

Protocole multipoint fiable et ordonné pour applications coopératives asynchrones

Amine SIAFA

Equipe des Systèmes Communicants
Université de Savoie, 73376 Le Bourget-du-Lac, France
Tel: 04 79 75 86 85 - Fax: 04 79 75 86 90
e-mail: siafa@univ-savoie.fr

Thèse réalisée sous la direction de M. Michel RIVEILL
Professeur à l'Université de Savoie

Table des matières

1	Introduction	3
2	Communication multipoints: synthèse et classification	7
2.1	Introduction	7
2.2	Concepts et définitions	8
2.2.1	Introduction	8
2.2.2	Notion de groupe de processus distribué	8
2.2.3	Avantages de la communication de groupe	9
2.2.4	Taxinomie des groupes de processus	10
2.2.5	Notions de vue de groupe et de synchronisme virtuel	11
2.2.6	Communication multipoint	13
2.2.7	Multipoint synchrone et multipoint asynchrone	15
2.2.8	Fiabilité dans le multipoint	15
2.2.9	Ordonnancement des messages	16
2.3	Synthèse des protocoles multipoint asynchrones	19
2.3.1	Protocoles multipoint dans les systèmes répartis	19
2.3.2	Protocoles multipoint de niveau transport OSI	24
2.4	Proposition d'un protocole multipoint flexible	49
2.4.1	Qualité de service	49
3	Conception de FMP : Protocole Multipoint Flexible	53

3.1	Introduction	53
3.2	Modèle	53
3.3	Architecture du système	54
3.4	Description des services fournis par FMP	54
3.4.1	Définitions relatives aux services fournis	54
3.4.2	Hypothèses de fonctionnement de FMP	55
3.4.3	Fonctionnalités de FMP	58
3.4.4	Description des unités du protocole FMP	62
3.5	Service de diffusion fiable (RDS)	66
3.5.1	Mécanisme d'acquittement	66
3.5.2	Gestion des pertes dans FMP	72
3.5.3	Mécanisme de requête	74
3.5.4	Mécanisme de retransmission	81
3.5.5	Terminaison de transfert et consensus unanime	85
3.5.6	Reprise après défaillances de sites ou partitionnements	88
3.6	Service de livraison totalement ordonnée (TODS)	88
3.6.1	Principe de TODS	88
3.6.2	Fonctionnement normal du service	89
3.6.3	Fonctionnement lors de défaillances	90
3.7	Service de livraison causale (CDS)	92
3.7.1	Motivations et définitions	92
3.7.2	Fonctionnement normal de CDS	94
3.7.3	Fonctionnement lors de défaillances ou de changements de vue	96
3.8	Conclusion	102

4.1	Introduction	103
4.2	Structures de données et algorithmes	104
4.2.1	Service RDS	104
4.2.2	Service TODS	108
4.2.3	Service CDS	111
4.3	Représentation du protocole sous forme d'automate	113
4.3.1	Fonctionnement normal du protocole	113
4.3.2	Fonctionnement lors de changements dans le groupe	119
4.4	Regard sur le langage Estelle	120
4.4.1	Spécification Estelle	121
4.5	Spécification formelle de FMP	123
4.5.1	Architecture du système	123
4.5.2	Comportements des modules de la spécification	125
4.6	Questions d'implémentation	130
4.7	Conclusion	133
5	Evaluation et perspectives	135
5.1	Evaluation	135
5.1.1	Protocoles existants	135
5.1.2	Propriétés intrinsèques de FMP	136
5.1.3	Service CDS	139
5.1.4	Limitations de l'étude réalisée	139
5.2	Perspectives	140

Chapitre 1

Introduction

Les performances des réseaux dans la dernière décennie aussi bien en fiabilité qu'en bande passante, ont conduit à une décentralisation des traitements vers des systèmes informatiques distribués, rendue indispensable, par les limites des systèmes centralisés en matière de mémoires, de stockage de l'information et de disponibilité des systèmes face à un grand nombre d'utilisateurs. Cette évolution a été également motivée par la nécessité d'adapter la structure d'un système à celle de l'application qu'il traite; de nombreuses applications sont intrinsèquement réparties, soit géographiquement, soit fonctionnellement. Le développement d'applications coopératives est venu par la suite répondre au besoin de reproduire, grâce aux systèmes informatiques répartis, les conditions de travail de groupe humain.

L'évolution des systèmes vers le modèle réparti a introduit de nouveaux problèmes de conception inhérents aux supports de communication, tels que la transmission fiable des données, l'ordonnancement des messages à leur livraison et la tolérance du système face à la défaillance d'une partie de ses composants.

Le développement d'applications coopératives repose sur un mécanisme d'échange de données à travers une collection de sites répartis. Les fonctionnalités offertes par ce type d'applications visent à exploiter les avantages des réseaux de communication sous-jacents, tout en évitant les inconvénients de ces derniers. A cet effet, ces applications nécessitent des services en communications spécifiques et par conséquent des protocoles de communications adaptés pour implémenter ces services. Le service de communication multipoint, appelée aussi communication de groupe ou "*multicast*", est un des services essentiels que le support de communication doit fournir. Il répond au besoin de structurer les applications coopératives et à réaliser la coopération entre les entités de celles ci. En effet, le service multipoint permet, non seulement, d'organiser les échanges de messages entre processus d'une même application, mais aussi, par un mécanisme de gestion de groupe

intégré, il prend en charge l'évolution structurelle et fonctionnelle de celle ci. Aussi, pour les données réparties ou duppliquées, il premet d'assurer un accès efficace et cohérent sans pour autant surcharger le réseau. Dans le cas des applications multi-média, il représente l'unique façon de garantir la fiabilité dans la transmission rapide des données entre les participants.

Le manque de primitives assurant une diffusion totale ("broadcast") ou restreinte ("multicast") fiable et ordonnée a introduit une multitude de solutions complexes et parfois superflues. En effet, bien qu'elles pallient aux problèmes intrinsèques aux supports de communication, ces solutions ne sont pas toujours réutilisables car les mécanismes de communication qu'elles implantent sont, le plus souvent, étroitement liées aux applications auxquelles elles sont dédiées. Cela a pour conséquence d'affecter l'interopérabilité entre applications ainsi que leur évolution. Pour cette raison, des mécanismes de communication communs et indépendants doivent permettre aux concepteurs d'application coopératives de focaliser uniquement sur les fonctionnalités que celles-ci doivent offrir.

Mon travail de thèse s'inscrit dans le cadre de ces mécanismes qui visent à fournir un environnement permettant la construction, l'exécution et l'évolution d'applications coopératives. Le réseau de communication supportant les interactions d'une application coopérative peut être aussi bien local qu'étendu. Dans le premier cas, l'application est dite *locale*, alors que dans le second cas, l'application est dite *étendue*. La mise en oeuvre du dernier type d'application est plus subtile que le premier. En effet, les problèmes de fiabilité, d'ordonnancement et d'extensibilité sont d'autant plus complexes que la taille des réseaux d'interconnexion augmente. Dans ce cadre, notre intérêt a porté sur les protocoles de communication qui supportent la communication de groupe au sein d'applications asynchrones étendues, dans lesquelles chaque entité peut être à la fois émetrice et réceptrice et où le mode de communication repose sur les échanges de messages asynchrones et discontinus.

Notre apport consiste en la spécification et l'implantation d'un protocole multipoint fiable et ordonné. A la différence des communications point-à-point (ou *unicast*) où les conditions requises pour le transport des données sont assez générales, la diversité des services fournis par les applications coopératives fait qu'elles ont des besoins variés en services de communication multipoint. De ce fait, pour que le protocole soit utilisé par un aussi large ensemble que possible d'applications, il est nécessaire de le munir d'une *flexibilité* qui suffit aux différents services de communication requis sans pénaliser les applications dont les exigences sont modestes.

Dans le premier chapitre, nous définissons le concept de groupe de processus distribué et introduisons la problématique liée à la communication de groupe. Nous faisons par la

suite une taxinomie des protocoles multipoint existants en classifiant ces derniers selon les propriétés de fiabilité et d'ordre sur lesquelles a porté notre étude. La dernière partie du chapitre évalue les protocoles étudiés et précise les arguments qui ont motivé notre proposition.

Dans le deuxième chapitre, nous proposons un modèle de service multipoint fiable et ordonné. Nous décrivons les fonctionnalités des différents services fournis dans le cadre du protocole proposé. Ces services réalisent d'une part les propriétés de flexibilité et d'extensibilité, et d'autre part, celles de fiabilité et d'ordre, qui permettent de répondre aux besoins très variés des application de groupe.

Le troisième chapitre est consacré à la spécification formelle du protocole proposé. Nous décrivons les structures de données du protocole et les actions qui y sont associées. Nous représentons ensuite le protocole sous forme de machines à états finis dans le but d'introduire la description formelle du comportement du protocole proposé.

Le quatrième chapitre conclut ce document par une évaluation critique du travail effectué, ponctuée par l'évocation de quelques perspectives sur lesquelles ce travail peut être étendu.

Chapitre 2

Communication multipoints: synthèse et classification

2.1 Introduction

Les protocoles de diffusion ont fait l'objet d'un nombre important de travaux de recherche. L'abondance de la littérature qui leur a été consacrée s'explique par des raisons aussi diverses que les objectifs multiples de conception, dans les caractéristiques propres aux architectures cibles (réseau point-à-point, réseau de diffusion, réseau local, réseau étendu, etc.), dans les différents besoins des applications cibles en services de communication, dans les caractéristiques intrinsèques des applications (groupes dynamiques ou statiques, groupes fermés ou ouverts, connaissance des membres requise ou non, etc.) et dans la différence entre les sémantiques de fiabilité et d'ordre considérées.

Dans ce chapitre, nous étudions les protocoles de diffusion fiable, atomique et causale les plus représentatifs de l'état de l'art. Dans la première section nous introduisons la problématique liée à la communication multi-destinataires fiable et ordonnée. La deuxième section est consacrée à l'étude des protocoles multipoint, réalisant les propriétés de fiabilité et d'ordre, les plus représentatifs de l'état de l'art. Les protocoles étudiés sont décrits selon les critères de fiabilité, décrits en introduction de la section, et d'ordre décrit dans la première section. La dernière section évalue les protocoles étudiés et précise les motivations qui nous ont amené à proposer un protocole multipoint dont la description fera l'objet des chapitres suivants.

2.2 Concepts et définitions

2.2.1 Introduction

L'introduction de la communication multidestinataires dans les systèmes distribués a mis en évidence de nouveaux concepts liés au modèle de groupe. Nous présentons la notion de groupe de processus distribués et étudions les différents concepts qui caractérisent la communication de groupe.

2.2.2 Notion de groupe de processus distribué

Un groupe de processus distribués est un ensemble formé d'un ou plusieurs processus s'exécutant sur des noeuds éventuellement différents et coopérant pour assurer un service donné. Cet ensemble est vu, et donc manipulé, comme une entité logique et est identifié par un identificateur de groupe unique.

L'idée de structurer une application distribuée en groupes de processus coopérant à la réalisation d'un traitement donné n'est pas nouvelle. En effet, le concept de groupe de processus a été introduit dans plusieurs systèmes dans le milieu des années 80, on citera le système V [CZ85], ISIS [Bir86], Amoeba [TR85], [AC93] et Circus [Coo90]. L'introduction de la notion de groupes de processus (ou d'objets) avait pour objectifs d'améliorer la fiabilité et la disponibilité ainsi que de réduire les temps d'accès aux divers services distribués offerts. Dans la suite, nous énumérons les principales propriétés de la notion de groupe relativement aux applications qui l'utilisent :

1. *faciliter l'accès à une ressource distribuée :*

Pour localiser une ressource distribuée, un processus client peut procéder selon deux alternatives : diffuser une seule requête à tous les sites ou bien émettre plusieurs requêtes successives aux sites pouvant être en possession de la ressource. Bien qu'efficace du point de vue temps de réponse, la première alternative induit une charge superflue au niveau des sites non détenteurs de la ressource afin de traiter la requête reçue. Dans la seconde alternative, le temps de réponse est plus élevé. De plus, il est nécessaire que le client maintienne une liste des sites détenteurs de chacun des services invoqués. Dans le cas où la migration des ressources est possible, la deuxième solution doit intégrer un mécanisme système supplémentaire permettant la localisation des serveurs mobiles.

Un compromis entre les deux alternatives est d'identifier les ressources réparties par des adresses multidestinataires indépendantes de la localisation. Ceci se traduit au

niveau client par une transparence ainsi que de meilleures performances lors des opérations d'accès aux ressources. Cette technique a été mise en oeuvre dans plusieurs implantations telles que [?] et [Hug91].

2. *Améliorer la disponibilité et la tolérance aux défaillances :*

La disponibilité d'un système distribué est définie par son aptitude à fonctionner de façon normale lorsqu'un ou plusieurs de ses serveurs deviennent inopérants. Identifier un groupe de serveurs redondants (i.e., équivalents mais physiquement indépendants) par une adresse multidestinaires est une solution efficace pour augmenter la disponibilité du système. Ainsi la défaillance d'un sous-ensemble de serveurs ne peut avoir qu'une faible incidence (en performance) sur les clients [Cri91].

3. *Simplifier l'écriture d'applications nécessitant la mise à jour de données réparties :*

Certaines applications distribuées nécessitent une mise à jour fréquente des données dupliquées. L'utilisation de la communication multidestinaires permet une mise en oeuvre plus simple grâce à une désignation transparente et collective. Les implantations qui peuvent tirer parti de la communication de groupe peuvent être aussi diverses que les applications manipulant des bases de données réparties [Pal88], les programmes de synchronisation d'horloges distribuées ([AB85]), les programmes assurant la mise à jour des informations relatives aux topologies dans les algorithmes de routage [Ram90], etc.

4. *Contrôler l'exécution de programmes distribués :*

Considérons un processus P qui contrôle l'exécution de n processus distribués P_1, P_2, \dots, P_n . Il est naturellement plus efficace de permettre à P d'émettre un message contenant une directive de contrôle à l'ensemble des processus plutôt que d'émettre n messages identiques.

2.2.3 Avantages de la communication de groupe

Bien qu'il soit possible d'implanter les opérations de communication de groupe en utilisant les primitives classiques de la communication de niveau processus¹ (ou bien 'point-à-point', en opposition à multipoint) (on parlera alors de simulation), quatre raisons majeures rendent ce choix inopportuns [Che88]:

- L'utilisation de primitives de niveau processus ne peut se faire que si le processus invoquant connaît l'identité individuelle de chacun des processus concernés par la primitive.

1. On parlera alors de simulation et non pas de communication de groupe réelle.

- L'utilisation de plusieurs communications point-à-point pour donner un effet équivalent à une seule communication de groupe est clairement moins efficace si le support de communication permet une émission multiple (réduction, à la fois, de la charge de l'émetteur et celle du réseau).
- L'utilisation d'opération de groupe offre une meilleur concurrence que celle d'opération de niveau processus. En effet, lorsqu'un processus diffuse une information à tous les processus participants, celle ci sera reçue dans des délais nettement meilleurs dans le cas d'une communication multipoint que si la diffusion était séquentielle. Cela permet, par conséquent, d'améliorer le degré de concurrence de l'application.
- La communication de groupe permet aux programmeurs une conception plus simple de leur applications grâce à l'abstraction qu'elle offre sur les groupe de processus. Celle ci a pour effet de leur soustraire tous les problèmes liés aux performances, à désignation, à la disponibilité, etc. De plus, la communication de groupe rend transparentes les interactions internes entre les membres d'un groupe [WZZ91].

2.2.4 Taxinomie des groupes de processus

Divers critères ont été utilisés dans la littérature pour la classification des groupes de processus. Un critère de classification communément utilisé est l'étendue de la communication impliquant les processus appartenant à un groupe. Ainsi, on peut distinguer deux types de groupes orthogonaux [LCN90] [DTH92] :

- **Groupes fermés** : où les membres d'un groupe ne peuvent recevoir de messages que des autres processus membres. C'est, par exemple, le cas dans certains systèmes de téléconférences [Pal88].
- **Groupes ouverts** : où les membres peuvent recevoir des messages aussi bien des autres membres que de processus n'appartenant pas au groupe. Dans [BSS91], on distingue trois différents types de groupes ouverts :
 1. *Groupes client-serveur* : un groupe de processus serveurs équivalents est accessible par un nombre important de clients. L'interaction entre clients et serveurs se fait selon le mode requête/réponse. Le client peut sélectionner, à priori, un serveur particulier ou bien diffuser sa requête à tous les membres du groupe. Dans le second cas, chaque serveur émet une réponse au client ainsi qu'à tous les autres serveurs, leur évitant, ainsi, de traiter une requête déjà satisfaite.

2. *Groupes de diffusion*: c'est un cas particulier du type précédent dans lequel le serveur diffuse ses réponses à tous les serveurs du groupe ainsi qu'à tous les clients désignés par la même adresse de groupe. Cela survient lorsque la réponse à la requête peut être utile aux autres clients.
3. *Groupes hiérarchiques*: où un groupe est formé de sous-groupes structurés sous forme d'arborescence. Lorsqu'un client invoque le groupe racine, ce dernier sélectionne l'un des sous-groupes fils et le charge de la traiter. Dès lors le client interagit directement avec le sous-groupe sélectionné, même si d'autres sous-groupes de l'arborescence participent à l'exécution de la requête.

D'autres critères se rapportant aux supports d'accueil, à la pérennité et aux caractères privé ou public, restreint ou non des groupes, permettent également de distinguer les types de groupes suivants :

- **Groupes locaux et groupes non locaux** : les premiers cités limitent le domaine d'exécution de ses processus membres à un seul site. Dans les autres, les membres peuvent être dispersés sur plusieurs sites d'exécution. Si la migration de processus est possible, un groupe local perd cette propriété dès que l'un de ses membres quitte le site d'accueil du groupe [Che88].
- **Groupes permanents et groupes momentanés** : un groupe momentané [Ngo91] est un groupe qui peut cesser d'exister avant la fin de l'application à laquelle il appartient. Un groupe permanent ne peut être supprimé qu'une fois l'exécution de l'application terminée.
- **Groupes publics et groupes privés** : dans le cas d'un groupe public, tout processus peut se joindre au groupe sans aucune autorisation préalable [Sia96]. Par contre, les groupes privés délèguent à un processus particulier la gestion des demandes d'adhésion selon des protocoles établis.
- **Groupes restreints et groupes non restreints** : un processus quelconque peut se joindre à un groupe non restreint, tandis que seuls les processus ayant les privilèges requis peuvent devenir membres d'un groupe restreint.

2.2.5 Notions de vue de groupe et de synchronisme virtuel

La communication de groupe ne peut être envisagée de façon optimale sans l'utilisation d'informations précises quant aux identités des membres formant la configuration courante du groupe. Pour cela, il est nécessaire de maintenir une cohérence entre les différentes

perceptions que les membres ont de cette configuration. La notion de vue d'un groupe de processus a été défini par Birman [BJ87b] comme un état global reflétant la composition et les propriétés globales de ce groupe à un instant logique t . Cela peut concerner l'ensemble des processus corrects (cf. 2.2.8 autorisés à participer au protocole de diffusion).

Dans les systèmes distribués, il n'est pas possible, en l'absence de mémoire commune, de synchroniser les différents processus autrement que par des envois de messages. Cependant, les délais de communication arbitraires et les risques de défaillance du sous-système de communication peuvent être à l'origine de perte ou de déséquencement de messages. De ce fait, un message contenant les propriétés globales d'un groupe (telle qu'une défaillance d'un membre ou l'adjonction d'un nouveau membre) peut ne pas être reçu au même moment, ni dans le même ordre (par rapport à d'autres messages diffusés) par tous ses destinataires, comme il peut ne pas être reçu par certains. Cet asynchronisme rend donc difficile le maintien d'une cohérence parmi tous les membres d'un groupe quant à son état global. L'exemple suivant illustre parfaitement l'importance cette difficulté [BJ87b] :

Soit un serveur implémenté par un groupe de processus. Supposons que la règle de gestion des requêtes est que le membre de plus petit identificateur (dans la vue en cours) répond à la requête. Pour que ce serveur fonctionne correctement, il est impératif que tous les membres aient, à tout moment, la même information sur les membres opérationnels du groupe. En effet, si certains membres "ignorent" que le membre de plus petite identité est défaillant, certaines requêtes ne seront jamais traitées. De même si un membre "croit" que tous les membres de plus petites identités sont défaillants alors qu'il existe au moins un parmi ces membres qui soit opérationnel, une même requête serait traitée par plusieurs processus. Par conséquent, il est nécessaire que, lors du changement d'état global du groupe, tous les membres concordent à traiter la requête en cours soit *avant*, soit *après* que le changement d'état ne devienne effectif. Ainsi, lorsque de nouveaux membres adhèrent à un groupe alors qu'un message est en voie d'acheminement, ce dernier sera délivré à tous les nouveaux membres ou bien à aucun membre.

Cristian [Cri91] cite deux conditions qu'un mécanisme assurant la cohérence globale dans un groupe de processus doit satisfaire :

- Tous les processus "voient" la même séquence de défaillances et de reprises (ou d'adjonctions) de processus de telle façon qu'à tout moment, tous les processus maintiennent une même liste des membres opérationnels avec lesquels il est possible de communiquer.
- Entre deux changements successifs dans la composition d'un groupe, tous les membres reçoivent la même séquence ordonnée de messages.

Lorsque le système de communication assure les deux conditions précitées pour tous les groupes, on dira que ces groupes sont *virtuellement synchrones*. Dans un environnement virtuellement synchrone, la séquence d'événements (diffusions, défaillances, reprises, etc.) observée par chaque processus est équivalente à celle qui se serait produite si les processus étaient dans un environnement où les événements se produisent de façon synchrone [BJ87a].

La mise en oeuvre des protocoles assurant le synchronisme virtuel soulève d'énormes difficultés et nécessite l'utilisation de protocoles de communication assurant certaines propriétés d'ordre sur la livraison des messages. Une solution à ce problème pourrait être apportée par les protocoles de diffusion fiables et totalement ordonnés (2.2.9). Néanmoins, à cause de son coût élevé, cette solution ne peut être retenue pour tous les types d'applications (certaines ne requièrent aucune contrainte sur l'ordonnancement des messages).

2.2.6 Communication multipoint

2.2.6.1 Définition du multipoint

La communication multipoint (ou *multicast*) est une technique utilisée pour transmettre des copies d'un message à un sous-ensemble parmi toutes les destinations possibles². Une façon d'implémenter le multicast est réalisée lorsqu'on envoie un courrier électronique à une liste de destinataires ou que l'on poste une intervention dans un groupe de "news". Mais cela devient rapidement lourd à gérer si la liste de destinataires varie souvent. De plus, cette approche conduit à faire circuler de multiples exemplaires des mêmes données sur un même lien, consommant ainsi de la bande passante. Des extensions ont donc été apportées au niveau de la couche IP pour pallier aux inconvénients cités plus haut. La diffusion de groupe au niveau IP correspond à la diffusion de groupe au niveau physique (Ethernet). Elle permet la diffusion d'un datagramme IP vers un ensemble de machines qui forment un même groupe de diffusion. Les membres du groupe peuvent être répartis sur des réseaux physiques distincts interconnectés par des passerelles. Le Multicast IP utilise la même signification de remise non fiable (best-effort) que les autres datagrammes IP, ce qui signifie que les datagrammes peuvent être perdus, dupliqués, retardés ou remis dans le désordre.

². Lorsque le sous-ensemble est l'ensemble des destinations possibles, la technique est appelée broadcasting.

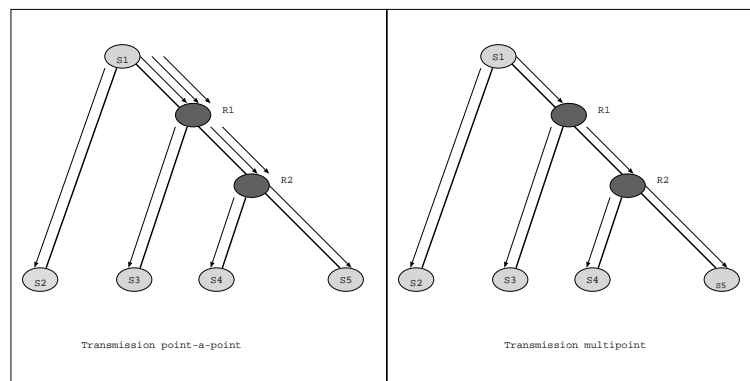


FIG. 2.1 - Exemple de transmission multipoint

2.2.6.1.1 Multicast au niveau IP :

Le multicast IP est aussi bien utilisé pour un réseau local que pour une interconnexion de réseaux locaux. Dans ce dernier cas, des *routeurs de multicast* acheminent les datagrammes de diffusion de groupe. Un site émetteur transmet les datagrammes multicast au routeur de groupe (sans toutefois le connaître explicitement) qui les achemine, si nécessaire, vers les autres réseaux constituant le groupe destinataire. Chaque réseau local se charge de remettre les datagrammes aux sites destinataires locaux. La norme IP décrit les protocoles utilisés par les passerelles pour déterminer l'appartenance à un groupe de diffusion, au niveau d'un réseau local.

