :` la procédure ordonnerPrivilège() est toujours invoquée pendant une ronde qui débute un nouveau tour. Elle définit l'ordre selon lequel les noeuds posséderont le privilège de diffuser des messages, jusqu'à la fin du tour qui commence (i.e. pendant la ronde courante et pendant les $N$ rondes qui suivent). Elle vérifie si tous les noeuds du système souhaitent diffuser des messages. Si elle trouve un noeud qui ne souhaite pas diffuser de message, elle déclare ce noeud comme silencieux en ajoutant son identifiant à la variable *silencieux*. Puis elle élit un noeud souhaitant diffuser plus d'un message pendant le tour qui vient pour lui allouer une ronde de diffusion supplémentaire. Ce noeud élu est alors déclaré comme étant super-émetteur et son identifiant est placé dans la variable *superEmetteur*. Le critère d'élection des noeuds pour devenir super-émetteur repose sur l'utilisation de la variable *historique*, conformément à ce qui a été définie dans la section précédente (lignes 75 à 87).

`Ordonnancement du co-privilège :` la procédure ordonnerCoprivilège() est également toujours invoquée pendant une ronde qui débute un nouveau tour. Elle parcourt la liste des noeuds super-émetteurs, puis alloue un co-privilège à un des noeuds silencieux pendant la (les) première(s) ronde(s) où un des super-émetteurs a le privilège de diffuser un message (lignes 88 à 100). Ainsi le protocole s'assure que le super-émetteur en question aura au moins une fois de plus le privilège de diffuser des messages pendant le tour courant. Notons bien qu'un même noeud peut apparaître plusieurs fois dans la liste des super-émetteurs.

## 4.3 Performances

Dans cette section, nous débutons par une analyse théorique des performances du protocole SPA. Nous donnons ensuite le résultat de simulations que nous avons effectuées afin de vérifier la validité de l'analyse théorique.

### 4.3.1 Performances théoriques

`Latence :` dans le protocole SPA, la latence d'une diffusion est de 1 ronde.

`Complexité en messages :` la complexité en messages dans le protocole SPA est de $N - 1$ messages.