L'appartenance à un groupe de diffusion est dynamique. Une machine peut appartenir à un nombre de groupes multicast arbitraire. Son appartenance à un groupe de diffusion détermine le fait qu'elle reçoive l'information destinée à ce groupe. Une machine peut toutefois envoyer des informations à un groupe multicast sans y appartenir.

La diffusion IP définit, également, l'adressage de groupe au niveau IP. Aux trois classes d'adressage IP traditionnelles (A, B et C) s'ajoute la classe D dite multipoint. Toute adresse IP commençant par "1110" appartient à cette classe et représente une adresse de groupe. Les 28 bits restants identifient les groupes de diffusion³. Chaque groupe de diffusion possède une *adresse multicast* unique qui est attribuée pour une utilisation temporaire. Une adresse de groupe est, ainsi, créée à la demande puis supprimée lorsque le groupe est vide.

3. Les adresses multicast sont prises de l'intervalle d'adresses compris entre 224.0.0.0 et 239.255.255.255

2.2.7 Multipoint synchrone et multipoint asynchrone

Deux modèles de systèmes répartis peuvent être considérés : soit un modèle à délai non borné, soit un modèle à délai borné. Le premier cas est essentiellement caractérisé par l'incapacité de donner une borne à priori sur les temps de transmission de messages et sur les délais de traitement. Ce type de système est désigné par le terme de système *asynchrone*. Dans le cas où une borne sur les délais de communication et de traitement, est à priori connue, on parle de système *synchrone*. En outre, ce type de système définit des contraintes sur le nombre de défaillances de composants, sur la redondance des liens et sur le flux de contrôle. Il est admis que la communication synchrone est plus simple à implanter que la communication asynchrone. L'aspect synchrone permet, en effet, de réduire le nombre de buffers contenant les messages non encore acquittés. La communication asynchrone offre, quant à elle, un plus grand degré de parallélisme, une bonne tolérance aux défaillances et une meilleure flexibilité.

2.2.8 Fiabilité dans le multipoint

Birman et al. [BSD88] définit la fiabilité dans un protocole de diffusion comme une primitive de communication qui assure les propriétés suivantes :

- Consensus unanime : tous les récepteurs corrects⁴ reçoivent la même valeur ou convergent vers une même valeur.
- Validité : si l'émetteur ne défaille pas durant la diffusion, soit tous les processus récepteurs délivrent la même valeur émise par l'émetteur, soit aucun d'entre eux ne le délivre.
- Terminaison : il existe un délai fini pour la réception, par tous les destinataires correctes, d'un message diffusé.

Autrement dit, la diffusion d'un message est dite fiable si tous les destinataires non défectueux de ce dernier le reçoivent (ou bien, en cas de panne non récupérable de l'émetteur pendant la diffusion, aucun ne doit le recevoir). Un protocole multipoint fiable doit résister aux pannes du système de communication et des sites [Kra91].

4. Un composant du système est dit correct, si en réponse à l'occurrence il se comporte de façon conforme à sa spécification [CAS85].

2.2.9 Ordonnement des messages

La propriété de fiabilité ne spécifie rien sur l'ordre de réception des messages par les destinataires, ni sur la relation entre l'ordre d'émission et l'ordre de réception. Pour une grande classe d'applications réparties (par exemple, celles nécessitant la mise à jour cohérente de données dupliquées), le protocole multipoint utilisé doit assurer une livraison ordonnée de messages à tous les destinataires. La sémantique d'ordre peut avoir différentes interprétations, Krakowiak [Kra91], spécifie deux propriétés dans une diffusion ordonnée :

1. La propriété d'*uniformité* impose que l'ordre de délivrance⁵ des messages soit identique pour tous les récepteurs.
2. La relation entre l'ordre d'émission et l'ordre de réception peut être spécifiée de trois manières :
 - a) aucune relation spécifiée,
 - b) l'ordre de réception est identique à l'ordre total (ou global) d'émission (si un ordre total sur les événements peut être défini).
 - c) l'ordre de réception est identique à l'ordre causal d'émission (si un ordre causal sur les événements peut être défini).

D'après [Kra91], un protocole de diffusion fiable qui possède les propriétés 1 et 2 (b ou c) est dit *atomique*⁶.

La définition de la diffusion atomique adoptée dans la suite de notre étude est celle donnée par les auteurs de [CAS85] [MA91] qui définissent un protocole de diffusion atomique comme une primitive vérifiant les trois propriétés suivantes :

1. unanimité: tout message émis par un processus correct est, soit délivré à tous ses destinataires, soit à aucun d'eux,
2. ordre total,
3. terminaison : un message diffusé par un émetteur correct est délivré à tous ces destinataires corrects en un temps fini (mais non préalablement connu).

5. Notons la différence existant entre la livraison de message, qui correspond à sa remise au site destinataire, et la délivrance de celui-ci qui correspond à sa remise par le site destinataire à l'application. Souvent ces deux termes sont utilisés indifféremment l'un pour l'autre.

6. Certains auteurs appellent diffusion atomique ce qui sera désigné ici par diffusion fiable.

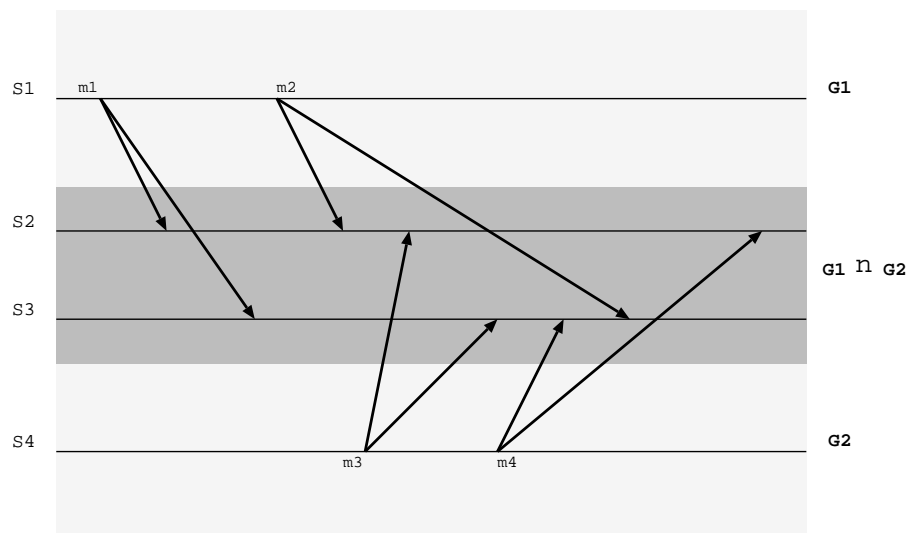


FIG. 2.2 - *Respect de l'ordre total pour des groupes multiples*

Relativement aux types de livraison totalement ordonnée, Garcia-Molina et al [GS91] définissent les trois types suivants :

1. *Ordre total avec source unique* : si un processus P émet vers un groupe G deux messages $m1$ et $m2$, tous les membres de G les reçoivent dans le même ordre. Lorsque l'ordre de livraison correspond à l'ordre d'émission, on dira que la livraison est conforme à l'ordre local [FT92]. Cet ordre peut s'obtenir par simple numérotation séquentielle du message par l'émetteur.
2. *Ordre total avec sources multiples* : si deux messages sont diffusés à un même groupe, tous ses membres les reçoivent dans un même ordre, même s'ils proviennent de deux processus différents.
3. *Ordre total avec groupes multiples* : si deux processus $P1$ et $P2$ diffusent deux messages $m1$ et $m2$ respectivement à deux groupes $G1$ et $G2$, alors $m1$ et $m2$ sont délivrés dans le même ordre à tous les membres communs à $G1$ et $G2$.

L'ordre total est un ordre arbitraire mais identique pour tout le groupe. Chaque membre du groupe délivre les messages en respectant cet ordre. De nombreuses applications nécessitent une livraison dont l'ordre est identique au niveau de chaque récepteur. C'est le cas, par exemple, d'une base de donnée répartie dont les données sont dupliquées et distribuées sur des serveurs différents formant un groupe. Dans un souci de préserver la cohérence de la base de données, il est essentiel que toutes les modifications sur une donnée soient faites dans un ordre identique sur tous les serveurs.

L'ordre causal, quant à lui, est communément défini comme l'extension, aux systèmes répartis, de la relation de causalité (ou relation FIFO: *irst In First Out*) définie par Lamport [Lam78] dans le cas d'événements se produisant sur un seul site. Il permet, en effet, d'établir un ordre partiel sur les événement se produisant sur plusieurs sites. En effet, si un message m_1 d'un émetteur provoque l'émission d'un message m_2 par un des récepteurs, le message m_1 sera délivré avant le message m_2 chez tous les récepteurs des deux messages. Aussi, lorsqu'un processus diffuse deux messages, m_1 et m_2 , l'ordre de livraison par les récepteurs est celui préétabli par émetteur. Dans les systèmes de téléconférence ou de listes de diffusions, le non respect de l'ordre causal peut conduire à des situations où une réponse est affichée avant même que la question qui en était à l'origine ne soit reçue.

2.2.9.1 Tolérance aux défaillances

La défaillance d'un membre dans une connexion point-à-point ne peut conduire qu'à la rupture de la communication (la connexion n'a plus lieu d'exister si l'une des deux extrémités devient hors service). En multipoint, la défaillance d'un membre ne remet pas forcément en question la communication de groupe. Plusieurs alternatives peuvent pallier à cette défaillance, en fonction de la sémantique adoptée par l'application. Cette sémantique peut fixer l'importance du membre défaillant, ou encore les conditions à remplir pour poursuivre la communication.

Dans le contexte multipoint, la tolérance aux défaillances peut avoir deux significations :

- soit le comportement que le système aurait lors de la défaillance d'un de ses composants est, à priori, bien défini,
- soit le système masque les défaillances à ses utilisateurs et continue à les servir de façon “normale” en dépit de l'occurrence de pannes (la tolérance aux défaillances est alors équivalente à la fiabilité du système) [Cri91].

Un protocole multipoint peut avoir plusieurs degrés de tolérance aux défaillances. Un protocole qui masque la défaillance de k sites est dit k -tolérant aux défaillances (ou $k \Leftrightarrow resilient$). Il peut être 1-tolérant aux défaillances, comme c'est le cas du protocole VMTP implémenté dans le système V [CZ85] [Che88], ou globalement-tolérant, comme c'est le cas dans le système ISIS [BJ87b] [BSS91].

2.3 Synthèse des protocoles multipoint asynchrones

Dans cette section, nous présentons un ensemble de protocoles de communication destinés aux architectures asynchrones. Nous commençons notre survol par la présentation de quelques systèmes distribués qui implantent la communication multipoint. Nous nous intéresserons ensuite aux protocoles multipoint de niveau transport du modèle OSI [?]. Nous décrivons en premier les mécanismes qui réalisent la fiabilité au niveau transport, sanctionnés par des exemples de protocoles fiables représentatifs. Nous étudions ensuite les protocoles qui réalisent les propriétés d'ordre causal et total. Nous donnons, en premier lieu, des exemples de protocoles de diffusion causale. En second lieu, nous donnons des exemples représentatifs de protocoles de diffusion atomique selon le procédé de contrôle utilisé (centralisé ou décentralisé).

2.3.1 Protocoles multipoint dans les systèmes répartis

La communication multipoint a été implantée dans plusieurs systèmes d'exploitation distribués. Certains systèmes se sont limités à implanter une couche de communication multipoint au dessus du noyau de communication du système UNIX. D'autres systèmes intègrent leur propre noyau de communication et proposent des primitives offrant plus de fonctionnalités et de flexibilité.

2.3.1.1 ISIS

ISIS [Bir86] [BJ87b] [BSS91] est un environnement de programmation distribué offrant une variété d'outils pour la construction de programmes distribués. Le système a été l'un des premiers protocoles à intégrer la notion de groupe virtuellement synchrone (2.2.5). Une version plus récente du système ISIS permet d'éviter que le fonctionnement du protocole n'interfère avec les traitements nécessaires à la diffusion des données de l'application; ce qui permet d'éviter l'interruption des flux de messages lorsqu'un changement survient dans la composition du groupe de processus [RB91].

Le service de communication du système ISIS offre essentiellement trois primitives de communication de groupe: *GBCAST()*, *ABCAST()* et *CBCAST()*.

La primitive *GBCAST()* (Group BroadCAST) est utilisée pour notifier aux membres opérationnels d'un groupe la défaillance, la reprise, le départ d'un membre, l'adhésion d'un nouveau membre, ou tout autre changement dans l'état global du groupe (tel que la migration d'un processus vers un autre site). Chaque membre maintient localement une

copie de la vue globale de son groupe qu'il met à jour à chaque réception d'un message *GBCAST*. Les messages *GBCAST* sont délivrés à tous les processus dans le même ordre relativement à tous les autres types de messages.

La primitive *ABCAST*, (Atomic BroadCAST) est destinée aux applications où tous les destinataires d'un message doivent le recevoir dans un même ordre (inconnu au moment de l'émission). La flexibilité de *ABCAST* est assurée par une chaîne de caractère (*label*) passée en argument et qui permet de n'imposer la contrainte d'atomicité que sur les messages ayant un même *label*. Ainsi, deux messages ayant un label identique sont délivrés dans le même ordre à tous leurs destinataires communs. Lorsque *label* a la valeur "*", le message sera ordonné relativement à tous les autres messages *ABCAST*.

La primitive *CBCAST* (Causal BroadCAST) est utilisée pour imposer, à la livraison, un ordre préétabli qui est l'ordre causal (2.2.9). Comme pour *ABCAST*, un argument (*clabels*) est utilisé pour établir un ordre partiel entre les *CBCAST*. En effet, bien qu'il soit possible d'ordonner tous messages causalement liés, ceci peut ne pas être nécessaire dans certains cas où les messages concernent des mises à jour sur des variables sémantiquement indépendantes.

CBCAST implante trois mécanismes différents de mise à jour de données dupliquées : atomique, où l'actualisation ne devient effective que lorsque l'émetteur reçoit tous les acquittements attendus, concurrent, où un coordonnateur effectue la mise à jour localement et en informe ses processus subordonnés en diffusant un *CBCAST* asynchrone, et retardé, où les verrous de mise à jour sont obtenus localement par le coordonnateur qui n'émet une transition qu'une fois celle-ci complètement construite et la possibilité d'interblocage écartée (la transaction est émise sous forme d'une série de *CBCAST* contenant les différentes actions de celle-ci).

ISIS permet les deux modes de communication dans l'utilisation de ces primitives : synchrone (où l'émetteur s'interrompt jusqu'à la réception de tous les messages réponse) et asynchrone (où l'émetteur ne s'interrompt pas du tout ou continue son exécution dès la réception d'une première réponse).

2.3.1.2 Horus

Le projet Horus [RBM96] a été initié dans un effort de reconception du système ISIS. Il vise à fournir un système de communication qui tient compte des exigences d'une multitude d'applications distribuées. Horus intègre un modèle de communication de groupe qui repose sur un ensemble de primitives de communication multipoint : fiable, non fiable, FIFO, causalement ordonnée, et totalement ordonnée. Il est "fortement" décomposé en

couches et reconfigurable à un haut degré de sorte que les application ne doivent tenir compte que des services fournis. Cet architecture permet à des groupes ayant des besoins différents de coexister dans le même système.

Le groupe, dans Horus, est une abstraction adressage par l'intermédiaire de laquelle une collection de membres peut interagir avec son environnement. Un membre d'un groupe est une extrémité de communication qui peut générer des messages et auquel des messages peuvent être adressés. Un processus (ou plus exactement, l'extrémité de communication détenue par le processus) peut être lié à un groupe, quitter ce groupe, ou être exclu du groupe en cas de comportement défaillant. De telles opérations entraînent des changements dans la configuration du groupe qui seront pris en compte par le service qui gère l'adjonction et le retrait de processus du groupe. A chaque changement de vue, ce service le communique aux membres du groupe. Horus assure le synchronisme virtuel en fournissant les protocoles qui délivrent la suite de changements de vues diffusée, dans un même ordre pour tous les processus du groupe.

Aussi, suite à un partitionnement du réseau, le groupe peut être divisé en une partition primaire et d'autres partitions de moindre importance. Horus fournit les mécanismes qui assurent, après recouvrement du réseau, d'une part, le regroupement des partitions, et d'autre part, l'établissement d'une nouvelles vue de groupe qui sera communiquée à tous les membres.

Par ailleurs, la communication intra-groupe se fait par une primitive multipoint qui permet à un membre d'adresser, en absence de défaillances, un message à tous les membres contenus dans sa vue de groupe. Si un problème de communication survient, le processus émetteur exclut de sa nouvelle vue de groupe, le (ou les) membres qui n'ont pas reçu le message diffusé.

Pour satisfaire les divers besoins en communication de différentes applications, les protocoles Horus sont répartis sur 15 couches différentes. Ces couches peuvent être empilées comme des *légos* de sorte que lors de la construction d'une application, seulement les couches nécessaires sont intégrées. Les couches suivantes sont les plus importantes du système Horus :

- **COM**: dans cette couche aucun protocole n'est implémenté. Son but est d'offrir une interface invariable qui est empilée sur les couches de communication les plus basses (par exemple, UDP ou ATM).
- **NAK**: cette couche offre un multipoint FIFO fiable, ainsi qu'une communication point-à-point. Le schéma d'acquittement est celui de l'acquittement négatif. De plus, si aucun message n'a été réceptionné durant un certain laps de temps, une confir-

mation négative est diffusée vers le groupe.

- **MBRSHP**: contient les protocoles qui garantissent une transmission atomique (2.2.9) des messages, ainsi que la gestion de l'adhésion et le retrait de membres. Grâce au protocole Flush, ce niveau assure le synchronisme virtuel.
- **STABLE**: quand tous les membres actifs d'un groupe ont reçu un message, celui-ci devient stable. De plus, chaque membre peut décider lui-même quand un message est considéré comme étant reçu.
- **CAUSAL**: cette couche mémorise les relations causales existant entre les messages à l'aide d'un vecteur temporel. La couche ORDER, basée sur ce vecteur temps, est chargée de l'expédition des messages.
- **TOTAL**: par l'intermédiaire d'un jeton que doit recevoir (éventuellement aussi, demander) tout membre qui veut envoyer un message, cette couche implante l'ordre total dans la délivrance des messages. A la différence d'un jeton tournant sur un anneau virtuel comprenant tous les sites, celui-ci circule au sein d'un anneau fermé d'émetteurs (les membres qui émettent actuellement). Cela fonctionne bien si ces émetteurs ont une charge assez constante.
- **CLTSVR**: Comme la couche MBRSHIP montre une diminution importante dans ses performances à partir d'environ 1000 membres, la couche CLTSVR a été conçue pour permettre au système de supporter des applications de quelques milliers de membres par groupe. Dans cette couche, les membres sont divisés en deux ensembles: serveur et client. Chaque serveur utilise la couche MBRSHIP et est responsable d'une collection de clients vers lesquels il redirige tout message réceptionné. Chaque client ne communique qu'avec son serveur. Les clients des serveurs défaillants sont pris en charge par d'autres serveurs. Avec cette couche, la taille des groupes peut être fortement augmentée, mais le retard concernant la livraison des messages est inévitable. Cette couche peut être empilée plusieurs fois de sorte qu'une structure hiérarchique comparable à un arbre peut être créée.

D'autres *micro-protocoles* sont également implémentés, telles que les couches FC (contrôle de flux) et FRAG (partitionnement).

Le système Horus est implanté au dessus du noyau “-MUTS” (Multi-threading and Unreliable Transport Service) qui offre un multi-threading préemptif ainsi qu'une collection de pilotes (drivers) pour différentes interfaces réseau tels que les sockets UDP, l'interface Mach ATM et l'interface X-kernel.

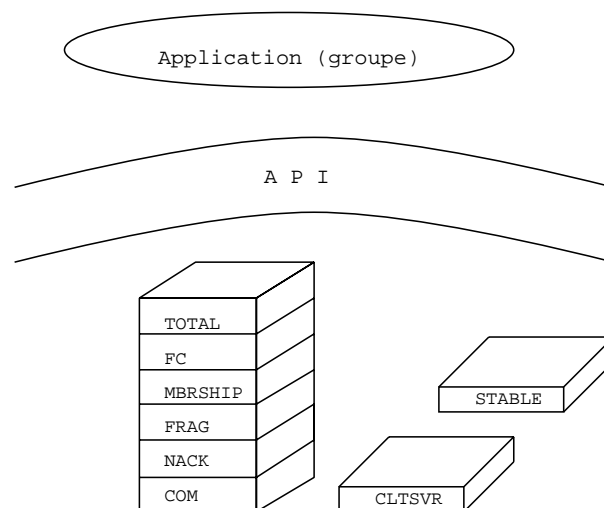


FIG. 2.3 - Couches de communication Horus

2.3.1.3 Totem

le système Totem [MMA⁺96] correspond à un set de protocoles de communication qui supporte la construction de systèmes répartis distribués et tolérants aux défaillances, tels que les systèmes de contrôle de trafic aérien ou les bases de données réparties pour les applications commerciales sensibles. Les mécanismes d'ordonnancement de message fournis par Totem permettent aux applications de maintenir la cohérence des données distribuées ou dupliquées en présence de défaillances. Les caractéristiques principales du système sont :

- communication multipoint ordonnée à travers des groupes de processus,
- un débit élevé et une latence (*latency*) prévisible faible,
- un code rendu portable à travers l'utilisation de caractéristiques standards d'Unix
- une rapide reconfiguration en cas de suppression d'un processeur défaillant, d'ajout de nouveaux processeurs et de recomposition d'un système partitionné,
- une continuité dans les services de toutes les composants d'un système partitionné à une niveau aussi consistant que possible avec l'état initial.

Aussi, Totem fournit une livraison de messages fiable et totalement ordonnée au sein de groupes de processus répartis sur un simple réseau local ou bien sur une interconnexion de multiples réseau locaux au sein d'un même domaine. Chaque réseau local interconnecté est organisé en un anneau logique. Un jeton circule, à travers l'anneau, comme un message point-à-point. Seul le site qui détient le jeton est autorisé à diffuser un message. Les champs

du jeton circulant entre les sites d'un réseau local permettent de fournir une livraison fiable (*safe delivery service*), totalement ordonnée (*Agreed delivery service*), la confirmation que les messages ont été reçus par tous les sites, le contrôle de flux et la détection de défaillances. L'ordre total est consistant à travers le réseau entier, malgré la défaillance de sites ou celle des communications, mais n'impose pas à tous les processus de délivrer tous les messages.

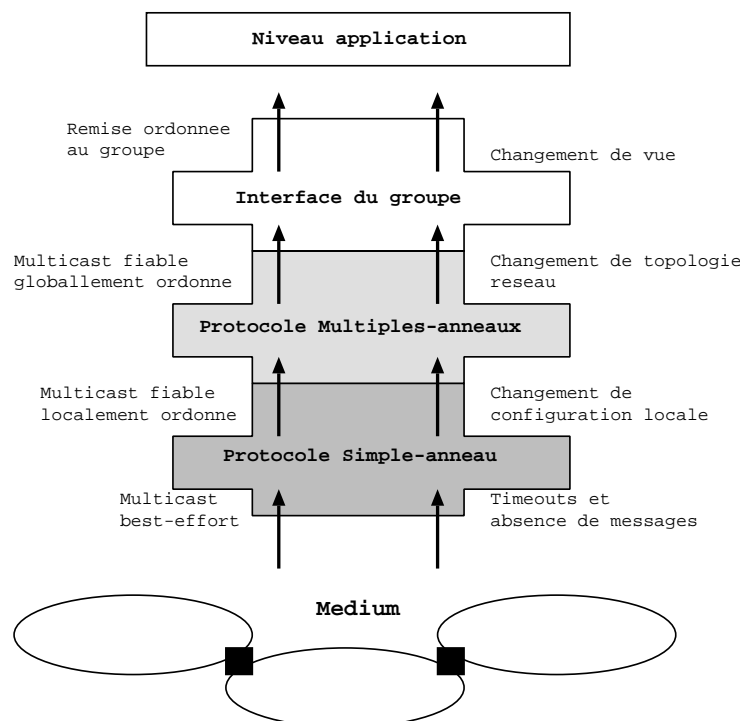


FIG. 2.4 - Architecture du système Totem

2.3.2 Protocoles multipoint de niveau transport OSI

2.3.2.1 Gestion des pertes dans les protocoles de transport

Le rôle d'une primitive de communication fiable est, en plus d'acheminer les message à travers le réseau d'interconnexion, de rendre transparent à l'utilisateur les mécanismes de détection et de correction d'erreurs pouvant se produire lors du transfert de messages. Parmi les mécanismes utilisés, les codes correcteurs d'erreur permettent de détecter, voir de corriger les erreurs de transmissions. Seulement, leur utilisation est coûteuse car elle exige la transmission de beaucoup d'informations redondantes. Le mécanisme le plus ré-

pandu est que le récepteur qui détecte une erreur de transmission ignore le message reçu. Une erreur de transmission détectée se traduit donc par une perte de message. D'autres possibilités de perte sont liées à la *congestion* d'équipements intermédiaires ou à l'insuffisance de ressources locales.

La technique traditionnelle de correction d'erreur consiste à détecter les messages perdus puis à les retransmettre. Elle utilise, généralement, l'un des deux mécanismes d'acquittements : positif ou négatif. Certains protocoles rendent inutile l'acquittement. Dans un échange de type requête/réponse, la réponse peut faire office d'acquittement. Nous présentons dans ce qui suit les mécanismes de détection de pertes, d'acquittements et de retransmission utilisés dans la communication multipoint.

2.3.2.1.1 Détection de pertes :

la détection de pertes peut être, aussi bien, à la charge de l'émetteur que des récepteurs. Pour cela, plusieurs méthodes sont utilisées :

- La première consiste à faire acquitter tous les messages reçus. Le site destinataire signale la bonne réception du message en transmettant un accusé de réception (ou *ACK*) à émetteur. Il effectue, ainsi, un *acquittement positif*. En cas d'absence d'acquittement, le message est retransmis par l'émetteur à l'expiration d'un certain délai de garde (ou *intervalle d'acquittement*). Un tel schéma peut conduire à une ambiguïté dans le mécanisme de retransmission [KP91]. En effet, lorsqu'un acquittement est reçu, si aucune indication n'est donnée quant au message acquitté (l'original ou celui retransmis), il est difficile de mesurer avec précision le temps d'aller-retour (*round-trip*⁷) sur la base duquel l'émetteur détermine l'intervalle d'acquittement.
- Dans la seconde méthode, la charge de détection d'erreurs est déplacée de l'émetteur vers le (ou les) récepteur(s). Pour cela les messages sont numérotés de manière séquentielle par l'émetteur. Le numéro de séquence permet aussi de détecter les messages redondants (ou doublons) qui ont pu être créés par des routeurs intermédiaires ou par l'émetteur suite à une retransmission inutile. Dans un réseau qui respecte l'ordre d'émission (tel que les réseaux locaux), une rupture dans la séquence des messages reçus est un signe de perte de message. Dans le cas de réseaux étendus, les paquets émis ne cheminent pas forcément par des routes identiques, un paquet peut précéder un autre paquet émis antérieurement. Ainsi, à la réception d'un paquet hors séquence, le destinataire fixe un délai d'attente pour l'arrivée du paquet manquant. A l'expiration de ce délai, il signale la perte par l'envoi d'une demande

7. *i.e.*, l'intervalle de temps entre l'émission d'un paquet et la réception de son acquittement.

de retransmission (ou *NACK*) du paquet perdu, il effectue ainsi un *acquiescement négatif*.

Seulement pour détecter un “trou” dans la séquence des messages reçus, il faut d’abord avoir reçu le (ou les) message(s) suivant(s). Afin de permettre une détection de perte assez rapide, un émetteur devra transmettre à intervalles de temps régulier. Lorsque temporairement, il ne dispose d’aucun message à émettre, il devra transmettre un message vide permettant aux récepteurs de détecter une perte éventuelle [CM84] [KTH89] [DK93]. Cette solution a été adoptée dans le protocole SRM [FJM95] où chaque site récepteur du groupe envoie, à son initiative, périodiquement un message (*session message*) qui annonce aux autres membres le dernier numéro de séquence réceptionné. Le message session contient également une estampille utilisée pour estimer la distance temporelle de chaque site par rapport aux autres.

- Certaines solutions composent les deux types d’acquiescement positif et négatif. Dans [XTP92], le schéma d’acquiescement est négatif mais la détection de perte se base aussi sur des demandes explicites d’acquiescements positifs de la part des émetteurs. un récepteur peut détecter une perte lorsqu’il constate qu’il n’a pas reçu le message pour lequel l’acquiescement est requis. Dans un contexte de communication de groupe, si les ACKs sont diffusés, leurs destinataires pourront les exploiter pour détecter les pertes. Ce type de détection de perte est utilisé, soit en conjonction avec les ruptures de séquence [CM84] [XTP92], soit comme mécanisme unique de détection de perte [Raj92].

Dans les deux schémas d’acquiescement, la retransmission des paquets perdus peut se faire selon le mode unicast ou multicast. La seconde alternative simplifie la tâche de l’émetteur mais impose une charge non nécessaire pour les sites ayant reçu correctement le message.

2.3.2.1.1.1 Acquiescements positifs :

Ce schéma de contrôle d’erreurs est typique des protocoles de transport dans les communication point-à-point tel que TCP. Dans ce schéma, les récepteurs doivent retourner un acquiescement explicite pour chaque message ou ensemble de message reçu. Si un ACK est perdu, le message correspondant est réémis. L’acquiescement peut être effectué par un paquet de contrôle adapté, comme il peut être transporté par le prochain message de chacun des récepteurs. Cette dernière technique est appelée le *piggy-backing*.

Le principal avantage de l’acquiescement positif, est que l’émetteur connaît régulièrement l’état de réception de chaque récepteur. De plus, elle permet au récepteur de supprimer

les messages acquittés des tampons associés. Cependant, pour s'assurer de la stabilité⁸ d'un message, l'émetteur devra collecter les ACKs de tous les membres de son groupe. Par conséquent, il devra connaître la configuration courante de son groupe, ce qui n'est pas aisément réalisable dans les environnement asynchrones. En outre, si la taille du groupe est importante, le réseau serait vite surchargé puisque chaque message est acquitté $n \Leftrightarrow 1$ fois (n étant la taille du groupe). Ainsi, l'émission d'un seul message "utile" est à l'origine de n messages transitant sur le réseau.

Pour ne pas avoir à maintenir une liste d'adhésion au niveau de chaque site, les ACKs sont diffusés à l'ensemble des participants au lieu d'être envoyés au seul émetteur. L'inconvénient est que cela peut accentuer, d'une part, le phénomène de congestion, et d'autre part le risque d'*implosion* présent au niveau de chaque site qui devra traiter les acquittements de tous les autres sites du groupe (ce qui représente $n \Leftrightarrow 2$ ACKs à traiter pour chacun des $n \Leftrightarrow 1$ sites récepteurs).

Les phénomènes de congestion et d'implosion peuvent également se produire même si les acquittements sont limités au seul émetteur. En effet, lorsqu'un grand nombre de récepteurs appartient au réseau local de l'émetteur. Les acquittements d'un message peuvent alors survenir au même moment pour cet ensemble de récepteurs [CP88]. Cela peut aussi être observé lors de la transmission de réponses suite à une requête diffusée vers un groupe de processus [AN95].

Deux solutions complémentaires de *contrôle de flux* sont alors nécessaires : la première est la technique de l'amortissement (*damping*) où chaque récepteur n'acquitte que s'il est le premier membre à acquitter. Dans le protocole XTP du projet Protocol Engine [XTP92], les récepteurs diffusent les ACKs à tout le groupe. Chaque récepteur annule sa propre diffusion de l'acquittement s'il reçoit l'ACK correspondant au message en instance d'être acquitté. La seconde solution est la technique de la temporisation ou de l'échantillonnage (*slotting*). Elle augmente l'efficacité de la première technique en imposant, pour chaque récepteur, un délais aléatoire avant la diffusion d'un ACK [Dan89].

2.3.2.1.1.2 Acquittements négatifs :

L'acquittement négatif permet à un récepteur de signaler la perte d'un ou de plusieurs messages et de demander explicitement leur retransmission. Les NACKs ne sont utilisés que pour les demandes de retransmission et ne permettent, donc, pas de s'assurer de la stabilité du message diffusé. Pour cela, tout émetteur doit conserver indéfiniment une copie de chaque message diffusé afin de satisfaire d'éventuelles demandes de retransmission. Cela soulève, au niveau de chaque site, le problème de la capacité de stockage qui peut

8. *i.e.*, la réception d'un message par tous ses destinataires.

être résolue par des techniques mixtes combinant acquittements négatifs et positifs : un acquittement est retourné, soit sur la demande de l'émetteur, soit spontanément par le récepteur lorsqu'un déséquencement est détecté.

Dans une communication de groupe, la gestion des retransmission n'est pas une tâche réservée au seul émetteur : n'importe quel membre qui a reçu le message pourrait le retransmettre. Pour cela, les acquittements négatifs peuvent être diffusés vers l'ensemble du groupe. Cela peut, toutefois, provoquer le phénomène de congestion évoqué plus haut. En effet, si la perte d'un message est due à une erreur de transmission sur le réseau local, les récepteurs acquitteront négativement dès la réception du message suivant, et donc de manière presque simultanée. Ce problème peut être évité en appliquant la technique de temporisation (2.3.2.1.1.1) à l'acquittement négatif. Ainsi, un récepteur ayant détecté une perte, temporisera un délai aléatoire avant de diffuser un NACK, si aucun autre récepteur ne l'a diffusé avant. Cette technique a été, à l'origine, utilisée dans XTP (*slotting/damping*) [XTP92] puis reprise dans de nombreux protocoles tel que SRM [FJM95].

Dabbous et Kiss [DK93] utilisent un mécanisme de file virtuelle pour assurer l'unicité des retransmission suite à des NACK diffusés. Dès qu'un membre émet un message, il se place en tête de la file virtuelle. Sa position dans la file sera modifiée chaque fois qu'il recevra un message d'un autre membre moins bien placé. Il sera alors rétrogradé d'un rang dans la file. Lorsqu'un membre détecte une perte, il diffuse, régulièrement, un NACK tant que la perte ne sera pas corrigée. Chaque NACK diffusé comporte un numéro d'ordre. Un membre entreprend de répondre à la demande de retransmission, uniquement si sa position dans la file est inférieur au numéro d'ordre de la demande. De cette manière, seul un membre placé en tête de file pourra répondre à la première demande. Chaque nouvelle demande concernera un nombre de plus en plus important parmi les membres.

2.3.2.1.1.3 Acquittements progressifs :

Une autre solution pour réaliser un acquittement positif, consiste à créer, au sein du groupe, un anneau logique sur lequel circule un jeton. Cette technique est utilisée dans CRP [CM84], où le passage de jeton qui s'opère entre sites récepteurs réalise un acquittement positif. A chaque réception d'un nouveau message ou à l'expiration d'une horloge, le jeton est passé par le site qui en est détenteur vers le récepteur qui le suit logiquement. Le jeton transporte une variable indiquant le numéro du message à acquitter. Le jeton est soit accepté par son destinataire, si celui ci a livré ce message ainsi que tous ses précédents, soit il reste bloqué, jusqu'à ce que le site destinataire reçoive l'ensemble de ces messages. De cette façon, un récepteur sait que les sites qui le précèdent logiquement ont reçu au moins autant de messages que lui depuis le début du tour. Les messages acquittés

au début du tour pourront être détruits après un tour complet du jeton.

Dans TPM [Raj92], le jeton transporte, cette fois, une variable dont le but est de signaler si un membre est en retard de réception par rapport aux autres membres (se trouvant sur des sites différents). Cette variable est initialisée par le premier membre qui vient de livrer un nouveau message. Le passage du jeton permet à son destinataire de comparer son état de réception et de constater un retard de réception éventuel. Dans ce cas, il recopie sur le jeton le numéro du dernier message qu'il a reçu : c'est l'acquittement par *décrémentation*. Après un tour complet du jeton, le numéro du message que tous les sites de l'anneau ont reçu, est connu.

L'acquittement progressif diminue fortement le nombre d'ACK et élimine, ainsi, le risque d'implosion. Il exige cependant une gestion de l'anneau virtuel et du passage du jeton.

2.3.2.1.2 Fiabilité partielle :

Certaines applications de groupe peuvent s'accommoder d'une fiabilité qui n'est pas absolue. C'est le cas des applications temps réel de transfert d'informations audio ou vidéo. D'autres applications tels que les services de localisation ou de collecte n'exigent une transmission fiable que vers un sous-ensemble, défini ou non, du groupe de processus. Nous présentons quatre types de fiabilité partielle adaptés à ces différents types d'applications.

1. Le premier consiste à fixer un seuil acceptable de taux de pertes. Cette solution n'utilise pas d'acquitements mais une propriété statistique liée à la probabilité de transmettre au moins une copie d'un message à tous les membres du groupe. Le principe est de diffuser M copies du message, le nombre M étant choisi de sorte à ce qu'au moins l'une des copies soit livrée à tous les processus du groupe. Le seuil d'erreur toléré correspond donc à la probabilité qu'un ou plusieurs destinataires ne reçoive aucune des copies du message.

L'avantage de ce mécanisme est qu'il n'utilise pas de message d'acquittement. Il peut s'appliquer aux groupes de grande taille sans pour autant générer d'implosions [Moc83]. Ses inconvénients sont que certains membres doivent traiter les $M \Leftrightarrow 1$ messages inutiles. Aussi, son comportement est le même en présence ou en absence de pertes (puisque'il suppose la présence d'erreurs). Enfin, il repose sur l'hypothèse que les pertes sont indépendantes.

2. Une autre utilisation des propriétés statistiques est faite par les protocoles dits *statistiquement fiables*. La fiabilité statistique approche la fiabilité totale (telle que définie dans 2.2.8 mais est à moindre coût. L'émetteur n'a pas besoin de maintenir

les états respectifs des récepteurs. Quand il envoie un message, il le garde dans une mémoire tampon un certain temps T . Après ce laps de temps, l'émetteur détermine s'il doit retransmettre le message. Cette décision est basée sur l'analyse des états des récepteurs, accumulés durant le temps T . La diffusion se termine lorsqu'une proportion donnée de récepteurs a reçu le message émis. Ce type de protocole s'avère particulièrement efficace pour les groupes dynamiques de grandes dimensions.

3. Le troisième type définit le sous-groupe, vers lequel la transmission doit être fiable, par sa composition [CP88] ou son cardinal [Raj92], ou encore par la vitesse de réaction de ses membres. Dans le cas où l'ensemble minimal sur lequel s'applique la fiabilité n'est défini que par le nombre N de ses membres, on parle alors de N -fiabilité (n -resilience). Les applications de localisation de services [CM88] ne requièrent, par exemple, qu'une diffusion 1-fiable.
4. Le quatrième offre une communication dite *best effort* (ce qui est possible) dans laquelle chaque message perdu ou altéré n'est pas retransmis. De plus, les messages altérés ne sont pas délivrés à leurs destinataires. Ce mode de communication multidestinataires est fourni, entre autres, par le protocole de transport UDP. Il est particulièrement adapté aux applications temps-réel pour lesquelles la fiabilité n'est pas un attribut critique et où les retransmissions sur erreur ne sont pas d'un grand intérêt.

2.3.2.2 Exemples de protocoles multipoint fiables

2.3.2.2.1 eXpress Transfer Protocol(XTP) :

XTP [SF92] est un protocole de transport conçu pour les systèmes temps réel et multimédia. Il regroupe dans une même couche appelée couche de transfert (*transfer layer*) les fonctions de niveau transport et celles de routage afin d'améliorer le contrôle de congestion [Mau93]. La diffusion des données se fait d'un émetteur vers un groupe de récepteurs. Le flux de données est unidirectionnel. Le développement du protocole s'est basé sur les hypothèses suivantes :

- L'émetteur ne doit avoir aucune connaissance de l'adresse explicite de l'ensemble de récepteurs. Il ne lui est, ainsi, pas nécessaire de maintenir l'état de réception individuelle de chaque destinataire.
- Les récepteurs ne sont pas autorisés à émettre des messages de données à l'émetteur.

- Les récepteurs ne peuvent générer des messages de contrôle sans une directive explicite de l'émetteur qui, seul, a la responsabilité de contrôler les échanges d'informations relatives aux états des différents récepteurs.

La transmission multipoint dans XTP se fait selon deux modes possibles : avec ou sans acquittements. Deux types de fiabilité sont offerts : best effort et statistiquement fiable. Dans le second cas, l'émetteur, qui ne connaît pas la cardinalité des récepteurs, utilise une méthode de retransmission basée sur le temps. Lorsqu'une transmission non fiable est choisie (on parlera plus loin de *qualité de service* (ou QoS) non fiable), l'émetteur désactive la procédure de retransmission en utilisant le bit NOERR du champ entête du message diffusé.

XTP attribue à chaque association établie entre un émetteur et un récepteur, une information d'état. Il s'agit d'une structure qui contient les paramètres importants de la communication tels que le numéro de séquence courant, le débit, la taille de la fenêtre d'anticipation⁹, le cheminement utilisé, l'estimation du temps d'aller-retour (ou *round-trip*¹⁰), et la qualité de service offerte.

L'émetteur diffuse périodiquement un paquet de contrôle CNTL[*sync=k*]. Ce paquet sert à solliciter les informations d'état de la part des récepteurs. A la réception de ce paquet, chaque récepteur l'acquiesce par un paquet CNTL[*echo=k*] servant à retourner les informations demandées par l'émetteur. Ceci permet de synchroniser l'émetteur et les différents récepteurs relativement aux informations d'état. L'état de réception d'un site est représenté par deux variables *rseq* (*received sequence number*) et *dseq* (*delivered sequence number*). La variable *dseq* correspond au numéro de séquence augmenté de 1, du dernier message reçu et délivré au récepteur correspondant par sa couche XTP. Cette information est la seule nécessaire pour libérer le buffer de l'émetteur. Si un paquet numéroté *rseq* est perdu ou altéré, l'émetteur le retransmet ainsi que tous ceux qui le suivent.

XTP n'offre pas un mécanisme spécifique de gestion d'adresses de groupe, une autorité externe, tel que IGMP [Dee89] est nécessaire pour la transmission d'adresses aux récepteurs. Dans le cas, où un récepteur veut rejoindre une conversation multicast en cours, il envoie un paquet spécial indiquant l'adresse de groupe auquel il veut se joindre. L'émetteur répond avec un paquet contenant les informations sur la conversation.

XTP utilise le round-trip comme paramètre de contrôle de congestion. En effet, à chaque intervalle de temps (dépendant du round-trip), un serveur XTP sollicite, de la part de chaque récepteur, un paquet d'acquiescement. Le contrôle de débit est explicite. En effet,

9. Nombre de messages diffusés par un émetteur sans être bloqué par l'attente des ACKs

10. i.e., l'intervalle entre l'émission d'un paquet et la réception de son acquiescement.

le protocole négocie pour chaque association la taille maximale du segment de données que l'émetteur peut transmettre ainsi que les valeurs maximales du débit et des rafales.

2.3.2.2 Multicast Transport Protocol (MTP) :

MTP (Multicast Transport Protocol) [AFM92] fournit un service de diffusion fiable pour un groupe dynamique de processus. Il ne nécessite pas la connaissance par un membre de tous les membres prenant part à la communication. Trois classes de membres composent le groupe :

- Le *master* crée, initialise et contrôle le comportement de la communication. Il peut aussi bien émettre et recevoir des messages des autres membres du groupe.
- Les *producteurs* sont les membres autorisés à transmettre et à recevoir les messages émis par les membres du groupe.
- Les *consommateurs* sont seulement capables de recevoir les données des producteurs.

Pour rejoindre une communication de groupe, un membre envoie une requête “*join*” au master. Cette requête contient la classe du membre, le type de QoS (fiable, non fiable), le type de transport ($1 \times N$, $N \times N$), le débit minimum et la taille maximum des unités de données. Une phase de négociation succède où le master compare les paramètres de la communication avec ceux de la demande. En cas de compatibilité, il envoie un message unicast contenant les paramètres courants de la communication ainsi que l'identificateur de la connexion multicast (qui formera le TSAP (Transport Service Access Point) de destination pour les messages diffusés par le nouveau membre). Si le candidat au groupe accepte les paramètres transmis, il rejoint le groupe.

Avant d'envoyer les données, chaque membre doit demander un “jeton de transmission” au master. Le jeton contient le numéro de séquence du message nécessaire au respect de l'ordre de délivrance.

MTP utilise une fenêtre pour le contrôle de flux. Ce champ indique le nombre maximum de paquets de données pouvant être diffusés par un membres durant un intervalle de temps préfixé. Chaque producteur garde le message diffusé pendant un certain temps dit de “rétention” dans le but d'une éventuelle retransmission.

Le recouvrement des erreurs de transmission utilise un schéma de retransmission sélective où les messages sont retransmis par multicast.

MTP a recours à une autorité externe pour l'allocation d'une adresse de groupe. La spécification MTP précise que le master doit vérifier que l'adresse NSAP choisie n'est pas

déjà utilisée. Pour cela, il envoie des requêtes d'adhésion à un éventuel groupe existant avec cette adresse. Dans l'absence de réponse, il procède à la création du groupe.

2.3.2.2.3 MTP-2 :

MTP-2 [BOG⁺94] est une version révisée de MTP qui résout certains problèmes mis en évidence dans MTP et qui introduit des fonctionnalités additionnelles. La fiabilité dans MTP-2 repose sur une détection de perte par les sites récepteurs. Les retransmissions, basées sur les NACKs, sont diffusées à tous les utilisateurs. Le protocole permet, néanmoins, une retransmission vers certaines régions du groupe de diffusion.

De plus, MTP-2 permet aux utilisateurs de spécifier les paramètres qui interviennent dans la qualité du service de transport fourni. Différents services de fiabilité sont supportés.

La gestion de groupe dans MTP-2 est déléguée à un site particulier, le *master*, qui est responsable des adhésions à une session multicast.

La politique de contrôle de flux permet aux récepteurs de tirer un ensemble de paramètres comme la constante de temps de transport, le nombre de paquets données par la constante et la taille du buffer de retransmission au niveau de chaque émetteur.

2.3.2.2.4 Reliable Adaptive Multicast Protocol (RAMP) :

Dans RAMP [KZ96], chaque station locale dispose d'une entité MGA (Multicast Group Authority) qui s'occupe de la gestion de l'espace d'adressage multicast, des fonctions d'adhésion et de la gestion du groupe. MGA maintient également l'état de tous les services enregistrés.

RAMP offre un service avec différentes QoS associés à un flux. A l'intérieur de chaque paquet transmis, un champ de l'en-tête indique le niveau de QoS associé. Toute requête de changement de QoS d'un flux doit être envoyée au MGA.

Le protocole utilise le numéro de séquence pour réaliser l'ordonnancement des messages diffusés. Il permet la livraison des paquets sans ordre pour les applications qui le requièrent.

RAMP utilise un recouvrement d'erreurs sélectif basé sur un acquittement négatif. Les requêtes de retransmission sont collectées pendant un certain temps au delà duquel l'émetteur réémet les messages requis. Afin de diminuer les retransmissions dupliquées, la valeur du temporisateur est proportionnelle aux nombres de requêtes de retransmission reçues. Si la perte de messages concerne un nombre important de récepteurs, ces derniers sont retransmis par multicast, sinon la retransmission se fait par unicast. En cas de recouvre-

ment des buffers de retransmission par des paquets plus récents, une réponse négative est émise aux destinataires.

RAMP utilise un mécanisme de contrôle de flux basé sur un contrôle de débit dynamique. En effet, il utilise des facteurs de réduction de débit pour recalculer le débit pour la retransmission. De plus, il utilise le mécanisme de priorité pour le trafic de données.

2.3.2.2.5 Reliable Mutlicast Transport protocol (RMTP) :

A l'instar des autres protocoles multipoint, RMTP [LP96] (développé par NTT et IBM) est basé uniquement sur une communication de bout-en-bout (*end-to-end protocol*). En d'autres termes, il fournit un service de niveau transport en déléguant les tâches de routage, "switching" et de contrôle de paquets aux couches basses de l'architecture réseau qui le supporte. RMTP est un protocole orienté connexion incluant la confirmation des récepteurs. Il est destiné aux applications de distribution massive et fiables de données pour un grand nombre d'utilisateurs authentifiés. La configuration du groupe est statique durant la session.