`Débit :` dans le cas où un seul noeud diffuse une infinité de messages, le débit théorique du protocole SPA est de 1. Ceci est du au fait que le noeud ayant une infinité de messages à émettre pourra diffuser un message par ronde. Dans le cas où tous les noeuds possèdent une infinité de messages à diffuser, le débit théorique de diffusion est borné par $\frac{N}{N-1}$. Néanmoins, le débit sera toujours égal à 1. En effet, de façon similaire au cas où un seul noeud a une

```
1: procedure initialisation(vueInitiale)
2:    aEnvoyer ← ∅
3:    silencieux ← ∅
4:    superEmetteurs ← ∅
5:    N ← vueInitiale.nbNoeuds
6:    numRonde ← 0
7:    debutTourCourant ← 0
8:    debutDeTour ← true
9:    decalage ← 0
10:   nbEmetteur ← 0
11:   reçu ← ⊥
12:   initTable()
13: fin

14: procedure initTable()
15:    pour chaque k ∈ [0 .. N − 1] faire
16:       souhait[k] ← 0
17:       historique[k] ← 0
18:       privilege[k] ← k
19:       coPrivilege[k] ← -1
20:    fin pour
21: fin

22: procedure toBroadcast(m)
23:    aEnvoyer ← aEnvoyer ∪ {m}
24: fin

25: lorsque debutDeRonde() faire
26:    nouvelleRonde()
27:    remise()
28:    si debutDeTour alors
29:       ordonnerPrivilege()
30:       ordonnerCoprivilege()
31:    fin si
32:    envoie()
33: fin lorsque

34: lorsque reception(n_j,m,S_j) faire
35:    si reçu ≠ ⊥ alors
36:       reçu ← m
37:    fin si
38:    pour chaque k ∈ [0 .. N − 1] faire
39:       si k ≠ i alors
40:          souhait[k] ← S_j[k]
41:       fin si
42:    fin pour
43: fin lorsque

44: procedure nouvelleRonde()
45:    numRonde ← numRonde + 1
46:    decalage ← numRonde − debutTourCourant
47:    si decalage = N alors
48:       debutDeTour ← true
49:       debutTourCourant ← numRonde
50:       decalage ← 0
51:    sinon
52:       debutDeTour ← false
53:    fin si
54: fin

55: procedure remise()
56:    si reçu ≠ ⊥ alors
57:       toDeliver(reçu)
58:       reçu ← ⊥
59:    fin si
60: fin

61: procedure envoie()
62:    si n_i = coPrivilege[i] alors
63:       souhait[i] ← |aEnvoyer|
64:       envoie(n_i,⊥,souhait) à privilege[i]
65:    sinon si n_i = privilege[i] ∧ aEnvoyer ≠ ∅ alors
66:       souhait[i] ← |aEnvoyer|
67:       m_temp ← aEnvoyer.getFirst()
68:       envoie(n_i,m_temp,souhait) à chaque n_k
69:       aEnvoyer ← aEnvoyer \ {m_temp}
70:    sinon si n_i = privilege[i] ∧ aEnvoyer = ∅ alors
71:       souhait[i] ← |aEnvoyer|
72:       envoie(n_i,⊥,souhait) à chaque n_k
73:    fin si
74: fin

75: procedure ordonnerPrivilege()
76:    pour chaque k ∈ [0 .. N − 1] faire
77:       si souhait[k] = 0 ∧ souhait[k] ≤
          superEmetteurs.retournerNombreInstance(k)
          alors
78:          silencieux ← silencieux ∪ {k}
79:          elu ← r | souhait[r] ≠ 0 ∧ ∀ i | souhait[i] ≠ 0,
             historique[r] ≤ historique[i]
80:          privilege[k] ← elu
81:          historique[elu] ← historique[elu] + 1
82:          superEmetteurs ← superEmetteurs ∪ {elu}
83:       sinon
84:          privilege[k] ← k
85:       fin si
86:    fin pour
87: fin

88: procedure ordonnerCoprivilege()
89:    cpt ← 0
90:    tant que silencieux ≠ ∅ ∧ superEmetteurs ≠ ∅
       faire
91:       si privilege[cpt] ∈ superEmetteurs alors
92:          coPrivilege[cpt] ← cs | cs ∈ silencieux
93:          silencieux ← silencieux \ {cs}
94:          superEmetteurs ← superEmetteurs \
             {privilege[cpt]}
95:       sinon
96:          coPrivilege[cpt] ← −1
97:       fin si
98:       cpt ← cpt + 1
99:    fin tant que
100: fin
```

FIG. 4.4 – Pseudo-code du protocole SPA pour le noeud $n_i$.

infinité de messages à diffuser, le protocole permet de diffuser et délivrer un message par ronde.

### 4.3.2 Simulation

Afin de confirmer l'analyse théorique des performances du protocole SPA, nous avons réalisé une simulation à l'aide du simulateur Peersim [1]. Peersim est un simulateur de protocoles distribués, basé sur le modèle de programmation objet et réalisé en Java. Il associe un objet à chaque noeud du système et nécessite le développement d'une méthode *Reception* qui permet de traiter les réceptions de messages. Peersim modélise un système synchrone. Durant chaque ronde, il appelle la méthode *Reception* de l'ensemble des noeuds. Par ailleurs, un noeud peut utiliser une méthode *envoie* pour émettre des messages à destination d'autres noeuds. Chaque noeud ne peut émettre et recevoir qu'un message par ronde, ce qui est conforme à notre modèle de performance. Notons, par ailleurs, que le simulateur Peersim permet de définir la topologie du réseau d'interconnexion des noeuds.

Nous avons étendu la version officielle de Peersim [1], afin de permettre le calcul de la latence moyenne des diffusions réalisées au cours d'une simulation, ainsi que de calculer le débit moyen de l'ensemble des diffusions. Chaque simulation que nous effectuons dure 500 rondes, afin d'obtenir des valeurs moyennes de la latence et du débit qui ne sont pas perturbées par la phase d'initialisation du protocole. Afin d'évaluer la latence moyenne, lorsqu'un message est délivré par chaque noeud, sa latence de diffusion est stockée dans un registre commun à tous les noeuds. A la fin de la simulation, nous calculons la moyenne des latences de tous les messages qui ont circulé sur le système de communication pendant la simulation. Le débit quand à lui est égal au nombre moyen de messages délivrés par ronde. Enfin, afin de vérifier que le protocole SPA est bien équitable, nous avons évalué la participation de chaque noeud aux diffusions réalisées par le protocole, en calculant le pourcentage des diffusions de chaque noeud par rapport au nombre total de diffusions réalisées pendant la simulation.

Nous avons simulé le protocole sur neuf systèmes différents, dont le nombre de noeuds varie de 2 à 10. Pour chacun de ces systèmes, nous avons simulé tous les cas d'exécution correspondant au cas où $k$ noeud(s) désirent diffuser une infinité de message, $k$ allant de 1 à N. Nous avons donc en tout réalisé plus de 200 itérations de cette simulation du protocole SPA. Nous avons relevé trois constantes dans les résultats de ces simulations :

- La latence moyenne de diffusion est toujours égale à 1.
- Le débit moyen de diffusion est toujours égale à 1.
- Tous les noeuds non-silencieux ont la même probabilité de diffuser leurs messages.

Ces simulations confirment donc l'analyse théorique des performances du protocole SPA. Celui-ci possède des performances qui ne dépendent pas du nombre de noeuds présents dans le système, ni du nombre de noeuds qui diffusent des messages. Par ailleurs, le protocole SPA garantit un débit et une latence optimaux. Cette simulation comfirme également que le protocole SPA est équitable.

## 4.4 Synthèse

Dans ce chapitre, nous avons présenté le protocole de diffusion totalement ordonnée SPA. Celui-ci garantit une latence et une complexité en messages optimales. Par ailleurs, il assure un débit constant de 1 messages par ronde. Enfin, il assure l'équité d'accès à la diffusion. Le

mécanisme d'ordonnancement du privilège utilisé dans ce protocole s'avère donc plus efficace que la stratégie de diffusion en anneau adoptée dans SCR.

# Conclusion

Dans ce projet de M2R, nous avons étudié les protocoles de diffusion totalement ordonnée pour les systèmes distribués synchrones. Nous avons utilisé trois métriques pour évaluer de façon théorique les performances de ces protocoles : la latence, la complexité en messages et le débit. Une étude des protocoles existant a montré qu'aucun de ces protocoles ne fournissait un débit efficace dans les différents cas d'utilisation considérés dans ce travail. En effet, la présence d'aqcuittements et de consensus dans les protocoles utilisant un historique des communications engendre des débits faibles dans tous les cas d'utilisation. Quant aux protocoles basés sur l'utilisation d'un privilège, ils ne garantissent un débit satisfaisant que lorsque tous les noeuds ont des messages à diffuser. Effectivement, ces protocoles ne prennent pas en compte les souhaits de diffusions des noeuds pour allouer le privilège de diffuser, ce qui engendre une baisse du débit de diffusion si au moins un noeud ne souhaite pas diffuser de messages.

Nous avons proposé deux protocoles de diffusion totalement ordonnée. Le premier, appelé SCR, est basé sur une diffusion des messages à l'aide d'un anneau virtuel. Il garantit un débit optimal dans tous les cas d'utilisation considérés. Néanmoins, ce protocole souffre d'une limitation importante : il est sujet au problème de famine. Lorsque plusieurs noeuds ont une infinité de messages à diffuser, certains noeuds peuvent ne pas avoir l'opportunité de diffuser leurs messages. Par ailleurs, la latence du protocole SCR est linéaire en fonction du nombre de noeuds dans le système, ce qui n'est pas optimal. Le second protocole que nous avons conçu, appelé SPA, garantit un débit théorique très proche du débit optimal, tout en étant optimal en latence et équitable. Ce protocole repose sur l'utilisation d'un privilège qui est alloué selon les besoins des noeuds du système. Des simulations ont validé ces différents résultats.