RMTP étant réalisé au dessus de la couche UDP/IP, la fiabilité dans RMTP est atteinte par la communication de bout-en-bout au lieu de l'utilisation des passerelles. Multicast IP est utilisé pour l'acheminement du flux de données du serveur vers les récepteurs. La détection d'erreurs de transmission est à la charge des récepteurs et les NACKs sont utilisés pour signaler les messages perdus. Pour une étape de retransmission de données, consistant en l'ensemble des messages qui n'ont pas été correctement réceptionnés par tous les récepteurs, un seul NACK ou ACK est utilisé. Les ACKs sont utilisés pour une complète notification de réception de données par les récepteurs, au lieu d'une notification implicite causé par un *timeout*. Pour éviter l'effet d'implosion d'ACKs, un algorithme temporel (*Backoff time algorithm*) est appliqué. Un paquet de sondage (*POLLing packet*) est également envoyé par les serveurs aux sites récepteurs silencieux. En plus, des retransmissions individuelles, cette technique permet d'éviter les retransmissions multicast inutiles. RMTP utilise une fonction de retransmission séparée par laquelle les messages perdus sont retransmis de manière individuelle aux sites récepteurs indisponibles durant la phase de retransmission principale.

RMTP utilise la technique de surveillance de débit (*Monitoring-based rate*) pour contrôler le flux. Ce mécanisme tend à améliorer le débit tout en réduisant les retransmissions redondantes.

2.3.2.2.6 Reliable Multicast Protocol (RMP) :

RMP [MWK95] est un protocole qui offre un service multicast totalement ordonné au dessus de la couche UDP/IP. Cette dernière fournit un service de diffusion avec meilleur effort, et un service d'adresses de groupe IP (classe D). RMP s'adapte aussi bien aux réseaux locaux qu'aux réseaux étendus et offre un choix de QoS varié (allant du service non fiable à une diffusion atomique k -tolérant aux défaillances) qui peut être sélectionné pour chaque message diffusé, selon les besoins de l'application.

RMP est organisé autour de trois entités : les processus RMP, les anneaux à jeton et les sites à jeton.

- Les *processus RMP* qui utilisent RMP comme protocole de transport. Chaque processus est identifié de manière unique par respectivement, l'adresse IP du site hôte et le numéro de port qu'il utilise.
- Les *anneaux à jeton* représentent l'ensemble des processus RMP qui communiquent pour réaliser l'ordonnancement des messages. La composition de l'anneau est dynamique, elle permet d'identifier le groupe en tant que tel.
- Les *listes à jeton* contiennent les listes des membres des anneaux correspondants. Ces listes sont créées par un processus *Origine* qui associe à chaque nouvelle liste un identificateur unique.

Lorsqu'un site détecte une défaillance, il peut lancer une procédure pour créer une nouvelle liste à jeton. Le protocole permet de construire un nouvel anneau en s'assurant qu'il y a un nombre minimum de sites provenant de l'anneau d'origine. RMP autorise l'adjonction de nouveaux sites durant la procédure de reconstruction. Pour améliorer l'extensibilité (*scalability*), RMP permet à des processus qui en sont pas membres d'un groupe, d'envoyer des messages à celui-ci et d'en recevoir les réponses et ce, à travers un mécanisme multi-RPC. Pour la même raison, RMP permet à des processus qui ne peuvent accéder à un support de diffusion de participer à la communication de groupe.

RMP traite le contrôle de congestion et le contrôle de flux comme un seul problème. Le mécanisme de contrôle utilise les algorithmes développés par Van Jacobson pour le protocole TCP (rrt, slow start, taille dynamique de la fenêtre, retransmission exponentielle) [Jac88]. Chaque émetteur du groupe maintient une fenêtre glissante (*sliding window*) régulée par les horloges de retransmission, les NACKs (des récepteurs), et les ACKs (du site-jeton). Un émetteur utilise la technique du *slow-start* pour augmenter la taille de sa fenêtre d'un paquet à chaque fois qu'un ACK est reçu. Quand l'horloge de retransmission expire (signalant une congestion), l'émetteur réduit de manière *exponentielle* la taille de sa fenêtre de transmission. Pour le contrôle de flux, un site récepteur qui détecte une perte,

diffuse un NACK au lieu de l'envoyer au site jeton. Quand un émetteur reçoit un NACK, il le traite juste comme une expiration d'horloge (réduction exponentielle de la fenêtre)

2.3.2.2.7 Scalable Reliable Multicast (SRM) :

Le développement du protocole SRM repose sur le principe de ALF (Application Level Framing) selon lequel la meilleure façon de satisfaire les divers besoins d'une application est de laisser autant de fonctionnalités que possible à l'application. Pour cela, les algorithmes de SRM doivent être conçus pour satisfaire la définition minimale d'un multicast fiable: "distribution finale de la donnée entière à tous les membres du groupe, sans imposer aucun ordre de distribution particulier". En cas de besoin, des mécanismes de niveau applicatif peuvent être ajoutés pour imposer un ordre de livraison souhaité. ALF a été plus tard élaboré avec un mécanisme *light-weight* de rendez-vous, basé sur le Multicast-IP. Le résultat est connu sous le nom de LWS (Light-Weight Session).

SRM [FJM95] a été développé pour distribuer les données parmi les sites d'une application coopérative de type tableau blanc (Wb). SRM est basé sur l'acquiescement négatif pour assurer une transmission multipoints fiable. Chaque site participant diffuse des NACK pour demander la retransmission des données perdues. Le recouvrement d'une perte peut être pris en charge par n'importe quel site du groupe qui détient la donnée requise. Le contrôle de transmission est, ainsi, basé sur les techniques de *slotting-damping* proposées dans XTP. En effet, pour éviter les retransmissions multiples d'une même donnée, les retransmissions sont diffusées vers l'ensemble du groupe (damping). De plus, un site qui détecte une perte, attend une période de temps aléatoire avant de diffuser son paquet requête. Les sites qui reçoivent les demandes de retransmission, temporisent à leur tour avant de retransmettre la donnée (slotting). Si un site reçoit une demande de retransmission alors qu'il est lui-même en instance de la diffuser, il annule sa propre demande. De même, un site qui reçoit une donnée retransmise alors qu'il est sur le point de la diffuser, annule sa retransmission.

Pour rejoindre une conférence en cours, un site annonce sa présence par un message d'adhésion. Les membres courants mettent à jour leur vue avec le site ajouté. La vue du nouveau membre est progressivement mise à jour à travers les *messages session* périodiques générés par les membres lorsque le groupe reste inactif durant une certaine période de temps. Les messages session sont également utilisés pour mesurer les distances temporelles (*round-trip*) entre membres, utilisée pour positionner les horloges de recouvrement d'erreurs (précédant la diffusion des requêtes et des données retransmises). Ainsi, les sites les plus proches auront les intervalles de temps les plus courts au niveau de leurs horloges. Cela permet d'éviter l'effet d'implosion des NACKs et des retransmissions.

2.3.2.3 Classification

2.3.2.3.1 Fiabilité :

Les protocoles de transport non fiables assurent une livraison “best effort”, alors que les protocoles fiables fournissent un mécanisme qui assure la réception des messages par chacun des sites du groupe. Les critères suivants caractérisent les différents protocoles décrits ci-dessus :

2.3.2.3.1.1 Mécanismes de fiabilité :

Les protocoles multicast fiables doivent détecter les erreurs avant de demander leur retransmission. Une approche consiste à déléguer à l'émetteur la tâche de détecter l'occurrence d'une erreur. Il s'agit de l'acquittement positif où l'émetteur s'assure, via les ACKs, que tous les récepteurs ont bien reçu le message diffusé. Une alternative à première approche est de rendre chaque récepteur responsable de la détection ses erreurs de réception. Il s'agit de l'acquittement négatif, où le récepteur signale, via un NACK, un “creux” dans les numéros de séquence reçus. La plupart des protocoles multicast utilisent soit un mécanisme d'acquittement négatif (SRM, MTP-2, RAMP, HORUS), soit optent pour une approche mixte combinant les deux approches (XTP, RMTP, XTP). D'autres protocoles utilisent la technique d'acquittement positif pour satisfaire la fiabilité (TPM [Raj92], Totem).

2.3.2.3.1.2 Requête de recouvrement :

La pratique commune pour signaler l'occurrence d'une erreur est d'envoyer un paquet de contrôle qui peut être, soit un ACK, soit un NACK. Etant donné que cette pratique peut conduire à un effet d'implosion des paquets de contrôle, des techniques ont été mises au point pour réduire le nombre de paquets requête.

2.3.2.3.1.3 Contrôle sur les feedbacks :

Les protocoles multicast qui utilisent l'acquittement négatif, doivent d'éviter le problème d'implosion des feedbacks (ou informations de retour) en fournissant des mécanismes qui restreignent le nombre de feedbacks générés par le groupe multicast. Les solutions proposées peuvent être classées en deux familles [Gro96]: celles basées sur la structure (*structure-based*) et celles basées sur le temps (*timer-based*). La première solution repose sur la désignation d'un site (soit un serveur dédié, soit un membre pré-assigné) chargé de filtrer les feedbacks. Une autre approche de la première solution consiste à organiser

le groupe en une certaine structure permettant de filtrer les feedbacks (RMTP [LP96], TMTP [YGS95]). La seconde solution repose sur une suppression probabiliste des feedbacks pour éviter l'implosion au niveau de la source d'émission. Les récepteurs dans SRM diffèrent, en effet, leurs demandes de retransmission pour un intervalle de temps uniformément distribué entre l'horloge courante et la distance temporelle les séparant de la source. Cette opération est appelée *slotting*. De ce fait, seuls les membres les plus proches de la source émettent leurs requêtes, permettant, ainsi, aux autres membres de supprimer leurs propres requêtes (*damping*). Cette opération est aussi utilisée dans les modèles de correction d'erreurs initiée par l'émetteur afin d'éviter l'envoi d'un acquittement qui serait un double d'un acquittement déjà émis par un autre site récepteur.

2.3.2.3.1.4 Retransmission :

Le problème final du multicast fiable est le procédé de retransmission des messages perdus. Les différentes approches existantes incluent la transmission unicast du message perdu vers le récepteur approprié, sa diffusion vers tous les récepteurs en les laissant filtrer les messages dupliqués, faire exécuter les retransmissions par les routeurs, ou demander le message auprès du site voisin.

2.3.2.3.2 Contrôle de congestion et de flux :

Le contrôle de débit de messages dans le multicast est compliqué par le fait que le protocole se doit d'accomoder plusieurs récepteurs simultanément. La façon de réaliser le contrôle de débit a un impact significatif sur les performances générales du protocole. A titre d'exemple, si l'émetteur opère constamment selon la vitesse de traitement du récepteur le plus lent, les processeurs récepteurs plus rapides restent dans un état latent alors qu'ils peuvent recevoir des informations additionnelles. Les mécanismes de contrôle de flux doivent donc adapter le débit de l'émetteur à ceux des différents récepteurs. Pour ce faire, deux principaux mécanismes sont utilisés [Dio94]:

Stop and wait : l'émetteur s'arrête et attend les acquittements après émission d'un message ou d'un ensemble de message.

Fenêtre de congestion : ce mécanisme peut être géré de deux façons :

- *Acquittements à l'initiative des récepteurs* : l'émetteur gère fenêtre de largeur K messages (numérotés de k à $k + K \ominus 1$). Un curseur mobile (ou bien *glissant*) pointe sur le prochain message à envoyer. Trois cas peuvent survenir :

1. Réception d'un acquittement négatif sur une donnée de rang n (n étant à

l'intérieur de la fenêtre): le curseur revient au message du rang retourné.

2. Réception de tous les acquittements positifs relatifs au message de rang n : le curseur glisse pour pointer sur le message de rang $n + 1$.
 3. Le dernier message de la fenêtre a été transmis : le curseur point désormais sur le message de rang $k + K$. Ainsi, K chances ont été laissées aux récepteurs pour recouvrer une éventuelle perte de message de rang k . Le paramètre K déterminé en fonction de la probabilité (établie) de perte d'un message, la rafale attendue de messages, et le délai de transmission du réseau.
- *Acquittements à la demande de l'émetteur* : le mécanisme est décrit par l'algorithme du saut (*bucket-algorithm*). Chaque intervalle de temps ST , un émetteur demande les paquets d'acquiescement correspondant aux messages émis. ST dépend du temps d'aller-retour mesuré à la connexion. Sa durée correspond au temps nécessaire à la réception d'une réponse à la demande d'acquiescement. Pour chaque requête d'acquiescement, l'émetteur crée un saut qui collecte les acquiescements correspondants. L'émetteur requiert au moins un message de contrôle de flux pour chaque récepteur, car il ne peut faire glisser le curseur de la fenêtre de congestion sans l'agrément de tous les récepteurs. L'algorithme contrôle les B sauts les plus récents. B est paramétré de sorte à laisser suffisamment de temps à un récepteur pour avoir au moins un de ses ACKs dans un des B sauts, malgré le round-trip correspondant à un transfert requête/acquiescement, et malgré les défaillances éventuelles du réseau. Si aucun ACK, présent dans l'un des sauts, ne correspond à un site récepteur, ce dernier est considéré comme défaillant. Chaque intervalle ST , l'émetteur examine les sauts et initie une phase de retransmission à partir des sauts les moins récents. Cela permet à l'émetteur de recevoir l'acquiescement du récepteur le plus lent et de ne procéder qu'à une seule retransmission.

2.3.2.3.3 Ordonnancement :

Bien que la livraison ordonnée n'est pas une exigence dans les protocoles multicast fiables, une partie d'entre eux assurent un service d'ordre total pour répondre à des besoins spécifiques des applications qu'ils supportent.

2.3.2.3.4 Gestion de groupe :

Si une session multicast permet l'adhésion et le retrait de membres pendant son déroulement, la durée de la session peut être plus longue que la durée de participation de chaque

membre. Ces adhésions tardives et retraits prématurés peuvent avoir un impact sur les mécanismes de recouvrement d'erreurs et de contrôle de flux des protocoles multicast.

Une partie de ces protocoles ne requière aucune connaissance de la configuration du group : les processus source émettent des données vers des adresses multicast auxquelles les processus récepteurs peuvent adhérer pour recevoir les données. Ce modèle de gestion implicite de groupe correspond à la méthode qu'utilise le Multicast IP pour gérer les groupes sur Internet. Il possède, de ce fait, une meilleure propriété d'extensibilité que le modèle de gestion explicite.

D'autres protocoles requièrent un contrôle total sur la configuration de groupe. Dans ce cas, la configuration est soit statique durant toute la session, soit un protocole de gestion de groupe explicite assure un service d'adhésion et de retrait fiable et cohérent. Une variante de cette dernière approche est de gérer l'adjonction de nouveaux sites et de les laisser quitter le groupe sans aucun contrôle. Cela peut concerner les applications de dissémination de données avec contrôle d'accès.

2.3.2.3.5 Application cible :

Bien que des protocoles, tels que RTP et XTP, ont été proposés comme des solutions générales au problème de la communication de groupe, les protocoles les plus récents ont été désignés pour des applications spécifiques. Des protocoles comme SRM et RAMP mais aussi une autre spécification de XTP, adressent les besoins en services temps-réel, sensibles aux latences, tels que les outils de téléconférences multimédia. D'un autre côté les protocoles, tel que RMTP (NTT) visent des services qui ne sont pas sensibles aux latences mais requièrent que les données soient livrées dans leur intégralité.

Protocole	Mécanisme de fiabilité	Requête de réparation	Contrôle sur les feedbacks	Retransmission	Contrôle de flux	Ordre	Gestion de groupe
RMP	ACK/NACK	multicast	-	unicast	fenêtre glissante	total	explicite
MTP-2	NACK	unicast	-	multicast	fenêtre fixe	total	explicite
RAMP	NACK	unicast	-	unicast/multicast	débit dynamique	à la source	explicite
XTP	ACK/NACK	unicast / multicast	Timer-based	unicast/multicast	débit fixe	à la source	explicite
RMTP	ACK	unicast	Structure-based	unicast/multicast	fenêtre	-	implicite
SRM	NACK	multicast	Timer-based	multicast	-	-	implicite

TAB. 2.1 - *Tableau comparatif*

2.3.2.4 Protocoles multipoint à ordonnancement causal

L'exécution d'un protocole de diffusion génère un ensemble d'évènements (émission, réception, traitement d'un message, ...). Elle est caractérisée par deux contraintes séquentielles fondamentales :

- les évènements produits sur un site (ou processeur) sont totalement ordonnés dans le temps,
- si l'on considère un message m , l'évènement $émission(m)$ précède l'évènement $réception(m)$.

Ces deux contraintes séquentielles structurent l'ensemble des évènements en une relation d'ordre partiel [Lam78]. Cette relation d'ordre partiel, notée " \rightarrow ", est appelée *relation de précédence causale* ou *relation de causalité*. L'idée qui préside à la définition à la définition d'un ordre causal est de répercuter sur les remises de messages les relations de précédence causale qui existent sur leur émission.

De façon plus formelle, l'ordre causal est respecté si pour tout couple d'évènements $émission(m)$ et $réception(m')$, tels que $émission(m) > réception(m')$, alors :

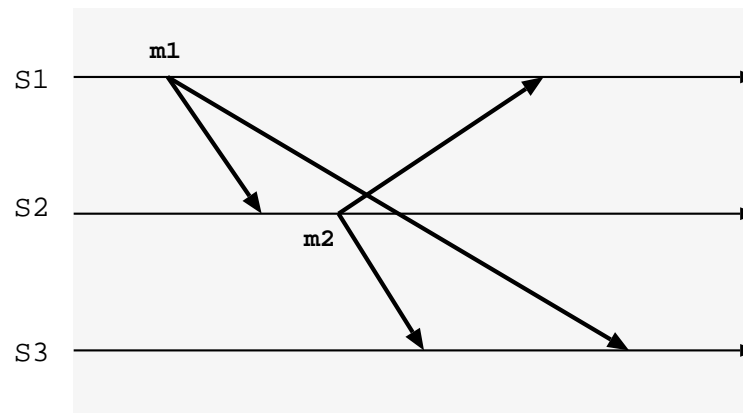
$\forall P \in destinataires(m) \cap destinataires(m')$, le message m est délivré avant le message m' .

Deux messages dont les émissions ne sont pas liés par la relation de causalité sont dits *causalement indépendents* ou *concurrents*. On écrit :

$$m_3 \parallel m_5 \Leftrightarrow \neg(m_3 \rightarrow m_5) \text{ et } \neg(m_5 \rightarrow m_3)$$

L'ordre causale stipule qu'un message m ne peut être remis à l'application que si tous les messages qui appartiennent au passé de m ont été remis à cette application. Par définition un message appartient au *passé causal* (ou à l'*historique*) de m s'il a été remis avant l'émission de m .

Tous les protocoles de diffusion causale étudiés se déroulent en une phase: 'es qu'un site S_i reçoit un message m dont le passé causal contient uniquement des messages déjà remis par S_i , alors S_i remet m . Cette remise immédiate peut en présence de défaillances être à l'origine d'un blocage. Considérons l'exemple suivant: S_i diffuse un message m_1 qui est reçu uniquement par S_j . S_j diffuse à son tour un message m_2 dépendant de m_1 . S_i et S_j défont par la suite. Aucun des sites du groupe de diffusion, à la réception de m_2 ne peut obtenir le message m_1 . Ces processus récepteurs sont alors bloqués. Un site S_i ayant délivré un message m peut, donc, être amené à le retransmettre tant qu'il existe un site S_j

FIG. 2.5 - *Exemple d'ordre causal*

n'ayant pas reçu m . Il est, donc, nécessaire que S_i garde une copie des messages délivrés en prévision d'une éventuelle retransmission. Toutefois, le protocole doit permettre à un site de savoir quand la copie d'un message peut être détruite.

2.3.2.4.1 Respect de l'ordre par transmission de tampon historique :

Une première réalisation du protocole CBCAST (cf. 2.3.1.1) reposait sur la transmission par un processus $P1$, avec tout message m , d'un tampon historique comprenant une copie de tout autre message m' précédant causalement m et non encore délivré par $P1$, même si le processus récepteur $P2$ n'est pas destinataire de m' [?]. Ainsi, toutes les informations nécessaires à une livraison respectant l'ordre causale sont incluses dans le message transmis, ce qui évite toute autre communication entre les deux processus. Ce protocole se caractérise par son coût en termes de taille des messages et des buffers ainsi que par la charge qu'il induit au niveau de réseau. Diverses optimisations ont permis de réduire la taille du tampon, notamment en remplaçant le contenu d'un message par un identificateur unique permettant de le retrouver.

2.3.2.4.2 Respect de l'ordre par horloges vectorielles :

Cette technique est basée sur la notion d'horloge logique introduite par [Lam78]. Elle est utilisée dans une réalisation plus récente du protocole CBCAST [BSS91] pour assurer un ordre causal pour des groupes multiples. Chaque membre d'un groupe entretient une horloge vectorielle logique qui reflète l'historique local des communications en comptabilisant les messages émis par chacun des membres du groupe. Cette horloge accompagne chaque message émis. Elle permet aux récepteurs de déterminer l'ordre de livraison et

de réactualiser leur propre horloge. Les émetteurs sont tenus d'envoyer régulièrement des messages vides lorsqu'ils n'ont rien à transmettre pour permettre aux récepteurs d'actualiser leur horloge. Nous décrivons dans ce qui suit le principe de fonctionnement du protocole CBCAST : Chacun des processus $p_i (i = 1..n)$ entretient une horloge V_i .

- Avant de diffuser un message m , un processus du groupe p_i exécute $V_i[i] := V_i[i] + 1$, et associe au message m une estampille V_m égale à V_i
- A la réception d'un message m , envoyé par p_i et estampillé V_m , un processus $p_j (j \neq i)$ retarde la délivrance de m jusqu'à ce que les conditions suivantes soient remplies :
 - $V_m[i] = V_j[i] + 1$
 - $\forall k \neq i : V_m[k] \leq V_j[k]$
- Après délivrance de m , l'horloge locale V_j est mise à jour de la façon suivante : $V_j[k] := \max(V_j[k], V_m)$ pour $k = 1..n$
- un processus ne retrade pas la livraison d'un message qu'il diffuse.

Le schéma suivant reprend l'exemple de la figure 2.5 pour lequel l'ordre causal n'était pas respecté. Initialement les vecteurs temporels de chaque membre sont $(0, 0, 0)$. Chaque composante du vecteur indique le nombre de message émis par l'un des membres (ici A, B, C , dans l'ordre). Lorsque A diffuse le message m_A , il l'accompagne de son horloge locale mise à jour $((1, 0, 0))$. Le site B reçoit m_A , et met à jour son horloge locale. Le prochain message m_B diffusé par B contiendra l'horloge $(1, 1, 0)$ indiquant que A et B ont émis un message. Quand C reçoit m_B , il établira, après comparaison avec son vecteur local, qu'il lui manque un message de A . Ainsi, m_B ne sera délivré par le site C qu'après réception et livraison de m_A .

L'ordre causal impose deux contraintes. Il faut tout d'abord que chaque membre connaisse la composition du groupe, ainsi qu'un ordre fixé à priori pour affecter une composante du vecteur à chaque membre du groupe. Ensuite, la taille du vecteur transmis et conservé localement dépend du nombre de membres du groupe. Si les groupes deviennent très grands, il est nécessaire de réduire les vecteurs [BSS91].

2.3.2.4.3 Respect de l'ordre par horloges matricielles :

Raynal et al. [RST91] proposent un algorithme utilisant des horloges matricielles pour assurer la livraison causale. Chaque site S_i maintient localement un vecteur d'entiers REC_i , de taille n (correspondant à la taille du groupe), et une matrice d'entiers $SENT_i$, de dimension $n \times n$. Le principe de l'algorithme consiste à ce qu'à tout message, émis par

un processus P_i (s'exécutant sur S_i), soit associée la matrice locale $SENT_i$ dans laquelle l'élément $SENT_i[k, j]$ représente, selon la perception de P_i , le nombre de message émis de P_k vers P_j . Malgré sa simplicité, cet algorithme présente deux inconvénient majeurs : la taille des messages transportés, et le nombre n^2 de comparaisons, entre paires d'entiers, nécessaires afin de décider de la délivrance du message reçu.

Protocole	Représentation de la causalité	Mode de défaillance	Degré de parallélisme	Détection-Recouvrement
Fast CBCAST	compteur & vecteur de compteur	processeur : arrêt (une défaillance par groupe) médiun : non défaillance	\leq nombre de membres	protocoles non intégrés Monitoring protocol Membership protocol
Raynal	vecteur de compteur	processeur : non défaillant médiun : non défaillant	\leq nombre de membres	pas d'algorithme

TAB. 2.2 - *Tableau récapitulatif des protocoles d'ordre causal*

2.3.2.5 Protocoles multipoint atomiques

2.3.2.5.1 Protocoles atomiques à contrôle décentralisé :

Dans ce type de protocoles, tous les sites du groupe sont responsables de la diffusion des messages, de la réception des acquittements correspondants et de la livraison totalement ordonnée. Cette technique distribuée laisse le contrôle de la numérotation à l'émetteur du message.

2.3.2.5.1.1 Protocole ABCAST :

Le protocole ABCAST (cf. 2.3.1.1) [BJ87b] résiste aux défaillances des sites et assure un ordre de réception uniforme par ordonnancement total des messages. Cet ordonnancement est réalisé au moyen d'estampilles. Un site qui reçoit un message (à destination d'un processus de ce site) le met dans une file d'attente locale et le marque comme "en attente". Il renvoie à l'émetteur un ACK estampillé par la date de réception du message. Lorsque l'émetteur a reçu tous les ACK pour un message, il affecte à ce message une estampille définitive qui est la plus grande des différentes estampilles des ACK du message. Cette estampille est rediffusée à tous les sites destinataires. Chaque destinataire affecte définitivement cette estampille au message correspondant en attente, marque celui ci comme "prêt" et réordonne la file dans l'ordre croissant des estampilles. Si le message de tête est dans l'état prêt, il est délivré aux processus destinataires sur le site. Cette méthode garantit que deux messages sont dans le même ordre dans toutes les files où ils se trouvent et assurent donc l'uniformité. Ce protocole utilise une validation à deux phases; il nécessite $3n$ messages pour une diffusion vers n sites, en l'absence de pannes.

En cas de panne de l'émetteur avant la fin de la rediffusion des estampilles, un des sites destinataires, soit p , (déterminé par élection) reprend le protocole et le mène jusqu'à terme. Pour cela, p interroge, pour chaque message marqué "en attente" dans sa file, les autres sites. Si un site possède le message à l'état "prêt", p transmet à tous les processus la valeur correspondante de l'estampille de ce site. Sinon, p rediffuse le message comme s'il n'avait jamais été transmis. Ce protocole nécessite de conserver les messages dans la file après leur remise à leur destinataire, et de les détruire ultérieurement par ramassage de miettes.

L'idée développée dans ABCAST a été reprise et améliorée dans le protocole ABP [MA91] qui respecte l'ordre total pour de groupes multiples tout en réduisant le nombre de phases (initialement deux phases) à une seule.

Les solutions liées à un contrôle décentralisé présentent deux inconvénients : d'une part, le coût en messages tend à être élevé et, d'autre part, le synchronisme requis pour qu'un processus soit informé de la stabilité du message qu'il a émis résulte en une diminution sensible de la concurrence dans les diffusions.

Protocole	Nombre de phases	Mode de défaillance	Degré de parallélisme	Partitionnement	Détection de défaillance
ABCAST	2 phases	processus: arrêt médium: omission	$\leq k.n$ $k=\text{constante}$	non toléré	algorithme View Manager

TAB. 2.3 - *Tableau de synthèse d'un protocole décentralisé*

2.3.2.5.2 Protocoles atomiques à contrôle centralisé :

La diffusion à contrôle centralisé repose sur l'existence d'un processus privilégié unique appelé *séquenceur* ou *coordonnateur*. Tous les messages du groupe transitent par le séquenceur qui, à son tour, les communique, selon leur ordre d'arrivée, aux autres membres du groupe. Il établit, ainsi, un ordre total de remise des messages.

La méthode de désignation du serveur d'ordre peut être statique ou dynamique. Dans le premier cas, le rôle de séquenceur est attribué à un processus du groupe pour toute la durée de la communication. Dans le second cas, ce rôle peut être interprété par n'importe quel processus du groupe, soit par élection, soit par la circulation d'un jeton. Cette dernière solution repose sur l'établissement d'un anneau virtuel entre certains ou tous les membres d'un groupe. Le serveur est, ainsi, désigné par un privilège associé à la détention du jeton. Le numéro d'ordre global définitif peut être associé aux messages dès leur première émission, si l'émetteur dispose d'un numéro de séquence globale par la circulation du jeton. Aussi, le serveur d'ordre peut varier au cours des diffusions (*exemple*: à chaque

diffusion ou bien toutes les k diffusions). La méthode choisie peut être rendue tolérante à une défaillance du séquenceur par l'élection d'un nouveau séquenceur parmi les membres du groupe.

2.3.2.5.2.1 Protocole RBP :

Chang et Maxemchuk [CM84] décrivent TRP (Reliable Broadcast Protocol) qui utilise un mécanisme centralisé destiné à réduire le coût de synchronisation nécessaire pour garantir l'ordre dans la délivrance des messages. Il utilise une combinaison des deux schémas d'acquiescement : positif et négatif pour présenter le service de livraison sous deux aspects. Pour les récepteurs, le service apparaît comme un système avec un site émetteur unique, appelé site jeton, qui sérialise le processus d'envoi des messages. De cette façon, le système est perçu comme un schéma à acquiescements négatifs par ces mêmes émetteurs qui, grâce à la numérotation séquentielle des messages, peuvent détecter les pertes de messages. D'autre part, le site jeton envoie un ACK contenant un numéro de séquence spécial appelé *timestamp* en réponse à tout message diffusé dans le groupe. De cette façon, le service est perçu, par tout émetteur de messages, comme un système à acquiescements positifs. L'émetteur sait, ainsi, que son message a été reçu par au moins un site du groupe (le site jeton en l'occurrence). Le serveur d'ordre est identifié par la détention d'un jeton circulant au sein d'un anneau. Etant donné que le coordonnateur peut changer dans le temps, le protocole est considéré par certains auteurs comme étant à contrôle décentralisé.

2.3.2.5.2.2 Protocole de Kaashoek et al. :

Le protocole décrit par Kaashoek et al. [KTH89] utilise un serveur d'ordre unique par lequel transitent les messages diffusés. Le séquenceur attribue un *numéro d'ordre* à chaque message reçu, conserve, dans un tampon historique, une copie de ce message séquencé puis le diffuse. Si tous les messages jusqu'à un numéro d'ordre k ont été acquiescés par tous leurs destinataires, ils peuvent être éliminés du tampon. Chaque destinataire accepte les messages dans l'ordre de leurs numéros et conserve le numéro s du dernier message reçu. Si un message arrive avec un numéro supérieur à $s + 1$, le récepteur le met en attente et demande au séquenceur de lui renvoyer une copie du ou des messages manquants. Si le tampon historique devient plein, le séquenceur entame une phase de synchronisation dans laquelle il s'assure que tous les messages présents dans le tampon ont bien été reçus par tous les destinataires. Il renvoie si nécessaire les messages perdus puis il vide le tampon.

Ce protocole vise avant tout l'efficacité, sur un réseau Ethernet dont le médium de communication supporte la diffusion, dans l'hypothèse où les diffusions sont fréquentes et où les messages diffusés sont brefs. Les acquiescements ne sont pas envoyés systématique-

ment mais sont transportés dans le prochain message du destinataire vers le récepteur (piggybacking).

Dans [BSS91], Birman et al. proposent une nouvelle implantation de la primitive ABCAST, nommée Fast ABCAST. Le protocole utilise la primitive de diffusion causale CBCAST pour fournir un ordre total conforme à l'ordre causal. Pour diffuser un message, tout émetteur le marque "indélivrable" puis le diffuse en utilisant CBCAST. Tout récepteur insère le message reçu dans sa file des CBCASTs et ne le retire pas même si tous les messages le précédant causalement sont délivrés. Seul le serveur d'ordre (désigné de façon statique) délivre les messages reçus. Il établit, ainsi, un ordre de livraison local qu'il mémorise dans une liste, appelée *sets-order*. Après un certain délai, ou après réception d'un nombre k fixé de messages, le serveur d'ordre diffuse cette liste en utilisant la primitive CBCAST (avant de recevoir la liste de livraison, un récepteur aura forcément reçu tous les messages de cette liste). A la réception de la liste, chaque membre peut, alors, livrer les messages en respectant l'ordre fixé par le serveur d'ordre (cet ordre est un ordre causal).

2.3.2.5.2.3 Protocole RMP :

Plus récemment, RMP [MWK95] se base sur le protocole TRP (Token Ring Protocol) [CM84] pour fournir une livraison totalement ordonnée dans le but d'assurer le synchronisme virtuel tel que défini dans le projet ISIS (cf 2.3.1.1). Le service de livraison ordonnée utilise une combinaison des deux techniques d'acquiescement (ACK et NAK). Les *listes à jeton* contiennent les listes des membres appartenant aux anneaux correspondants. Un site diffuse, en utilisant une adresse de groupe, ses messages vers tous les sites inscrits dans la liste à jeton choisie. Pour chaque message émis, un seul site (celui qui détient le jeton au moment de la réception) répond en envoyant un ACK. Ainsi, l'émetteur sait qu'au moins un site de la liste a reçu correctement le message. Pour permettre un ordonnancement total, les messages doivent être ordonnés à la source. Pour cela, l'ACK retourné par le site-jeton (*token site*) contient un numéro de séquence, appelé *timestamp* permettant un ordonnancement global pour tous les sites. Le token site doit conserver une copie de tous les messages acquittés afin de les retransmettre si nécessaire. A l'intérieur de l'anneau, l'acquiescement est négatif.

Les inconvénients majeurs de cette famille de protocole sont leur faible tolérance aux défaillances ainsi que les phénomènes de congestion et/ou d'implosion (appelés aussi *goulots d'étranglement*) qu'ils ont tendance à provoquer du fait de la concentration des messages diffusés ainsi que des acquiescements au niveau du site séquenceur. De plus, le protocole proposé par Kaashoek et al. [KTH89] présente un inconvénient supplémentaire, à savoir la taille importante des buffers nécessaires au stockage des messages réceptionnés.

Protocole	Nombre de phases	Mode de défaillance	Degré de parallélisme	Partitionnement	Détection de défaillance
RBP	$(n + L)$ passages de jeton $n =$ nombre de membres $L =$ paramètre défini	processus: arrêt médium: omission	$\leq n$	non toléré	protocole non décrit
Kaashoek et al.	2 phases	processus: omission médium: omission	n en phase 1 1 en phase 2	non toléré	protocole non décrit
Fast ABCAST	2 phases	processus: arrêt (1 défaillance/groupe) médium: fiable	$\leq n$	non toléré	Protocoles autonomes: Monitoring protocol Group view protocol
RMP	$(n + L)$ passages de jeton	processus: arrêt médium: omission	$\leq n$	K-résilience & Livraison atomique	Timeouts-algorithm

TAB. 2.4 - Tableau récapitulatif des protocoles centralisés

2.3.2.5.3 Protocoles basés sur la construction d'arbres de diffusion :

Plusieurs protocoles multipoint sont basés sur la construction d'arbre de diffusion [Ram90] [GS91]. Ces derniers sont, généralement, dédiés aux réseaux de type point-à-point (qui ne supportent pas la diffusion au niveau physique). L'inconvénient majeur de ce type de diffusion est que le temps nécessaire à une diffusion est proportionnelle à la hauteur de l'arbre qui, elle, est inversement proportionnelle à la probabilité d'apparition de goulots d'étranglement. En d'autres termes, il n'est possible d'améliorer le temps de diffusion qu'en augmentant le risque d'apparition de goulots d'étranglement.

Dans [GS91], Garcia-Molina décrit un protocole qui construit un “*graphe de propagation*” qui est une arborescence formée à partir des différents noeuds du groupe. Il fait transiter, par des sous-arbres, les messages diffusés, de façon à ce que l'ordre soit assuré par plusieurs sites. Le graphe construit détermine le chemin à suivre par chaque message pour atteindre ses destinataires. L'algorithme décrit comporte deux phases : dans la première phase, le graphe de propagation est construit à partir des groupes de processus définis. Ce graphe est ensuite utilisé dans la seconde phase par un protocole de diffusion. Il est supposé qu'un seul site construit le graphe de propagation et le communique ensuite aux autres membres.

Pour diffuser un message m , le processus source l'adresse à un site particulier du groupe appelé destination primaire. Chaque site maintient un numéro de séquence pour chacun des sites auxquels il émet des messages (en respectant le graphe de propagation). Ce numéro permet aux récepteurs de détecter les éventuels cas de perte ou de déséquence de messages. Deux files de messages existent au niveau de chaque site : la première pour les messages à délivrer aux processus locaux, la seconde pour les messages arrivant avant d'autres les précédant. L'acheminement des message se déroule de la manière suivant : lorsqu'un site reçoit un message dont il n'est pas destinataire, il détermine si un ou plusieurs de ses descendants en est (sont) destinataire(s). Si c'est le cas, il réémet le

message, accompagné du numéro de séquence correspondant, à tous ses fils qui sont racines de sous-arbres contenant les destinataires déterminés.

2.4 Proposition d'un protocole multipoint flexible

2.4.1 Qualité de service

Un nombre important de protocoles de communication destinés à satisfaire divers besoins des applications et diverses contraintes inhérentes aux réseaux de communication, a été proposé. La variété des applications pour lesquelles les protocoles étudiés ont été réalisés, entraîne, pour une large part, une divergence entre les sémantiques de fiabilité utilisées, mais aussi, dans les considérations techniques qui guident le développement de ces protocoles. A titre d'exemple, SRM supporte des sessions multipoint fiables impliquant un très grand nombre de participants dans des applications multimédia interactives sensibles aux délais de transfert. Pour cela, SRM réduit les informations de retour (feedbacks) en utilisant les acquittements négatifs. Seulement, cette approche, basée sur les NACKs, combinée à l'absence d'un mécanisme d'adhésion/retrait du groupe, ne garantit pas qu'un utilisateur particulier reçoive une donnée diffusée. En effet, si une entité diffuse une donnée puis quitte le groupe, une entité qui n'a pas reçu la donnée diffusée ne pourra pas la recouvrer puisque son émetteur ne diffusera plus de données. On peut prévoir un protocole de gestion de groupe qui impose à une entité quittant le groupe de s'assurer que tous ces messages ont été correctement reçus par les membres du groupe de diffusion. D'autre part, RMTP et TMTP utilisent des serveurs hiérarchisés adaptés aux applications de dissémination de données (telles que les News) qui sont délivrées dans leur intégralité, sinon le transfert échoue.

Etant donnée la variété des besoins et des sémantiques de services nécessaires aux applications, la conception d'un protocole multipoint satisfaisant un large ensemble d'applications paraît impossible sans l'introduction de la notion de *qualité de service*. La qualité de service définit un ensemble de paramètres associés à une connexion entre l'entité applicative et l'entité transport locale qui fournit le service multipoint. On trouve généralement deux classes de paramètres : les paramètres *fonctionnels* et les paramètres de *performance*. Les paramètres fonctionnels correspondent au niveau application et font référence aux mécanismes de fiabilité, d'ordre, de synchronisation, etc. Les paramètres de performance permettent la quantification de la qualité de service souhaitée par l'utilisateur comme le temps de réponse (latence), le débit, le taux d'erreurs, etc.

Nous nous intéressons dans notre étude à la première classe de paramètres et plus particu-

lèrement aux services de fiabilité et d'ordre destinés aux applications de groupe réparties sur des réseaux étendus. L'introduction de la qualité de service sur les services fournis, confère au protocole la propriété de *flexibilité* qui permet de répondre aux besoins variés des applications, sans pour autant pénaliser les applications qui ne requièrent qu'une partie ou aucun des services fournis. De plus, les services proposés peuvent co-exister au sein d'une même application. En effet, au cours d'une même session, un utilisateur peut requérir plusieurs types de service en fonction des contraintes de livraison qu'il désire imposer sur le paquet remis à l'entité de transport locale. Cela peut être rendu possible en appliquant la qualité de service requise sur chaque paquet diffusé. Aussi, les mécanismes internes utilisés par les entités de niveau transport, telle que la gestion de groupe, requièrent un service d'ordre total que le niveau utilisateur peut ne pas requérir. Pour ces différentes raisons, mais aussi pour d'autres, que le tableau ci-dessous présente, nous proposons un protocole multipoint flexible, nommé FMP (*Flexible Multicast Protocol*) [Sia97], qui offre différentes qualités de fiabilité et d'ordre.

Service multipoint	Qualité de service requise	Niveau d'implantation
Contrôle d'erreur	Différents niveaux de sémantique peuvent être supportés à travers les champs réservés des paquets	P/S
Détection d'erreurs	à la charge des émetteurs (ACK) ou à la charge des récepteurs (NACK) ou solution mixte	S
Requêtes	mode de communication : point-à-point vers le site source ou multipoint	P
Retransmissions	retransmissions point-à-point ou multipoint assurées par le site source ou par un site quelconque	P
Gestion de groupe	protocole d'adhésion/retrait, ou aucune connaissance sur la composition du groupe	S
Structure de groupe	émetteurs et/ou récepteurs hiérarchisés ou non	S
Ordre	remise non ordonnée, ordonnée pour une source unique, partiellement ou totalement ordonnée pour des sources multiples	P
Dans la dernière colonne, la lettre P signifie : service sélectionnable sur la base de paquets, alors que S signifie : à l'établissement de la session		

TAB. 2.5 - *Besoins des applications en types de service*

2.4.1.1 Fiabilité

La sémantique de fiabilité adoptée par une application est étroitement liée aux besoins de celle-ci. Certaines applications requièrent uniquement que les données soient remises à tous les membres participants. D'autres imposent une remise fiable, ordonnée selon la source. D'autres, encore, n'exigent une remise fiable que vers une partie (majoritaire ou

pas) des membres du groupe.

L'approche que nous avons adoptée permet de concevoir différents services séparés, chacun optimisant un scénario d'utilisation. En adoptant un schéma d'acquiescement négatif, relayé du mécanisme de temporisation et d'échantillonnage (slotting/damping) permettant le contrôle de flux, notre solution fait une synthèse de deux propositions existantes : d'une part, celle adoptée par SRM [FJM95] qui repose sur le principe d'une architecture minimal d'accueil d'applications *Application Level Framing*, et d'autre part, celle adoptée par RMP [RST91] qui fournit une plateforme offrant divers services multipoints sélectionnables sur la base de chaque paquet diffusé. L'intérêt est de fournir une plateforme de communication multipoint flexible qui offre des services variés sélectionnables par un ensemble large d'applications.

2.4.1.2 Ordre

Les protocoles multipoint assurant les propriétés d'ordre causal ou d'atomicité sont des outils puissants qui simplifient de manière importante le développement d'applications dans les environnements distribués. Toutefois, le coût généralement élevé, nécessaire à la mise en oeuvre de ces protocoles a suscité une controverse "ardente" parmi la communauté de chercheurs dans le domaine. Ainsi, certains travaux [CS93] [Coo94] tendent à établir que les propriétés de causalité et d'atomicité sont souvent indésirables ou, du moins, de "petite" utilité comparativement à leur coût. D'autres travaux [Bir94] [Ren94] estiment, au contraire, que ces propriétés offrent aux applications un environnement de développement essentiel difficilement incontournable.

Même si l'utilité des primitives assurant l'ordre total et/ou causal n'est pas unanimement admise, leur intérêt n'est pas pour autant réduit. Les raisons du rejet sont liées aux implémentations coûteuses et peu performantes des protocoles, et non pas aux propriétés d'ordre elles même. De nombreux protocoles de transport ont, depuis, été développés. Bien qu'ils assurent une distribution totalement ordonnée, leurs performances restent satisfaisantes [MWK95] [BOG⁺94] [RBM96]. Pour notre part, nous proposons deux modules d'ordonnement total et causal. Le premier fournit la propriété de synchronisme virtuel utile pour la mise à jour des vues du groupe. Le second est une alternative moins coûteuse à l'ordre total ; il n'induit pas le degré de synchronisme élevé inhérent aux protocoles atomiques. Les deux modules d'ordre utilisent un module multipoint fiable sous-jacent. Pour ne pas pénaliser les applications nécessitant uniquement un service fiable, le fonctionnement du dernier service doit rester indépendant de celui des deux premiers. Comme pour le service fiable, la réalisation des deux services d'ordre accorde une priorité maximale aux performances de chacun des services en terme de limitation du flux de données

et de paquets de contrôle.

2.4.1.3 Services fournis

FMP fournit les services de fiabilité et d'ordre suivants, sélectionnables soit à l'établissement d'une session multipoint, soit sur la base de chaque paquet diffusé :

- fiabilité modulable : de la diffusion non fiable (de type “best-effort”) à la fiabilité absolue exigeant une remise fiable, selon l'ordre d'émission, à tous les membres. Entre les deux, se déclinent les propriétés de fiabilité majoritaire (k -fiabilité, k représentant un nombre majoritaire de sites) et de fiabilité absolue avec remise sans respect de l'ordre d'émission,
- deux modules adjacents au dessus du module de fiabilité absolue, réalisant les propriétés d'ordre causal et total. L'ordre causal est une alternative moins coûteuse à l'ordre total. Il peut, effectivement, offrir les propriétés de l'ordre total sans pour autant induire le degré de synchronisme souvent élevé inhérent aux protocoles atomiques. Par ailleurs l'ordre total permet au protocole de garantir un modèle d'exécution virtuellement synchrone utilisable notamment par tout protocole d'adhésion/retrait implémenté au dessus du module de communication atomique.

Chapitre 3

Conception de FMP : Protocole Multipoint Flexible

3.1 Introduction

Notre travail a comme objectif, la conception de FMP (Flexible Multicast Protocol), un protocole multicast flexible, puis son implantation en utilisant un langage de spécification formelle (Estelle [ISO89]), dans un effort de maintien d'un modèle de représentation proche des fonctionnalités spécifiées. Dans ce chapitre, nous décrivons les services de diffusion fiable, totalement ordonné et causalement ordonné, implémentés dans le cadre de FMP dont les motivations ont été énoncées dans le chapitre précédent. Le modèle adopté dans FMP est décrit dans la section 2. Les services fournis par FMP sont présentés dans la section 3. La section 4 est consacrée à la description du service de livraison fiable. Nous poursuivons, dans la section 5, la présentation des fonctionnalités de FMP, en décrivant le service de livraison totalement ordonné (ou diffusion atomique). La section 6 décrit le dernier service implanté dans cadre de FMP, qui est le service de livraison causale. Dans les trois dernières sections, l'étude porte, en premier, sur le fonctionnement des services décrits, en l'absence de défaillances, puis sur le comportement des processus émetteurs et récepteurs dans un environnement défaillant.

3.2 Modèle

FMP est basé sur un modèle d'interaction entre des processus opérant sur des sites interconnectés par un réseau de communication étendu. Nous allons présenter les entités qui constituent l'architecture du système et qui sont impliquées dans les services fournis par le protocole. Nous allons aussi décrire comment nous avons ajusté le modèle pour accomplir

les propriétés souhaitées.

3.3 Architecture du système

Le niveau utilisateur du protocole multipoints FMP est appelé processus de l'application. Le fournisseur de service local est appelé processus de communication. Les processus de communication qui implémentent le protocole FMP sont appelés processus FMP ou entités FMP. Ils sont interconnectés par un réseau étendu supportant la communication de groupe. Le service multipoints fiable, atomique ou causal est fourni par l'interaction des processus FMP du groupe. Relativement à une invocation du service FMP fourni, un processus membre du groupe joue soit le rôle de l'émetteur, soit le rôle du récepteur. Au cours d'une session de groupe, chaque membre peut, indifféremment, avoir le rôle d'émetteur ou de récepteur.

Les processus FMP évoluent dans un environnement asynchrone. En conséquence, ils ne partagent pas de mémoire commune et communiquent par envoi de messages. Chaque processus FMP est associé à un noeud, appelé aussi site ou processeur, du réseau qui supporte l'exécution d'un ou plusieurs processus de l'application de groupe. De plus chaque processus FMP a accès à une horloge locale qui n'est pas synchronisée avec les horloges des autres noeuds du groupe. Les délais de transmission sont supposés non bornés, dans le sens où la terminaison du transfert d'un message n'est pas déterminée au préalable par son émetteur. Le protocole FMP repose sur un service de transport sans connexion et non fiable et un service réseau qui assure la communication par diffusion.

3.4 Description des services fournis par FMP

3.4.1 Définitions relatives aux services fournis

Nous donnons quelques définitions sur lesquels sont basés les mécanismes qui implémentent les différents services de FMP.

- **Consensus unanime** : si un processus correct diffuse un message alors soit tous les processus corrects délivrent le message, soit aucun d'entre eux ne le délivre.
- **Terminaison** : un protocole de diffusion possède la propriété de terminaison si chaque processus correct connaît l'issue de la diffusion en un temps fini.