Nous souhaitons désormais explorer différentes pistes de travail. Tout d'abord, nous souhaitons étudier l'impact des mécanismes de gestion des pannes sur les protocoles réalisés. S'il est clair que ces mécanismes ne perturbent pas le fonctionnement du protocole lorsqu'aucune panne ne survient, il n'est pas évident, en revanche, qu'ils garantissent une perturbation minimale de fonctionnement pendant la gestion d'une panne. Nous souhaitons donc les améliorer afin de garantir une fenêtre de vulnérabilité du protocole minimale. Une deuxième piste d'étude envisagée est la possibilité d'adapter le protocole SPA pour des systèmes distribués asynchrones. Ces systèmes ne font pas les hypothèses de synchronie sur lesquelles nous nous sommes basées pour élaborer les protocoles de diffusion présentés dans ce rapport. Nous souhaitons étudier la possibilité d'adapter le mécanisme d'ordonnancement des privilèges de SPA à ces systèmes. Enfin, nous souhaitons réaliser des évaluations de performances des protocoles proposés sur un système distribué réel, afin de confronter les résultats théoriques obtenus aux résultats pratiques. Le modèle de communication utilisé étant réaliste (un noeud ne peut émettre et recevoir qu'un noeud par ronde), nous nous attendons à ce que les mesures pratiques réalisées confirment les résultats théoriques présentés dans ce rapport.

# Bibliographie


[1] http ://peersim.sourceforge.net/.

[2] T. Abdelzaher, A. Shaikh, F. Jahanian, and K. Shin. Rtcast : lightweight multicast for real-time process groups. *RTAS '96 : Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS '96)*, 1996.

[3] Anceaume.e and Minet.p. Etude de protocole de diffusion atomique. *TR174 INRIA, Rocquencourt, France*, 1992.

[4] Ziv Bar-Joseph, Idit Keidar, and Nancy A. Lynch. Early-delivery dynamic atomic broadcast. *Proceedings of the 16th International Conference on Distributed Computing*, 2002.

[5] Bharali A.A. Berman P. Quick atomic broadcast (extended absrtact). *In Proceedings of 7th Internaitional Workshop on Distributed Algorithms (WDAG'93) (Lausanne Switzerland)*, 1993.

[6] F. Cristian, H. Aghali, R. Strong, and D. Dolev. Atomic broadcast : From simple message diffusion to byzantine agreement. *Proc. 15th Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, 1995.

[7] Xavier Défago, André Schiper, and Péter Urban. Total order broadcast and multicast algorithms : Taxonomy and survey. *ACM Computing Surveys (CSUR)*, 36(4) :372–421, 2004.

[8] Ajei S. Gopal and Sam Toueg. Reliable broadcast in synchronous and asynchronous environments (preliminary version). *Proceedings of the 3rd International Workshop on Distributed Algorithms*, 1989.

[9] Rachid Guerraoui, Ron R. Levy, Bastian Pochon, and Vivien Quema. High throughput total order broadcast for cluster environments. *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, 2006.

[10] Rachid Guerraoui and Luis Rodrigues. *Lecture notes,Introduction to Reliable Distributed Programming.* Springer-Verlag, 2005.

[11] Grünsteidl G. Kopetz H. and Reisinger J. Fault-tolerant membership service in a synchrononous distributed real-time system. *In Proceedings of 20 IFIP International Working Conf. on Dependable Computing for Critical applications (DCCA-1)*, pages (Tucson, AZ). A.Avizienis and J.–C. Laprie, Eds. Springer–Verlag. 411–429, 1991.

[12] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7) :558–565, 1978.

[13] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

# Annexe A

# Demonstration

Dans cet annexe, nous présentons une série de lemmes et de théorèmes ayant pour but de convaincre que le protocole SPA garantit bien l'ordre total de la diffusion qu'il propose. Nous établissons des démonstrations formelles sur les quatres propriétés qui définnissent une diffusion totalement ordonnée, comme elles sont définies dans la section 1.2.2 ; à savoir la `validité`, `l'intégrité`, `l'accord` et `l'ordre total`.

**Lemme A.1 (Co-émetteur)** *Soit $n_s$ un noeud étant silencieux pendant le tour $T_i$ : A la fin de $T_i$, tous les noeuds du système auront reçu l'information $I_s$ relatant le souhait de diffusion qu'avait le noeud $n_s$ au début du tour $T_i$.*

démonstration. La procédure ordonnerCoprivilege() organise les co-privilèges de façon à ce qu'un super-émetteur qui reçoit une information $I_s$ de la part d'un co-émetteur lors d'une ronde $r$ d'un tour $T_i$, possède systématiquement un second privilège plus tard dans le même tour (lignes 88 à 100). Le système ne connaissant pas les pannes de noeud (hypothèse H4), la diffusion de l'information $I_s$ sera assurée par ce super-émetteur avant la fin du tour $T_i$.

**Lemme A.2 (Intégrité de la connaissance distribuée)** *A chaque début de tour, tous les noeuds du système possèdent la même connaissance des variables : souhait.*

démonstration. La variable *souhait* est propagée pendant la diffusion des messages (lignes 64,68 et 72). Toutes les variables *souhait* des noeuds non-silencieux seront donc connues de tous les noeuds du système à la fin du tour. De plus si on généralise le Lemme A.1 à tous les noeuds silencieux, le protocole garantit que tous les souhaits de tous les noeuds seront connus de tous les noeuds à la fin de chaque tour.

**Lemme A.3 (Calcul de l'ordonnancement)** *L'ordonnancement des privilèges et co-privilèges sera calculé sur chaque site uniquement durant les rondes correspondant au début d'un nouveau tour.*

démonstration. Les procédures ordonnerPrivilege() et ordonnerCoprivilege() sont appelées uniquement lorsqu'un nouveau tour commence (lignes 28 à 31).

**Lemme A.4 (Privilege)** *Lors d'une ronde un seul noeud diffusera un message.*

démonstration. A chaque début de tour, chaque noeud calcule le privilège qui sera associé à chacune des $N$ prochaines rondes (ligne 29). Le Lemme A.2 nous assure que la variable *souhait* qui sert a calculer ce privilège (ligne 77) a la même valeur sur tous les noeuds lors de chaque ronde qui commence tour. D'aprés le Lemme A.3 l'ordonnancement ainsi calculé ne sera pas modifié jusqu'au début du prochain tour. De plus les procédures d'ordonnancement ordonnerPrivilege() et ordonnerCoprivilege() sont déterministes (elles n'utilisent pas de variable dont la valeur dépend d'un élément aléatoire ou dont la valeur est relative au noeud local.) (lignes 75 à 100). Au cours de chaque ronde un seul noeud possèdera alors le privilège de diffuser un message sur le réseau et tous les noeuds seront au courant de ce privilège (il n'y aura jamais deux noeuds connaissant un ordonnancement des privilèges différent).