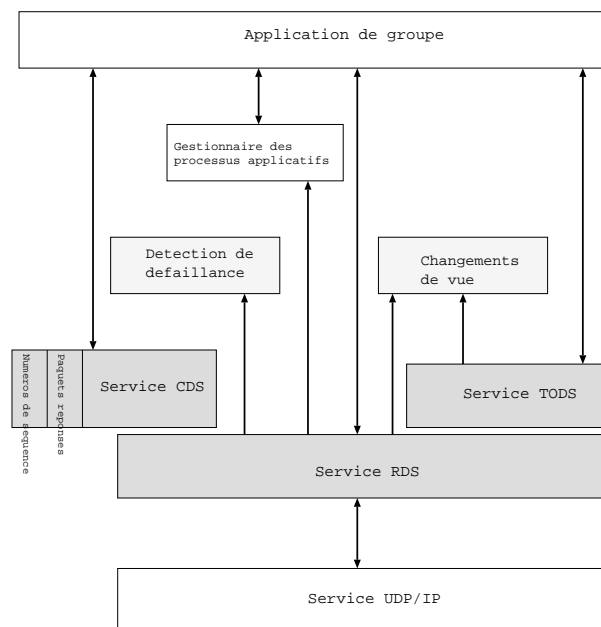


FIG. 3.1 - Architecture du protocole FMP

- **Ordre total** : en présence d'un seul groupe de diffusion, tous les processus corrects du groupe remettent les messages dans le même ordre.
- **ordre causal** : en présence d'un seul groupe de diffusion, la remise des messages respecte l'ordre causal, mais on ne peut rien dire sur l'ordre de livraison des messages non liés causalement.

Le service de livraison fiable (CDS : *Reliable Delivery Service*) garantit les deux propriétés caractéristiques de consensus unanime et de terminaison.

Le service de livraison atomique (TODS : *Total Ordered Delivery Service*) garantit les trois propriétés de consensus, de terminaison et de d'ordre total.

Le service de diffusion causale (*Causal Delidery Service*) garantit les trois propriétés de consensus unanime, de terminaison et d'ordre causal. Notons qu'aucun des protocoles de diffusion causale, étudiés dans le chapitre 2, ne garantit la propriété de terminaison en présence de défaillances.

3.4.2 Hypothèses de fonctionnement de FMP

- **Service de communication** : FMP est conçu au dessus d'une couche de transport et d'une couche réseau qui fournissent un service de diffusion non fiable et non ordonné (avec meilleur effort) qui possède les deux caractéristiques suivantes : si un message est altéré lors de sa transmission, il est ignoré par ses récepteurs. D'autre

part, il n'assure pas le séquençement des messages provenant d'un même émetteur. Ce service est présent aussi bien dans les réseaux locaux que dans une d'interconnexion de réseaux, tel que Internet, sous forme d'une fonction de transport (assurée par la couche UDP) et d'une fonction de routage (assurée par la couche IP comprenant le service Multicast IP). La couche IP définit des mécanismes d'adressage et de gestion de groupe. Les messages sont diffusés vers des groupes désignés par des adresses IP sans aucune connaissance préalable sur les membres des différents groupes. Pour recevoir un message diffusé au groupe, un processus doit simplement annoncer son intention via un message d'adhésion envoyé au routeur du réseau local auquel son site appartient. Tout processus peut joindre ou quitter le groupe sans que cela n'affecte les interactions entre les autres membres du groupe.

- **Modèle de groupe** : Les services FMP sont assurés pour un groupe ouvert, dynamique et indéfini. L'émission des messages concerne aussi bien les processus membres que les non membres du groupe. La composition réelle d'un groupe n'est connue par aucun de ses membres.
- **Données des utilisateurs** : Les données utilisateurs transmises sont de tailles variables mais bornées. La taille maximale est fixée par le processus application lors de la négociation consécutive à sa première invocation du service FMP. Pour chaque message diffusé, le niveau utilisateur transmet au protocole les paramètres qui indiquent la qualité de service, tels que la fiabilité, l'ordre, le débit souhaité, que le service invoqué devra assurer.
- **Application cibles** : Les applications qui nécessitent les services de diffusion fiable sont nombreuses. Deux modèles d'application regroupent une grande partie d'entre elles : le modèle "producteur-consommateur" (publisher-subscriber) et le modèle "client-serveur". Le premier modèle permet à un processus d'adhérer ou de "s'inscrire" à un groupe et de consommer les messages publiés. Il ne requiert pas une identification explicite de la destination des messages. Il s'adapte bien aux applications distribuées dont les entités sont mobiles, tels que les systèmes multi-agents. Ce modèle regroupe, aussi, les applications multimédia temps réel (systèmes de téléconférence) et les applications de conception collaborative (éditeurs coopératifs). Le modèle client-serveur constitue une architecture plus classique, organisée autour de processus serveurs qui réceptionnent puis traitent requêtes émises par des processus clients qui restent en attente des réponses retournées par un des serveurs. Souvent, la désignation du groupe de serveurs ou de clients est explicite lors de la diffusion des requêtes ou des réponses. Les applications de recherche d'informations ou de localisation de services sont des exemples d'applications de ce modèle. Deux autres modèles d'application sont également utilisés : le modèle des "groupes de diffusion"

et le modèle de "groupe hiérarchique". Le premier décompose le groupe en deux ensembles : l'ensemble des émetteurs, constitué par les processus actifs, et l'ensemble des processus récepteurs, constitué par les processus passifs. Les services de News et de DNS (Domain Name Service) sont un exemple d'applications de ce modèle. Le second consiste en de multiples groupes composants. Chaque groupe composant, ainsi que le support d'interconnexion des groupes peuvent utiliser d'autres modèles de communication. Cette approche est pratique pour les applications où l'extensibilité et la sécurité sont les préoccupations principales. Citons également les applications liées aux systèmes répartis (calcul réparti, gestion d'informations en copies dupliquées, synchronisation d'horloges, etc.) qui peuvent tirer parti des services de FMP, notamment, les services de livraison totale et causale.

- **Mod de défaillance** : Le mode de défaillance adopté concerne aussi bien les sites FMP que réseau d'interconnexion¹. En ce qui concerne les sites, deux cas sont considérés :
 - les défaillances par arrêt : un processus qui fonctionnait correctement s'arrête définitivement, provoquant le départ du site du groupe,
 - les défaillances par omission : dues, soit à des problèmes de tampons pleins, soit à des rejets suite à des erreurs de transmission. Le protocole tolère à la fois les omissions en émission et en réception.

Les défaillances du réseau de communication peuvent conduire à un partitionnement du réseau, toléré également par le protocole. Elles sont du mode défaillances par omission.

La défaillance d'un message peut résulter de la saturation du tampon de l'émetteur, ou de celui d'un récepteur. Un nombre arbitraire de messages peuvent être perdus mais lorsqu'un message est traité par un site, il n'a subi aucune altération. Une défaillance est détectée lorsqu'un site ne peut plus communiquer avec un autre site du groupe après un certain nombre de tentatives. Ce dernier site est, alors considéré comme non opérationnel. Le nombre de tentatives devra être suffisamment grand pour éviter qu'un site ne soit considéré, à tort, comme défaillant, et suffisamment petit pour une détection rapide des défaillances.

– Détection des défaillances

Dans les protocoles à acquittements positifs, une défaillance est détectée lorsque la communication entre le groupe et un site échoue après un certain nombre d'essais. Après chaque expiration du *timeout*, un émetteur retransmet le message jusqu'à

1. Etant donné les réseaux actuels, cela est rare (\ll 1% des paquets).

la réception de l'ACK correspondant. Par opposition, FMP, qui utilise un schéma d'acquiescement négatif, retransmet les NACKs un nombre de fois suffisant pour que les messages perdus soient réceptionnés. Ainsi, toute perte de messages génère une forme d'acquiescement positif. Si cet acquiescement n'est pas reçu dans une période de temps déterminée², le site qui ne répond pas est considéré comme défaillant.

FMP n'intègre pas de mécanisme de recouvrement de défaillances. Ce mécanisme peut être implémenté au dessus du service FMP.

- **Contrôle de flux et de congestion** : le contrôle de flux est effectué sur les messages de contrôle tels que ACK et/ou NACK pour éviter l'effet d'implosion de ACKs ou de NACKs au niveau d'un site FMP.

Le contrôle de congestion est effectué sur les messages de données pour éviter qu'un émetteur ne congestionne le tampon de réception d'un site receveur, provoquant ainsi une perte de paquets. L'émetteur utilise généralement un mécanisme de fenêtre globale pour adapter sa vitesse de transmission à celle de réception des sites destinataires. L'étude réalisée n'a pas abordé ce dernier aspect de régulation de flux.

3.4.3 Fonctionnalités de FMP

FMP fournit un mécanisme de transport par l'intermédiaire duquel un utilisateur peut concevoir et implémenter des applications distribuées fiables et cohérentes, sans se soucier des primitives de communication de plus bas niveaux. Dans cet objectif, FMP fournit les fonctionnalités suivantes :

- fiabilité,
- respect des propriétés d'ordre total,
- respect de l'ordre causal,
- contrôle de flux,
- synchronisme virtuel, assuré grâce au service d'ordre total,
- support du modèle de groupe de processus,
- flexibilité dans le choix des services fournis,
- protocole multi-thread permettant à un message de requête de ne concerner qu'un seul site source à la fois.

2. voir l'étude réalisée dans (3.5.3.3)

Notre approche dans la conception de FMP a été de prototyper le protocole en adoptant pour cela une démarche pratique sans pour autant être industrielle. Les propriétés liées aux performances de débits et aux délais de communication dans les conditions de fonctionnement normal du protocole, n'ont, donc, pas été notre priorité principale dans la spécification et la réalisation du protocole. Cependant, lors des différents choix de conception, notre souci majeur a été de concilier au maximum la fiabilité et la cohérence des services fournis, avec l'aspect contrôle de flux et de congestion du protocole.

3.4.3.1 Modèle de transfert de données

Tous les processus de l'application reçoivent toutes les données émises par un processus du groupe et ce, dans un ordre différent, un ordre partiel ou un ordre global. Le modèle de flux de données entre processus applications est un modèle de communication de type *multi-producteurs/multi-consommateurs*. Chaque processus produit et met ses données dans sa file locale d'émission. Le service de communication FMP extrait les données mises dans les files d'émissions des processus locaux et les dépose dans une file globale unique au site. La donnée en tête de file est dupliquée et déposée dans la file de réception de chaque processus local, puis diffusée à travers le média de communication multipoints.

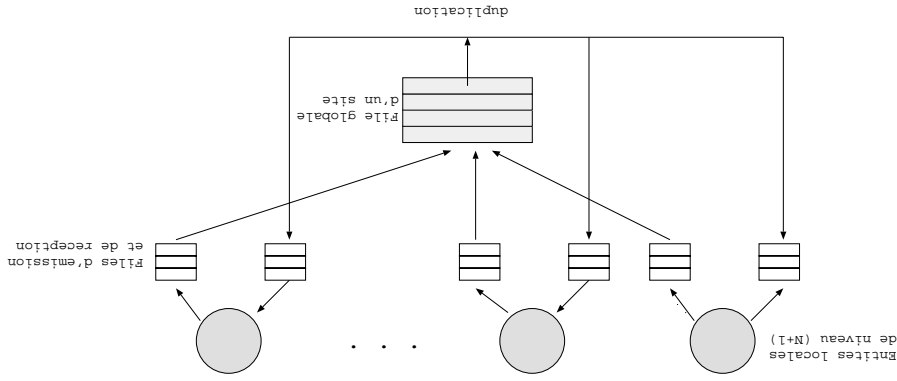


FIG. 3.2 - *modèle multi-producteurs/multi-consommateurs*

3.4.3.2 Services sélectionnables

Différents types d'applications exigent différents niveaux de fiabilité et d'ordre. Pour supporter ces différents niveaux, FMP adapte sa politique de livraison des messages aux besoins du niveau utilisateur et la rend flexible. Pour permettre une utilisation flexible des services fournis, chaque message comporte un champ indiquant le niveau de qualité

de service devant être assuré par le protocole lors de la livraison. Chaque message est livré selon la qualité de service (QoS) sélectionnée par le processus émetteur. La qualité de service est associée à un message depuis son émission jusqu'à sa destruction de toutes les files de réception des sites du groupe. Les qualités de service fournies par FMP sont : diffusion non fiable, fiable, totalement ordonnée et causalement ordonnée.

Niveau de QoS	Description
Non fiable	délivrance à la réception. Le site émetteur n'assigne pas de numéro de séquence à ce type de message. Si ce message est perdu, il ne sera pas requis par la suite
Fiable	délivrance à la réception. Un message perdu est requis. Un numéro de séquence est assigné à ce type de message
Fiabilité majoritaire	délivrance une fois que la majorité des sites ont correctement reçu le message
Ordonné à la source	délivrance à retardée jusqu'à la réception des paquets manquants originaires du même émetteur
Totalement ordonné	délivrance retardée jusqu'à ce que tous les messages de même type en transit, dont les estampilles sont inférieurs ont été livrés
Causalement ordonné	délivrance respecte l'ordre causal d'émission à travers le groupe

TAB. 3.1 - *Tableau des niveaux de QoS*

3.4.3.3 Etablissement d'une connexion multipoints

Pour joindre une session en cours, un nouveau processus de niveau application demande au processus FMP local, l'établissent d'une connexion multipoints. Il utilise pour cela une primitive de demande qui comporte les paramètres suivants :

- adresse individuelle du processus demandeur de la connexion,
- adresse du groupe destinataire des données diffusées par le nouveau processus,
- paramètres de connexion qui spécifient les qualités de service demandées, telles que :
 - la taille maximale des données produites par le processus,
 - les débits maximaux et minimaux acceptés par le processus : nombre d'octets transmis par le service FMP par unité de temps,
 - le délai de latence : délai maximum entre la prise en compte, par le service FMP local, d'une primitive de service et la confirmation, au processeur demandeur, de son exécution,
 - le nombre maximum de pertes successives tolérées,

- le type de service de transfert : non confirmé (NACK) par défaut.

Les données échangées au préalable ne seront pas communiquées, par le service FMP, au processus qui rejoint une connexion établie.

3.4.3.4 Changement de vue et synchronisme virtuel

Dans les groupes ouverts, un changement de vue correspond, pour un membre donné, à sa nouvelle vue de groupe consécutive à la mise à jour de la liste des membres, suite à la réception par le membre d'une information de départ ou d'arrivée d'autres membres.

Un membre peut détenir un *clé* qui permet d'implanter un mécanisme d'exclusion mutuelle entre sites. Une autre information, associée aux clés, concerne les *verrous*. Un verrou marquant un message diffusé, représente un mécanisme qui permet au message de n'être traité que par un site privilégié. La sémantique associée à un verrou est totalement liée au rôle que le service de diffusion lui attribue.

Un changement de vue peut être géré de manière efficace en considérant cet événement comme un changement dans l'état global du groupe. Ce changement peut être, ainsi, accompli par un message de QoS atomique. De cette façon les changements de vues sont sérialisés, ce qui permet d'assurer la propriété de synchronisme virtuel à travers le groupe. Prenons comme exemple, un site se retirant du groupe qui diffuse une requête de changement de vue, comportant un verrou. Cette requête est reçue par tout le groupe, mais seul un site privilégié, détenant la clé correspondant au verrou, la traite. Ce dernier établit un ordre entre les messages donnée de QoS atomique et les autres requêtes de changement de vue. Ainsi, chaque requête de changement de vue est estampillée d'un numéro de séquence global. Le site séquenceur diffuse, ensuite, une nouvelle configuration du groupe qui sera enregistrée dans le même ordre au niveau de chaque site. Pour assurer le synchronisme virtuel, à chaque requête doit correspondre une nouvelle vue de groupe, diffusée par le site jeton dans leur ordre de réception. Une solution où le site jeton ferait la synthèse des requêtes puis renverrait une nouvelle liste à l'ensemble des site, violerait, en effet, la propriété de synchronisme, puisqu'entre deux requêtes de changement de vue, des messages de QoS atomique peuvent être reçus par le site séquenceur.

3.4.3.4.0.1 Aspects de changement de vue :

Certains types d'applications peuvent nécessiter une politique de gestion de groupe qui ne valide le départ d'un site qu'une fois que celui-ci s'est assuré qu'aucun membre n'est en attente, actuelle ou à venir, de paquets qu'il détient. Cette précaution peut être compensée

par le fait qu'un paquet perdu n'est pas forcément retransmis par son site source, mais par un site aléatoire du groupe qui le détient. Le départ d'un site n'a d'incidence sur le mécanisme de recouvrement que si ce site a subi une défaillance par omission lors de l'émission du message requis.

Quand ce site diffuse une requête de changement de vue signalant son départ, il continue de fonctionner de façon normale, tant qu'il n'a pas reçu une nouvelle liste qui confirme sa suppression du groupe. Aussi, si le site en question est le site jeton, il continue de fonctionner jusqu'à ce qu'il ait passé le rôle de site séquenceur à un nouveau site élu. A la réception de la nouvelle liste et selon les besoins des applications, le site s'assure qu'aucun site ne demande ou ne pourrait demander un paquet qu'il détient. Le site positionne, par la suite, une horloge de départ dont l'expiration le fait définitivement quitter le groupe. L'horloge permet d'éviter qu'une défaillance du réseau de communication ne bloque indéfiniment le site partant.

3.4.4 Description des unités du protocole FMP

Les données transmises par un processus de l'application sont appelées unités de données de service (SDU). Elles ont une taille maximum négociée à la demande d'établissement de la connexion. Un SDU est, éventuellement, décomposé en plusieurs paquets consécutifs qui sont les unités de données du protocole (PDU). Un PDU constitue l'entité que manipule le service de transport FMP (le niveau transport niveau N). Chaque PDU est transporté vers les entités réceptrices analogues au processus FMP émetteur.

3.4.4.1 Format général des paquets

Chaque PDU se décompose en une en-tête, contenant les informations de contrôle du protocole, suivie d'une partie contenant les données de l'application. Le format général des PDUs comporte les champs suivants :

- *Identification du protocole* : ce champ comporte le numéro et la version du protocole.
- *Numéro de séquence* : permet l'identification d'un PDU, mais aussi à chaque site FMP d'effectuer les principales opérations de service du protocole (maintien en séquence, détection de perte, livraison, etc.).
- *Adresse source* : adresse IP du site FMP source du PDU diffusée. En cas de rediffusion, c'est l'adresse du site FMP qui retransmet qui est insérée, et non l'adresse du site source. Notons qu'il n'est pas conceptuellement utile d'indiquer l'adresse du

processus applicatif émetteur. Il suffit que chaque site FMP maintienne un compteur initialisé à 0 et correspondant au nombre de processus utilisateurs locaux. Ce compteur est actualisé localement de la manière suivante : il est incrémenté de 1 à chaque fois qu'un nouveau processus demande son adhésion au groupe et que celle-ci est acceptée (selon un protocole d'adhésion niveau $N + 1$). Le compteur est décrémenté de un lorsqu'un processus demande son retrait du groupe. Lorsque le compteur atteint la valeur nulle, le site FMP signale son retrait du groupe et ne traitera plus les messages diffusés, jusqu'à ce qu'un nouveau processus utilisateur local vient se joindre au groupe.

- *Adresse de retransmission*: utilisée par le service de livraison fiable. En cas de retransmission, ce champ indique le site source du PDU retransmis.
- *Adresse destination*: il s'agit de l'adresse du groupe d'entités FMP destinataires du PDU. Une fois le PDU correctement réceptionné, chaque entité dépose le paquet dans les files de réception des processus locaux.
- *Type de PDU*: les différents types de PDUs manipulés par FMP, sont décrit plus bas.
- *Crédit de la fenêtre*: fenêtre maximale permettant de borner le flux et le débit. Le contrôle de débit permet de partager la bande passante du média de diffusion, selon des mécanismes établis entre les sites du groupe que nous n'aborderons pas dans cette étude.
- *Champs optionnels*: tel que le champ d'EM ou verrou disponible sur certains PDUs. Son but est de donner des privilèges au seul site, détenant la clé, auquel sont destinés ces PDUs.
- *Champ données*: qui contient les données utilisateurs diffusées.

3.4.4.2 Types des paquets

- **PDU donnée**: véhicule les données utilisateur. Ce PDU est numéroté. Il peut comporter des champs optionnels lorsque le PDU est destiné au site séquenceur chargé d'ordonner les messages de QoS atomique.
- **PDU d'EM**: utilisé pour l'élection d'un nouveau site séquenceur en cas de défaillance du site séquenceur (ou de choix stratégique dicté par le gestionnaire de niveau $N + 1$). Notons que le processus gestionnaire est celui qui a demandé auprès

En-tête fixe du protocole FMP			
Version	Longueur des champs d'option	Type de paquet	Options
En-tête du paquet diffusé			
Données transportées			

En-tête d'un paquet Donnée					
Verrou (Optionnel)	QoS	Site émetteur		Numéro de séquence	Longueur du champ donnée
		Adresse IP	Port UDP		

En-tête d'un paquet NACK		
Site émetteur	Site source des paquets requis	liste (non séquentielle) des paquets requis

TAB. 3.2 - *Format de deux PDUs du protocole FMP*

du processus FMP local l'établissement d'une connexion multipoints lorsque celle-ci n'était pas active.

De plus, les privilèges attribués par ce champ concernent le site séquenceur qui est le seul le site FMP possédant la clé. Il réalise les fonctions suivantes :

- Numérotation : il attribue un numéro de séquence global au PDU reçu, pour un ordonnancement respectant l'ordre total à travers le groupe.
- Mise à jour de la vue de groupe : il a aussi la responsabilité d'établir les PDU de changement de vue auxquels il attribue un numéro d'ordre global.

Le champ d'EM accompagne les PDU de données, les PDU de retransmission ainsi que les PDU de requête de changement de vue.

- **PDU d'adhésion** : construit par le site FMP qui reçoit une demande d'adhésion de la part d'une entité (N+1) locale. Selon le protocole d'adhésion choisit par l'application, la demande d'adhésion peut être uniquement notifiée aux entités de niveau $N + 1$ membres du groupe, ou bien soumise à l'acceptation d'une partie ou de la totalité de ses entités membres. Le protocole d'adhésion relève donc du rôle du niveau supérieur, FMP se charge uniquement de lui fournir le support de diffusion souhaité.
- **PDU d'accord et de refus** : ce PDU est utilisé par les entités de l'application pour signaler un accord ou un refus, lorsqu'elles sont consultées pour une demande d'adhésion d'une entité analogue. Le gestionnaire du groupe collecte et analyse les

réponses reçues, selon des critères propres à l'application, puis notifie un PDU d'accord ou de refus diffusé ou transmis directement par le processus FMP local. S'il s'agit du premier processus associé, l'entité FMP locale diffusera un PDU requête de changement de vue pour une mise à jour de la liste des sites FMP gérée par le protocole (la liste des entités $N + 1$ est gérée par le protocole d'adhésion de niveau $N + 1$).

- **PDU de requête de changement de vue** : par lequel un site FMP demande son insertion dans le groupe pour satisfaire la première demande d'établissement de connexion multipoints provenant d'un processus local de niveau application. De manière symétrique, un site signale son retrait du groupe des entités FMP, pour satisfaire une demande de retrait provenant du dernier processus local de niveau $N + 1$. Une autre circonstance de diffusion de ce PDU, concerne un site FMP qui détecte la défaillance d'un site analogue. Il diffuse une demande de changement de vue traitée par le site séquenceur. Ce PDU est numéroté.
- **PDU de changement de vue** : à la réception d'une requête de changement de vue, le site séquenceur actualise la liste des sites FMP membres du groupe, puis insère cette liste dans un PDU de changement de vue qu'il diffuse vers le groupe pour une actualisation uniforme des vues de chaque site FMP. Ce PDU est numéroté.
- **PDU de requête de retransmission** : permet à un site d'indiquer, aux sites FMP analogues, une perte de PDU(s) et de requérir la retransmission des PDUs signalés. Ce paquet n'est pas numéroté. Ce PDU contient un champ qui permet de spécifier le service FMP émetteur (RDS, CDS ou TODS) et, en conséquence, le type de paquet de retransmission qui devra être retourné.
- **PDU de retransmission** : il s'agit d'un paquet qui contient une donnée perdue et dont la retransmission a été requise. Ce PDU est numéroté. Selon, le type de paquet requête réceptionné, le PDU de retransmission sera une copie conforme du PDU requis, ou bien il comportera des informations additionnelles, comme l'identification du site retransmetteur.
- **PDU ACK** : ce paquet est diffusé exclusivement par le site séquenceur pour établir à l'ensemble du groupe un ordre de livraison sur les paquets de QoS atomique. Ce paquet reflète l'ordre de réception par le séquenceur, dans un délai borné, de paquets données ou de requêtes de changement de vue.
- **PDU de requête de numéro d'ordre causal** : ce paquet est utilisé par le service CDS pour traiter toute première diffusion de message de QoS causale par un site non séquenceur. Le service CDS du site émetteur diffuse une requête d'attribution

d'un numéro d'ordre global. Comme pour les requêtes de changement de vue, cette requête contient un *verrou* qui lui permet de n'être traitée que par le site séquenceur.