**Lemme A.5 (Latence)** *Si un noeud déclenche la diffusion d'un message $m$ lors d'une ronde $r$, alors tous les noeuds du système délivreront $m$ lors de la ronde $r+1$.*

démonstration. Si un message $m$ a été diffusé lors d'une ronde $r$ à l'aide de la primitive *envoie*(...) par un noeud $n_i$ (ligne 68), il est impossible pour chacun des noeuds de traiter la réception de ce message pendant la ronde $r$. Cette affirmation est due à l'ordonnancement des actions effectuées par notre algorithme (lignes 25 à 33). De plus le Lemme A.4 garantit que seul $m$ transitera sur les canaux de communications depuis $n_i$ vers les autres noeuds. Donc tous les noeuds du système disposeront de leur capacité de réception pour recevoir le message $m$. D'aprés l'hypothèse H1, lors de la ronde $r+1$, tous les noeuds du système auront reçu $m$ et l'auront stocker dans la variable *reçu* (voir ligne 36). Pendant cette même ronde, tous les noeuds invoqueront la méthode remise() et délivreront le message $m$ (ligne 57). Enfin, l'hypothèse H4 excluant les pannes des noeuds, le protocole SPA garantit que strictement tous les noeuds du système auront délivrés $m$ lors de la ronde $r+1$.

**Théorème A.6 (Validité)** *le protocole SPA garantit que si un noeud correct $n_i$ diffuse un messsage $m$, alors $m$ sera délivré par $n_i$ au bout d'un temps fini.*

démonstration. D'aprés le Lemme A.5, si $n_i$ diffuse $m$ lors d'une ronde $r$, alors il le délivrera lors de la ronde $r+1$ (car $n_i$ est inclus dans l'ensemble des destinataires) (ligne 68).

**Théorème A.7 (Intégrité)** *Le protocole garantit que pour chaque message $m$, chaque noeud qui délivre $m$, délivre $m$ au plus une fois et seulement si $m$ a été intialement diffusé par un noeud quelconque.*

démonstration. Les hypothèses faites par un système distribué synchrone assurent que si un message $m$ est reçu par un noeud $n_i$, il a été initialement envoyé par un noeud du système (pas de création de message). De plus la procédure de remise de message, s'assure qu'une fois $m$ délivré, ce message est supprimé localement (voir ligne 58). Aucun doublons de $m$ ne pourra donc apparaître dans le système. La propriété d'intégrité est donc respectée.

**Lemme A.8 (Diffusion fiable et strictement atomique)** *Chaque message $m$ diffusé sur le réseau de communication lors d'une ronde $r$ est délivré de façon fiable et strictement atomique, c'est à dire que tous les noeuds reçoivent $m$ et seulement $m$ lors de la ronde $r+1$*

démonstration. Le Lemme A.4 nous assure qu'un seul noeud aura le privilège de diffuser un message lors d'une ronde $r$. D'aprés le Lemme A.5, si $m$ est diffusé lors de la ronde $r$, alors tous les noeuds délivrent $m$ lors de la ronde $r+1$. De plus, l'hypothèse H4 exclue la panne des noeuds. La fiabilité et l'atomicité de la diffusion sont alors assurées.

**Théorème A.9 (Accord)** *Le protocole SPA garantit que si un noeud $n_i$ délivre un message $m$, alors tous les noeuds corrects du système délivreront $m$.*

démonstration. D'aprés le théorème d'intégrité, si $m$ est délivré par $n_i$, c'est qu'il a été initialement diffusé par un noeud du système. Le Lemme A.8 nous affirme que si $m$ est diffusé par un noeud, alors tous les noeuds du système délivrent $m$ lors de la prochaine ronde.

**Théorème A.10 (Ordre total)** *Le protocole SPA garantit que pour chaque couple de message $m$ et $m'$, si un noeud $n_i$ délivre $m$ sans avoir délivré $m'$, alors aucun noeud ne délivrera $m'$ avant $m$.*

démonstration. le Lemme A.8 assure que la diffusion de chaque message est strictement atomique, ce qui garantit que tous les messages circulant sur le système sont délivrés dans le même ordre.

**Théorème A.11 (Diffusion totalement ordonnée)** *Le protocole SPA fournit une diffusion totalement ordonnée.*

démonstration. le protocole SPA respecte les propriétées suivantes : Validité, Intégrité, Accord et Ordre total, il implante donc bien une diffusion totalement ordonnée.