- **PDU d'attribution de numéro d'ordre causal** : En réponse à une requête d'attribution d'un numéro d'ordre global, le site séquenceur retourne un PDU contenant un identificateur unique qui sera utilisé par le site requéreur lors de toute diffusion de message de QoS causale. Une autre information essentielle accompagne l'identificateur retourné. IL s'agit d'une structure de donnée représentant l'état de réception, par le séquenceur, de messages de QoS causale.

3.4.4.3 Cycles de numérotation des PDUs

La numérotation des paquets transmis par les entités FMP est confrontée au problème des cycles de numérotation [Cor81]. La taille du champ d'en-tête réservé au numéro de séquence étant limité, le protocole est amené inévitablement à utiliser les mêmes numéros pour des paquets différents.

L'intervalle de temps séparant deux utilisations successives d'un même numéro est appelé le temps de cycle. Si le nombre de bits du champ réservé au numéro de séquence est de n bits, l'incrémentation des numéros de séquence d'un site est limitée à $2^n \Leftrightarrow 1$. Par conséquent toutes les opérations arithmétiques et de comparaison effectuées sur un numéro de séquence doivent être faites en *modulo* $2^n \Leftrightarrow 1$. En fixant la taille du champ numéro de séquence à 23 bits, FMP exige qu'au plus 2^{23} paquets peuvent être créés sur une période égale à la durée de vie d'un paquet datagramme (*i.e.* avant la suppression du PDU des files d'émission ou de réception) au sein du réseau d'interconnexion. Cette condition reste vérifiée lorsque le service IP est utilisé comme service réseau. Aussi, pour éviter toute confusion entre un PDU et un ancien doublon de numéro identique, il est important d'accélérer le processus de détection de pertes et de retransmission, ce à quoi est destinée l'utilisation de paquets de contrôle spéciaux (*paquets état*) dans le protocole FMP, comme nous le verrons plus loin.

3.5 Service de diffusion fiable (RDS)

3.5.1 Mécanisme d'acquiescement

A la différence des communications unicast où les conditions requises pour le transport des données sont assez générales, les applications de groupe ont des besoins en fiabilité et en ordre très variés. En effet, certaines applications requièrent une diffusion totalement

ordonnée qui n'est pas adaptée à d'autres. Aussi, certaines architectures à données dupliquées impliquent une partie ou l'ensemble des sites dans les mises à jour et l'envoi de données, alors que d'autres concentrent les données sur un seul site qui devient ainsi l'unique source de données. Bien que l'on puisse concevoir un protocole générique qui prenne en compte les besoins les plus subtils (par exemple : une livraison de données dupliquées, fiable, totalement ordonnée, à partir d'un groupe de sites émetteurs), une telle approche pénalise les applications ayant des besoins plus modestes.

La meilleure façon de satisfaire la diversité des besoins des applications de groupe est de laisser autant de fonctionnalités et de flexibilité que possible à l'application. Pour cela, nous utiliserons le concept de QoS (Quality of Services) au niveau de l'interface de communication, pour permettre à un processus de l'application de spécifier ses besoins en communication lors de sa diffusion d'un message.

La fiabilité dans un protocole de transport est implémentée en utilisant l'un des deux schémas d'acquittement positif (ACK) ou négatif (NACK).

3.5.1.1 Acquittement positif

Dans le premier schéma (utilisé par TCP), chaque émetteur calcule son intervalle de retransmission³ et reste dans l'état de retransmission jusqu'à réception d'un ACK retourné par tout récepteur qui acquitte systématiquement tout paquet correctement reçu. Bien que cette approche accélère le temps de stabilité⁴ d'un message, elle s'adapte mal à la communication multi-destinataires. En effet, un nombre de problèmes apparaît avec l'utilisation de ce schéma d'acquittement :

- Le fait que chaque destinataire envoie un ACK pour chaque paquet reçu, a pour effet d'augmenter considérablement la charge de l'émetteur, responsable alors de traiter tous les ACK qui lui sont retournés. Cela a pour conséquence de diminuer les performances du protocole puisque le temps de stabilité d'un message augmente inéluctablement. De plus, si les ACK sont envoyés en même temps, cela entraînera une congestion du réseau.
- Aussi, si l'émetteur est responsable de la fiabilité de la livraison, il doit continuellement observer l'ensemble changeant (les groupes étant ouverts) des récepteurs actifs ainsi que l'état de réception de chacun d'eux (défaillance, numéro du dernier paquet

3. *i.e.*, le temps pendant lequel l'émetteur attend l'acquittement d'un paquet avant de décider de le réémettre.

4. *i.e.*, constatation par l'émetteur de la bonne réception, par le destinataire, du paquet émis.

reçu, etc.). Puisque le Multicast IP, sur lequel repose notre protocole, impose un niveau d'indirection⁵ entre l'émetteur et les récepteurs, il est coûteux de déterminer l'ensemble des récepteurs (à cause du caractère dynamique des groupes).

- Finalement, les algorithmes de contrôle de flux et de congestion devant être utilisés pour adapter le protocole aux conditions changeantes du réseau⁶, sont plus subtiles dans le cas du multicast : comment peut être estimé le temps d'aller-retour (round-trip⁷) pour l'initialisation de l'horloge de retransmission quand il peut y avoir plusieurs ordres de magnitude dans le temps de propagation des paquets vers des sites disséminés sur des réseaux physiques différents les uns des autres?. De même, quelle est la valeur de la fenêtre de congestion d'un site si la bande passante attribuée aux différents récepteurs varie selon les mêmes ordres de magnitude?

Ces problèmes illustrent bien que le couplage fort entre l'émetteur et les récepteurs n'est pas naturellement, ni aisément généralisable au cas du multicast.

3.5.1.2 Acquittement négatif

Dans le schéma d'acquittement négatif (NACK), la charge de détection d'erreurs de transmission est déplacée vers les sites destinataires. Les paquets sont estampillés avec des numéros de séquence que les destinataires utilisent pour assurer un service fiable, en détectant un "trou" dans le séquençement (significatif de la perte de paquets) et en demandant, alors, la retransmission des paquets perdus.

Il est important de souligner que les problèmes soulevés plus haut, ne se posent plus dans le cas du NACK, puisque chaque site entretient son propre état de réception. La charge par site est constante car indépendante de la taille du groupe. En outre, la connaissance des membres du groupe n'est pas significative. Il apparaît clairement que le second schéma est mieux adapté aux protocoles multicast.

L'inconvénient majeur de l'acquittement négatif est qu'aucune information sur les paquets transmis (ou *feedbacks* n'est retournée. De ce fait, un émetteur doit maintenir une copie de chaque paquet transmis dans le cas où il reçoit une demande de retransmission de la part d'un des sites destinataires. Afin d'éviter, autant que possible, le problème de congestionnement des tampons de réception, le site émetteur doit supprimer régulièrement

5. Le paquet est adressé au groupe logique et non pas aux sites qui le constituent physiquement à un instant fixé.

6. Comme pour TCP qui ajuste dynamiquement ses paramètres de contrôle (horloge de retransmission, fenêtre de congestion) en fonction des performances observées durant le transfert de données.

7. *i.e.*, l'intervalle entre l'émission d'un paquet et la réception de son acquittement.

une partie des paquets stockés dans sa file de réception. Plusieurs solutions sont possibles. La première consiste, lors de la réception d'une requête de retransmission, à supprimer tout paquet dont le numéro de séquence est inférieur au premier numéro de la séquence de paquets retransmis. Cependant, cette solution comporte un risque dans le cas où le site qui retransmet est le seul à détenir les paquets demandés et que ceux-ci sont à nouveau requis par le même site ou un autre site du groupe. De même, les sites ayant correctement reçu les paquets demandés et qui reçoivent la retransmission, doivent-ils aussi supprimer les paquets de leurs tampons respectifs ou bien prévoir le cas où le site retransmetteur quitte le groupe sans que les paquets retransmis ne soient reçus par le site demandeur?. La figure suivante illustre le problème exposé :

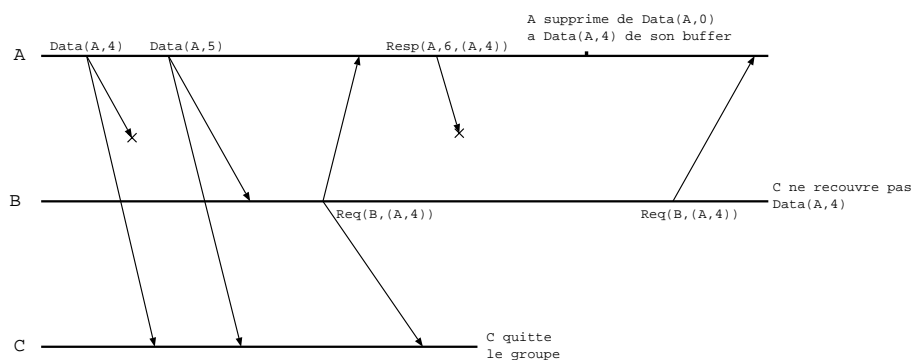


FIG. 3.3 - *Suppression à tort de paquets d'un tampon*

3.5.1.3 Message état

Un autre problème spécifique au schéma d'acquiescement négatif est qu'un paquet perdu n'est détecté que lorsque le paquet qui le suit en séquence est correctement reçu, ce qui pourrait prendre un temps considérable dans les applications ayant des périodes d'inactivité importantes et répétées. Une solution souvent utilisée est que chaque membre du groupe diffuse périodiquement un paquet spécial (nommé *paquet état*) qui annonce le dernier numéro de séquence reçu des membres actifs du groupe. Pour diffuser un paquet état, chaque site dispose d'une structure de donnée dynamique qu'il construit au fur et à mesure des messages qu'il reçoit entre deux émissions de paquet état. A chaque réception d'une donnée, le site vérifie si l'émetteur dispose d'une entrée dans la liste. Si c'est le cas, il met à jour le numéro de séquence correspondant à l'émetteur. Sinon, il rajoute une entrée en fin de structure et initialise sa valeur au numéro de séquence reçu.

Une horloge spécifique contrôle la période d'émission du paquet état. à son expiration, l'émetteur insère dans le paquet état la structure qui reproduit l'état de réception du site relativement aux sites ayant diffusé des données.

Ainsi, un paquet état comprend deux parties : une partie donnée contenant la *table d'état de réception* qui reporte, pour chaque site actif du groupe, le dernier numéro de séquence reçu de ce site, et une partie en-tête contenant l'identification du site émetteur ainsi que les informations utilisées par les sites récepteurs pour calculer leur distances respectives par rapport à l'émetteur du message état. Nous verrons plus loin (3.5.2) comment ces distances sont calculées. Nous verrons, également dans le prochain paragraphe les autres traitements effectués par les récepteurs d'un paquet état.

Notons, enfin, qu'un paquet état n'est pas numéroté. Le service de livraison FMP assure, par conséquent, une livraison non fiable du paquet. La raison pour ce choix est double : d'une part, ne pas surcharger, le mécanisme de recouvrement d'erreurs, en cas de perte d'un paquet état, mais aussi, le caractère périodique du paquet état fait qu'une perte signalée par un paquet état le sera également dans le prochain paquet si le premier est perdu.

3.5.1.3.1 Mécanismes adoptés :

Le service de livraison fiable fourni par FMP repose sur un schéma d'acquittement négatif qui comme nous l'avons vu est plus adapté à la communication multi-destinataires dans les groupes ouverts et étendus (en nombre et en surface).

Concernant les messages état, FMP doit prendre en compte le fait qu'entre la diffusion d'un paquet état (supposons par un site S_a) et sa réception (par S_b), il est possible que le site récepteur diffuse de nouveaux messages. Si aucune solution n'est prévue, le site récepteur déclenchera, à tort, un processus de retransmission. Le paquet état étant, aussi, utilisé pour la mise à jour des distances temporelles⁸ entre sites, cette technique nous permet d'éviter le problème cité plus haut, de la manière suivante :

Soient :

- t_a , le temps transmis par S_a dans son message état, correspondant au temps de réception par S_a du dernier paquet état émis par S_b ,
- t_b1 , le temps d'émission du dernier paquet fiable, par S_b . t_b constitue un champ d'un enregistrement qui conserve le numéro de séquence du dernier paquet de QoS fiable diffusé,
- t_b2 , le temps de réception, par S_b du paquet état diffusé par S_a ,
- $D_{a,b}$ la nouvelle distance temporelle calculée par S_b , à la réception du paquet état diffusé par S_a ,

8. La distance temporelle entre deux noeuds est égale au temps d'aller-retour (RTT) divisé par deux

- Seq_1 , le dernier paquet fiable reçu par S_a à partir de S_b ,
- Seq_n , le dernier paquet fiable émis par S_b ,

L'algorithme suivant décrit un procédé simple qui permet de vérifier, lorsque le paquet acquitté par S_a ne correspond pas au dernier paquet émis par S_b , si le site S_a a reçu le dernier paquet émis par S_b avant la diffusion du paquet état. Si c'est le cas, alors la valeur Seq_1 retournée dans le paquet état doit correspondre à Seq_n . Cependant, si Seq_1 est strictement inférieur à Seq_n , c'est que S_a a manqué la réception des derniers paquets diffusés par S_b .

3.5.1.3.1.1 Algorithme :

```

si  $t_b1 = t_a$ 
  alors ignorer paquet état
  sinon si  $Seq_1 < Seq_n$ 
    alors si  $t_b1 + D_{a,b} > t_b2 - D_{a,b}$ 
      alors ignorer paquet état
      sinon retransmission des paquets de l'intervalle  $[Seq_1 + 1, Seq_n]$ 
    fsi
  sinon ignorer paquet état
fisi
fisi

```

Cet algorithme ne résout pas le cas où : $t_b1 + D_{a,b} \leq t_b2 \Leftrightarrow D_{a,b}$. Théoriquement, un nouveau test doit porter sur la bonne réception, par S_a du paquet fiable dont le temps d'émission (soit, t_i) par S_b précède directement t_b1 , et ainsi de suite jusqu'au paquet dont le temps succède directement à t_a . Si à un niveau de ces vérifications successives, la condition $t_i + D_{a,b} > t_b2 \Leftrightarrow D_{a,b}$ est vérifiée, le site S_b retransmet les paquets de l'intervalle $[Seq_i, Seq_n]$, Seq_i correspondant au paquet diffusé par S_b à l'instant t_i , puis arrête l'itération. Seulement, en pratique cette version de l'algorithme est subordonnée au fait que chaque paquet donnée ou réponse émis doit être estampillé du temps local d'émission. Ainsi, à la réception d'un paquet état, S_b applique l'algorithme en vérifiant, dans sa file de réception, les temps d'émission relatifs à chaque paquet stocké. Pour effectuer les vérifications uniquement sur les paquets émis par le site local, il faudrait s'assurer que l'identificateur du paquet vérifié correspond au site local. Tout ce traitement est, évidemment, coûteux en ressource processeur et en temps, ce qui diminue de son intérêt. De plus, le temps d'émission constitue une information redondante par rapport au numéro de séquence qui suffit aux fonctionnalités du protocole. Pour diminuer le nombre de vérification, on peut estampiller chaque paquet émis par son temps d'émission.

Ainsi, à chaque réception de message provenant de S_b , S_a met à jour l'entrée, correspondant à S_b , dans la table d'état de réception (ce qui sous-entend une table d'état plus

grande que si la première version de l'algorithme était adoptée). S_a y insère le dernier numéro de séquence reçu, ainsi que le temps d'émission qui accompagne chaque paquet reçu. Cette valeur sera retournée dans le paquet état et utilisée, par S_b , pour le calcul de la distance $D_{a,b}$. De cette manière, la valeur de $D_{a,b}$ sera encore plus proche de la distance temporelle réelle, consolidant, ainsi, le test : $t_{b1} + D_{a,b} > t_{b2} \Leftrightarrow D_{a,b}$. D'un autre côté, elle permet de n'effectuer les vérification que sur les paquets dont le temps est supérieur à la valeur retournée (dans la première version, le temps de référence retourné est celui de la dernière émission par S_b du paquet état et non pas d'un paquet donnée ou réponse). Malgré l'amélioration proposée dans la dernière version de l'algorithme, il reste qu'elle est lourde dans son exécution, au niveau du la file de réception de S_b , de la table d'état de S_a et du marquage de chaque paquet par son temps d'émission. Pour cela, nous avons retenu, malgré son incomplétude, la première version de l'algorithme.

3.5.1.3.2 Suppression de paquets des buffers de réception :

La méthode adoptée pour supprimer les paquets considérés comme stables, utilise la diffusion des paquets état : pour chaque site source de paquet, le site récepteur d'un paquet état prend l'estampille minimale acquittée par l'ensemble des site et supprime tous les paquets de son tampon dont l'estampille est inférieure ou égale à l'estampille communément acquittée. Par opposition à la solution décrite dans 3.5.1.2, cette solution constitue un moyen de suppression efficace.

3.5.2 Gestion des pertes dans FMP

Lorsqu'un message est diffusé, chaque site du groupe est individuellement responsable de la détection des pertes et de la demande de retransmission. Le service réseau IP utilisé par FMP supprime tout message altéré lors de sa diffusion. Un code de détection d'erreurs pallie, ainsi, à l'absence d'un code de correction d'erreur (largement plus coûteux en informations et en temps et parfois inefficace dans le cas d'applications totalement fiables).

Une perte est normalement détectée par la réception d'un paquet dont le numéro de séquence est supérieur au dernier numéro de séquence augmenté de un. Ce mécanisme nécessite la diffusion périodique de paquets état.

3.5.2.1 Calcul des distances entre noeuds

Un paquet état contient une estampille utilisée pour estimer la *distance temporelle* de chaque site par rapport aux autres. Pour calculer la distance entre deux sites, nous utili-

serons la méthode décrite dans [FJM95]. Il s'agit d'une version simplifiée de l'algorithme de synchronisation utilisé dans NTP [Mil92]. Le principe est le suivant :

Supposons qu'un site S_a envoie à l'instant t_1 un paquet P_1 , reçu par le site S_b à l'instant t_2 . S_b génère à l'instant t_3 un paquet état P_2 , marqué de (t_1, Δ) où $\Delta = t_3 \Leftrightarrow t_2$. Lorsqu'au temps t_4 , S_a reçoit P_2 , il peut estimer la distance temporelle qui le sépare de S_b , c.a.d. : $(t_4 \Leftrightarrow t_1 \Leftrightarrow \Delta)/2$.

Cette estimation suppose que les chemins sont symétriques. Cette hypothèse est également faite dans la conception de FMP.

3.5.2.2 Technique du slotting/damping

Le mécanisme de recouvrement de perte utilisé dans FMP a été à l'origine utilisé dans XTP [XTP92] puis repris dans de nombreux protocoles tel que SRM [FJM95]. Il s'agit de deux techniques complémentaires de contrôle de flux : la première est la technique de l'amortissement (*damping*) où chaque récepteur ne diffuse un message de contrôle que s'il est le premier membre à diffuser. La seconde est la technique de la temporisation ou de l'échantillonnage (*slotting*). Elle augmente l'efficacité de la première technique en imposant, pour chaque récepteur, un délais aléatoire avant la diffusion d'un message de contrôle [Dan89].

Lorsqu'un site (soit, S_a) détecte la perte d'un paquet, il temporise un laps de temps, déterminé à partir de sa distance par rapport à l'émetteur (soit, S_b) du paquet. A l'expiration de l'horloge, appelée *horloge de requête*, il diffuse une demande de retransmission de la séquence de paquets perdus. Bien que plusieurs sites peuvent détecter la perte d'un même paquet, le site (soit, S_c) le plus proche du point de défaillance (et donc de S_b) a la plus grande probabilité que son horloge expire avant les autres sites. Ces derniers qui reçoivent la requête de retransmission suspendent leur processus de demande de retransmission. Ce mécanisme a pour objectif d'éviter l'effet d'implosion des requêtes de retransmission qui pénaliserait le débit de données sur le réseau de communication.

Tout site détenant une copie du paquet requis en retransmission peut répondre à la demande. Il positionne une horloge, appelée *horloge de retransmission*, à une valeur dépendant de sa distance par rapport à S_c . Lorsque l'horloge expire, il diffuse le paquet requis. Les autres sites, dont la distance par rapport à S_c retarde l'expiration de leurs horloges de retransmission, reçoivent le paquet retransmis et, de ce fait, suspendent leurs processus de retransmission. Cela a pour objectif d'éviter l'effet d'implosion des retransmissions.

Ainsi, dans une topologie avec plusieurs délais de retransmission, un paquet perdu risque

fort de ne déclencher qu'une seule requête, *de la part du site juste en aval du point de défaillance*, et une seule réponse *de la part du site juste en amont du point de défaillance*. Nous reviendrons ultérieurement sur les détails de mise en oeuvre des mécanismes de requête et de reprise. Cette démarche présente, comme nous l'avons déjà cité, l'inconvénient d'imposer à tout site de conserver dans un tampon de réception une copie des paquets correctement reçus, en vue d'une éventuelle retransmission. Pour éviter une saturation du tampon local, tout site doit supprimer les paquets dont le numéro de séquence est inférieur au numéro retourné par la paquet état, une fois que ce dernier a été réceptionné d'un membre du groupe.

3.5.3 Mécanisme de requête

Une requête est un paquet de contrôle utilisé par le protocole pour assurer un service de livraison fiable. Sa structure est composée de plusieurs champs, dont, notamment, le champ identifiant le site originaire de la requête et le champ identifiant l'ensemble des paquets requis. Cet ensemble est formé d'un champ identifiant le site duquel les paquets requis ont été perdus ou mal réceptionnés, et d'un champ contenant un tableau identifiant les numéros de paquets perdus. Ces numéros de séquence ne constituent pas forcément une suite numérique. La raison est donnée par le mécanisme de gestion des requêtes décrit plus bas.

3.5.3.1 Non numérotation des requêtes

Les paquets requête ne sont pas numérotés. Leur perte n'entraîne donc aucun mécanisme de réparation explicite. Pour assurer que tout site émetteur d'une requête reçoit la réponse attendue dans un laps de temps borné, le mécanisme de fiabilité prévoit que le site émetteur rediffuse la requête à l'expiration d'un délai de garde fixé. Ce procédé est renouvelé trois fois, puis, en l'absence de réponse, un autre site ayant détecté la même perte, prend le relai de la procédure de requête. Le schéma de fiabilité proposé suppose en effet que le premier site demandeur est dans un état incorrect l'empêchant de recouvrir les données perdues.

3.5.3.2 Algorithme de reprise

L'algorithme de reprise constitue la base de la fiabilité du protocole. Lorsqu'un site S_a détecte une perte de paquets originaires de S_b , il positionne l'horloge de requête à $D_{a,b}$ qui correspond au temps ajusté requis par un paquet émis par S_a pour atteindre S_b . La

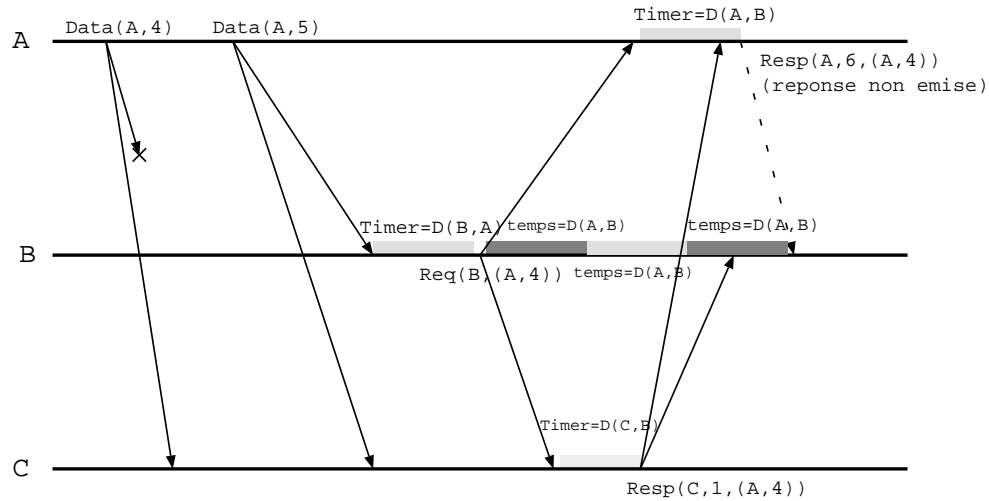
valeur de l'horloge de requête est prise de l'intervalle de temps $[C_1 * D_{a,b}, C_2 * D_{a,b}]$, où C_1 et C_2 représentent deux constantes positives (avec $C_2 > C_1$). L'échantillonnage de la valeur de l'horloge a deux objectifs : d'une part, il permet de majorer la valeur de $D_{a,b}$ pour prendre en compte le temps de traitement de la requête par le site récepteur. D'autre part, il permet d'éviter que plusieurs sites ne diffusent leur requêtes en même temps (effet d'implosion des requêtes). Pour des sites appartenant à un même réseau local, le fait que chaque site tire une valeur d'un intervalle diminue la probabilité d'envois simultanés. Les distances respectives de ces sites sont, en effet, égales par rapport à un site source (S_b) se trouvant à l'extérieur du réseau local.

Périodiquement, les paquets état sont utilisés par les sites du groupe pour réajuster leurs distances temporelles respectives, en fonction des conditions changeantes des flux de données sur le réseau de communication.

Si les paquets requis parviennent à S_a avant l'expiration de l'horloge de requête, le processus de demande de retransmission est suspendu. Dans le cas où l'horloge de requête expire normalement, S_a diffuse sa requête et triple la valeur de l'horloge de requête, dans l'attente des paquets réponse. Le triplement de la valeur de l'horloge correspond au cumul des trois temps suivants : le temps nécessaire à la requête pour atteindre le site le plus proche, la temporisation du site récepteur avant de retransmettre le paquet requis, et enfin le temps d'acheminement du paquet réponse jusqu'à S_a . Si le site S_b est le site le plus proche de S_a , alors $3 * D_{a,b}$ est le temps minimum pour que la réponse de S_b parvienne à S_a . Sinon, la réponse du site le plus proche (par exemple, S_c) parviendra avant l'expiration de la nouvelle valeur de l'horloge. Toutefois, le risque de la retransmission de la requête subsiste, si le site S_b est le plus proche et si une défaillance temporaire du réseau l'empêche de répondre à la requête.

A l'expiration du nouveau délai de garde, la requête initiale est retransmise. Le procédé est répété deux fois au maximum. Le site demandeur est alors considéré comme défaillant ou appartenant à une partie défaillante du réseau. Dans cette hypothèse, il est judicieux de faire relayer la requête de retransmission par un autre site ayant aussi détecté la perte.

Si le site S_a reçoit une requête pour le même paquet avant que son horloge n'expire, il repositionne alors son horloge de requête à trois fois la valeur initiale. Cela a pour but de garantir la reprise, par un autre site, du processus de demande de retransmission en cas de défaillances du site initiateur de la requête ou du site chargé de retransmettre le paquet (si ce site est unique). Les valeurs successivement prises par l'horloge de requête sont successivement tirées de la distribution : $3^i D_{a,b}$, $i = 1, 2, 3, \dots$. Le choix de tripler la valeur de l'horloge est justifié par le fait que si des diffusions successives d'une même requête devaient avoir lieu, elles le seraient par un même site. En effet, le fait qu'un site

FIG. 3.4 - Schéma explicatif de $3 * D_{a,b}$

(S_e) triple la valeur de son horloge de requête à l'émission de la requête, et que le site S_a double uniquement la sienne à la réception de la requête de S_e , peut engendrer la situation décrite ci-dessous :

Hypothèses : $D_{a,b} = 3$ (unités de temps), $D_{e,b} = 2$ et $D_{a,e} = 1$

- Emission de la requête par S_e : $RequestTimer_{S_e} = 3 * RequestTimer_{S_e} = 6$
- Réception de la requête par S_a : $RequestTimer_{S_a} = 2 * RequestTimer_{S_a} = 6$
- Expiration de l'horloge de S_e : ré-émission de la requête, $RequestTimer_{S_e} = 18$,
- Réception de la requête par S_a : $RequestTimer_{S_a} = 12$,
- Expiration de l'horloge de S_a : émission de la requête, $RequestTimer_{S_a} = 36$,
- Réception de la requête par S_e : $RequestTimer_{S_e} = 36$,
- Expiration de l'horloge de S_a : ré-émission de la requête, $RequestTimer_{S_a} = 108$,
- Réception de la requête par S_e : $RequestTimer_{S_e} = 72$,
- Expiration de l'horloge de S_e : ré-émission de la requête,
- etc.

cela conduirait à une alternance dans l'émission de la requête dans le cas où aucun site fonctionnel ne peut, momentanément, y répondre, ce qui rendrait le processus de recouvrement plus complexe.

3.5.3.3 Problème des requêtes multiples

A cause de la nature probabiliste de ces algorithmes de recouvrement (les horloges sont aléatoirement positionnées à partir d'intervalles de valeurs), il n'est pas exclu qu'un paquet perdu soit suivi de plusieurs requête de retransmission. Ainsi, il est possible qu'un site reçoive une requête de retransmission immédiatement après avoir émis la sienne, ou bien après avoir reçu un paquet réponse conséquent à sa propre requête diffusée. Deux solutions permettent de gérer ces deux cas de figure.

La première est une solution en amont. Elle se situe au niveau de l'insertion d'un nouveau site dans le groupe et concerne les applications qui intègrent un protocole d'adhésion et de retrait de niveau supérieur (relativement au niveau session), qui gère l'adjonction et le retrait de processus dans (ou de) la liste des entités de l'application⁹. Le site gestionnaire du groupe¹⁰ attribue à chaque nouveau site appartenant à un réseau local composante du groupe, une constante C_i , tirée d'un intervalle d'entiers positifs $[C_1, C_n]$, augmentée à chaque nouvelle adhésion d'un site du même réseau local. Ainsi, on élimine le risque que plus d'un site d'un même réseau local n'émette une requête de retransmission. Cette solution évite, seulement, que deux sites d'un même réseau local ne diffusent une même requête de retransmission.

La seconde solution est une solution en aval. Elle est mise en évidence par les situations suivantes : S_e diffuse une requête après une détection de perte. S_b , qui est un site du groupe pouvant détenir le paquet requis, reçoit la requête émise par S_e .

La structure de donnée, correspondant à la file de réception d'un site FMP, adoptée pour cette solution est présentée dans le chapitre suivant (4.2.1.2). Elle comporte 3 ou 4 champs, selon le type d'information contenue par un élément de la liste. Le premier champ (*State*) désigne le type de l'élément de la file (donnée, donnée retransmise, ou requête). Le second champ contient le paquet stocké. Pour les paquets donnée, le troisième champ (*AnsweredPacket*) indique, si le paquet a été retransmis ou non par le site local, ou s'il a été retransmis par un autre site. Le dernier champ (*ReqSource*) identifie le plus récent site originaire de la requête dans le cas où le paquet donnée stocké a été retransmis par le site récepteur de la requête.

Dans le cas d'un paquet requête, les deux champs qui suivent le champ *State*, correspondent, respectivement, au premier paquet et au dernier paquet requis, le quatrième champ n'existe pas dans ce cas. Dans le cas d'un paquet retransmis, c'est le champ *AnsweredPa-*

9. Rappelons la différence faite entre le groupe formé par les sites ou noeuds du réseau, qui fait l'objet de notre étude, et le groupe de niveau supérieur formé par les processus de l'application supportés par ces derniers sites.

10. qui peut être, par exemple, le site contenant le processus créateur de l'application.

cket qui n'existe pas.

A la réception d'un paquet requête par S_b , deux situations peuvent se présenter : soit S_b détient le paquet requis, soit il ne l'a pas. L'algorithme ci-dessous explicite les traitements effectués dans chacune des deux situations.

1. S_b a le paquet requis : dans ce cas, trois nouvelles possibilités se présentent :

a) la requête reçue est un "doublet", S_b est en cours de diffusion de la réponse correspondante. Deux cas, alors, sont possibles :

- le site originaire des deux requêtes est le même : S_b déduit que le site originaire du paquet perdu (S_a , par exemple) peut être défaillant. En effet, comme S_e calcule son horloge de requête en fonction de S_a , si S_e retransmet sa requête, c'est que, soit que S_a n'a pu y répondre, ou qu'un partitionnement entre S_a et S_e a empêché la réponse de S_a de parvenir, soit S_e est défaillant et n'a donc pas reçu la réponse de S_a ou d'un autre site plus proche. Dans l'éventualité de la défaillance de S_a , S_b remplace la valeur de son horloge de retransmission à $D_{b,e}$, en prévision d'une prochaine retransmission. Si le protocole le prévoit, S_b déclenche un mécanisme de recouvrement de défaillance concernant le site S_a , ou la partition comprenant le site S_a ,

- l'origine de la requête est différente de la seconde : S_b ignore la requête reçue.

b) S_b a déjà diffusé le paquet réponse correspondant :

le champ AnsweredPacket, initialement à 0, relatif à la donnée stockée signale que la paquet réponse a déjà été émis. Après émission du paquet réponse, S_b ignore tout message de requête sur cette donnée durant une période de temps déterminée. S'il y a lieu, il modifie, aussi, le champ identifiant le site originaire de la requête. Pour déterminer la période de temps suffisante pour ignorer les requêtes multiples, nous devons émettre les hypothèses suivantes : les topologies et débits des réseaux traversés par les paquets sont relativement équivalents. Il en est de même pour l'état de congestionnement de ces réseaux interconnectés. Soient les sites S_a , S_b , S_y et S_x tels que: $D_{a,y} < D_{b,y}$, $D_{b,x} = \text{Max}(D_{b,i})$ et $D_{b,y} = \text{Min}(D_{b,i})$, $i = 1, n$, $i \neq b$, n étant la taille du groupe.

Supposons que les sites S_x et S_y détectent une perte de paquets originaires de S_a . S_y diffuse une requête de recouvrement. S_b y répond et inhibe le traitement

de requêtes durant T . Le temps total correspondant à la situation décrite ci-dessus, est :

$D_{a,y}^{11} + D_{a,y}^{12} + D_{y,b}^{13} + D_{y,b}^{14} + T$. Afin d'éviter une seconde réponse inutile du site S_b , on doit avoir :

$$2 * D_{a,y} + 2 * D_{y,b} + T > 2 * D_{a,x} + D_{x,b} \quad (1)$$

la partie droite de l'inéquation correspond au temps d'acheminement de la donnée de S_a vers S_x (le site le plus éloigné de S_b), à la temporisation du site S_x avant de diffuser la requête et, enfin, au temps d'acheminement de la requête vers S_b .

$$(1) \text{ est équivalente à : } T > 2 * D_{a,x} + D_{x,b} \Leftrightarrow 2 * D_{a,y} \Leftrightarrow 2 * D_{y,b}$$

Désignons par X , la partie droite de l'inéquation :

$$X < 2 * D_{a,x} + D_{x,b} \quad (2)$$

Les hypothèses faites ci-dessus, permettent décrire :

$$D_{a,x} < D_{a,b} + D_{b,x} \quad (3)$$

Comme $D_{a,b} < D_{b,x}$, (3) devient : $D_{a,x} < 2 * D_{b,x}$,

ce qui ramène (2) à : $X < 5 * D_{b,x}$

d'où (1) devient équivalente à : $T > 5 * D_{b,x}$

La temporisation suffisante pour éviter les réponses successives d'un site S_i est donc : $T = 5 * D_{i,x}$, où S_x est le site le plus éloigné par rapport à S_i .

Une exception est faite lorsqu'un site reçoit deux fois la même requête sur une donnée, soit qu'il a retransmise, soit qu'il a réceptionné un paquet réponse correspondant. Nous indiquerons dans la section qui traite les messages de réparation, comment un site détermine l'une des deux informations citées ci-dessus. Dans ce cas, il initie un nouveau processus de retransmission sans attendre l'expiration de l'horloge d'inhibition. Notons que la valeur $D_{i,x}$ est actualisée au fur et à mesure qu'un site S_i reçoit les messages état (si S_i reçoit un message état de S_a alors $D_{i,x} = D_{i,a}$ si $D_{i,a} > D_{i,x}$). Le problème qui peut se poser est qu'un nouveau site (qui n'a pas encore diffusé de paquet état) plus éloigné que S_x émet une requête qui pourrait à S_i après expiration de la période d'inhibition. S_i procédera alors à une retransmission inutile de la donnée requise.

c) S_b a déjà reçu la réponse correspondant à la requête :

S_b procède à la modification du champ *ReqSource*. S'il s'agit d'une deuxième requête d'un même site, S_b initie un nouveau processus de retransmission. Dans

11. temps d'acheminement d'une donnée de S_a vers S_y

12. horloge de requête de S_y

13. temps d'acheminement de la requête de S_y vers S_b

14. horloge de retransmission de S_b

le cas contraire, il ignore la requête si celle-ci porte sur une donnée déjà requise (par S_b ou par un autre site) et que l'horloge d'inhibition est encore active. Si la donnée demandée était déjà requise par S_B , celui-ci le détecte grâce à la présence du paquet réponse dans sa liste de réception. Si la donnée a été requise par un autre site, S_B le détecte grâce au champ *AnsweredPacket* relatif à la donnée requise qui est stockée également dans la file de réception.

2. S_b n'a pas le paquet requis : trois cas sont possibles :

a) S_b est en cours d'envoi du même paquet requête

S_b constate qu'il est en instance de diffuser le paquet requête reçu, il ignore donc la requête et suspend son propre processus de demande de retransmission, selon le procédé décrit plus haut (triplement de la valeur de l'horloge de retransmission).

b) S_b a déjà envoyé le paquet requête reçu

S_b constate l'envoi préalable de la requête grâce à une configuration spéciale de la file de réception. En effet, la série de paquets manquants est remplacée par un paquet spécial inséré dans la liste juste avant la séquence de paquets qui avait déclenché la diffusion de la requête. Ce paquet spécial contient les informations suivantes : premier paquet perdu, dernier paquet perdu. Il est inséré dans la liste après l'émission de la requête correspondante. Comme S_b ne peut pas avoir réceptionné le paquet réponse, il ignore le paquet requête reçu.

c) S_b n'est ni en cours d'envoi de la demande de retransmission, ni a émis au préalable celle-ci

S_b initie un processus de requête portant sur le même paquet, pour prendre le relais de demande de retransmission en cas de défaillance du site source de la requête reçue.

Nous avons adopté la seconde solution dans la conception de FMP, pour, d'une part, son adaptabilité à une plus large collection d'applications, comme celles qui ne prévoient pas de protocole d'adhésion et de retrait de processus du groupe. D'autre part, cette solution consolide le mécanisme de fiabilité voulu dans la conception de FMP en fournissant un élément de détection de défaillance que peuvent exploiter les applications sensibles aux défaillances. D'autant plus que la première solution n'élimine pas les différents cas de multiples requêtes dans le cas où celles-ci émanent de sites appartenant à des réseaux locaux différents.

3.5.4 Mécanisme de retransmission

Un paquet retransmis constitue une réponse à une requête diffusée au sein du groupe. A une seule requête identifiant une liste de paquets perdus, peut correspondre un ensemble de paquets réponse. Comme pour un paquet donnée, la structure d'un paquet réponse comprend deux parties : une partie en-tête qui permet les opérations de contrôle de fiabilité, et une partie contenant la donnée retransmise. Le premier champ d'en-tête identifie le site à l'origine de la diffusion, ainsi que le numéro de séquence attribué par ce dernier. L'utilité de numéroter un paquet réponse est discutée plus bas. Le second champ d'en-tête identifie le site originaire ainsi que le numéro de séquence du paquet retransmis dans le cas où le site source de la donnée n'est pas le site retransmetteur. Le choix de rajouter ce second champ est également discuté plus bas.

Un site qui répond à une demande de retransmission, est, soit détenteur de la donnée requise, soit qu'il a reçu le paquet réponse diffusé par un autre site. Dans les deux cas, il construit son message de recouvrement, en se limitant à la donnée requise et à l'identificateur de l'émetteur original de la donnée qui est rediffusée. En d'autres termes, si la donnée requise a été reçue, par un site S_i , dans un paquet réponse diffusé par un site S_j , et si l'origine de cette donnée est le site S_o , alors si S_i retransmet cette donnée, il indiquera le site source (S_o) et non pas le site retransmetteur (S_j).

Un paquet réponse n'est sauvegardé ni par son site émetteur, ni par les sites récepteurs. Il est stocké, pour une éventuelle retransmission, uniquement dans les files de réception des sites qui l'ont requis. A la différence d'un paquet donnée sauvegardé, un élément de la file de réception, contenant un paquet réponse ne comporte pas le champ "AnsweredPacket". La présence d'un paquet réponse signifie, à elle seule, le fait que ce dernier a déjà été diffusé dans le groupe.

3.5.4.1 Numérotation des messages de retransmission

Le problème de numérotation ou de non numérotation des paquets réponse nous a contraint à choisir entre deux sémantiques de performance qui s'opposent : la première porte sur la définition de la performance en termes de débit et de flux de données, la seconde porte sur l'aspect fiabilité du protocole. La recherche des performances de débits optimales conduit à la non numérotation des paquets réponse. En effet, une QoS fiable appliquée sur un paquet réponse entraîne une charge supplémentaire pour tout récepteur de ce paquet. Celui-ci devra détecter une éventuelle perte et mettre à jour sa file de réception de la manière décrite dans 3.5.3.3.

De plus, si une perte est détectée, le site récepteur déclenche un processus de requête qui augmente la charge du site (ressource processeur demandée par l'horloge) et du réseau après émission de la requête. D'un côté conceptuel, le choix de ne pas numéroter un paquet réponse diffusé est justifié par le fait que si un site récepteur (soit, S_a) a perdu des paquets donnée diffusés, au préalable, par le site émetteur du paquet réponse (soit, S_b), il aura l'occasion de détecter cette perte dès la réception de nouveaux paquets donnée émises par S_b . Cependant, certaines application peuvent vouloir favoriser l'aspect fiabilité sur l'aspect performances. Pour cela, le service de livraison doit prendre en compte les points suivants : le risque de départ (accidentel ou volontaire) du site S_a avant que S_b n'ait diffusé de nouveaux paquets. Dans le cas d'un départ volontaire, le protocole peut prévoir qu'un site quittant le groupe diffuse un paquet état, puis temporise un laps de temps suffisant pour la réception d'éventuelles retransmissions. Cela est, toutefois, contradictoire avec l'objectif initial qui vise à réduire, autant que possible, les messages de contrôle. Le second point à considérer, est le besoin d'accélérer la détection d'éventuelles pertes. Dans la situation décrite ci-dessus, S_a peut détecter un éventuelle perte dès la réception du paquet réponse numéroté par S_b .

Comme notre approche, dans la conception de FMP, a été de mettre l'accent sur la flexibilité du protocole, il est plus adapté de laisser au processus qui crée la session de groupe, d'opter, à sa connexion, pour l'un ou l'autre aspect de fiabilité, selon les besoins de l'application. Dans le prototype de FMP réalisé, nous avons choisi de numéroter les paquets réponse.

3.5.4.2 Identification de l'émetteur d'un paquet retransmis

Concernant le second champ d'en-tête, sa présence relève d'un choix conceptuel justifié par le fait qu'un paquet réponse provient du site le plus proche (qui n'est pas forcément le site originaire de la donnée requise) du site ayant fait la demande de retransmission. Ainsi, un site (S_a) qui reçoit un paquet réponse, procède à la détection de perte portant sur le paquet réponse lui même, mais aussi sur le paquet retransmis si celui-ci n'a pas la même origine (soit, S_c) que le paquet réponse (émis, par exemple par S_B). Comme pour le problème de numérotation des paquets réponse, ce choix génère un traitement, éventuellement inutile, pour le site récepteur (dans le cas où les pertes détectées sont en cours de transmission), mais permet d'anticiper la détection de pertes. En effet, si réellement il y a perte, celle-ci ne sera détectée par S_a qu'à la prochaine diffusion de paquets donnée par S_c . Cela a l'inconvénient, d'une part, de retarder les processus du site S_a sur les autres processus de la session courante. D'autre part cela risque de priver, irrémédiablement, ces mêmes processus des données diffusées par S_c (en non reçues par

S_a) si celui-ci quitte le groupe sans diffuser de nouvelles données. Le souci de renforcer la fiabilité du protocole, nous avons opté pour l'identification des données retransmises.

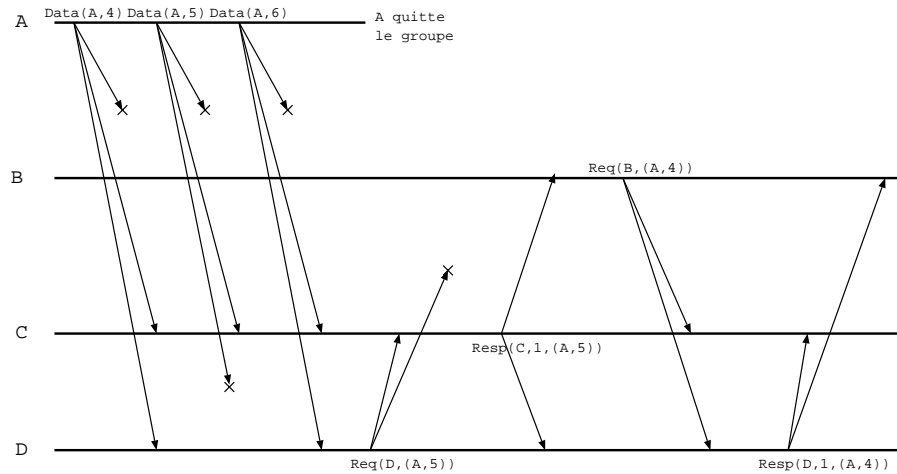


FIG. 3.5 - Numérotation des paquets réponse

3.5.4.3 Description du mécanisme

Nous décrivons à présent le mécanisme de recouvrement de perte après sa détection : lorsqu'un site (S_c) reçoit la requête de S_a et qu'il est en mesure d'y répondre, il programme la diffusion du paquet requis à l'expiration de son horloge de réponse positionnée en fonction de $D_{a,c}$. La valeur de l'horloge de requête est aléatoirement choisie de l'intervalle $[D_1 * D_{a,c}, D_2 * D_{a,c}]$, $D_2 > D_1$.

Comme pour l'émission de requêtes, cette précaution permet d'éviter l'effet d'implosion des réponses des sites d'un même réseau local. Si S_c réceptionne le paquet requis avant que son horloge n'expire il annule sa réponse programmée et met à jour le champ *AnsweredPacket* relatif à la donnée retransmise afin d'éviter les réponses multiples. Pour que le mécanisme décrit dans 3.5.3.3 soit accompli, il est nécessaire que tout émetteur S_i de message réponse y insère la temporisation ($5 * D_{i,x}$, S_x , étant le site le plus éloigné de S_i) qui doit être respectée par tout site détenteur du message retransmis.

Ainsi, un site S_j qui reçoit le paquet réponse avant d'avoir reçu la requête correspondante, ignore cette requête durant le laps de temps indiqué. Pour ce faire, S_j consulte le champ *AnsweredPacket* relatif à la donnée retransmise. Si la valeur de ce champ est à 2, S_j déduit qu'il est le site qui a retransmis le paquet requis. La valeur de *AnsweredPacket* est dans notre cas, égale à 1, indiquant que la donnée a été retransmise par un autre site que S_j . Ainsi, si S_j reçoit un paquet requête doublon, deux stratégies de recouvrement sont possible : la première consiste à ce que S_j vérifie si la valeur $5 * D_{i,x}$ de l'horloge n'est pas

atteinte avant de retransmettre la donnée requise. La seconde consiste à supposer qu'à la suite d'un partitionnement, la réponse diffusée par S_i n'a pu atteindre le site dont il vient de recevoir la requête. Dans cette éventualité, S_j entame un processus de retransmission sans attendre l'expiration de l'horloge d'inhibition des requêtes. Nous avons, cependant adopté le premier mécanisme car la requête doublon peut être simplement due au congestionnement du réseau et non pas à un partitionnement de celui-ci. En effet, nous avons vu dans ?? que lorsque le réseau était utilisé à 75% de ses capacités, on peut observer une variation du temps d'aller-retour (RTT) d'un facteur de 16 [Jac88]. Dans un système asynchrone (où les délais ne sont pas bornés), il est impossible de distinguer, dans un délai court, entre le défaut de transmission d'un message et un temps de transmission très long.

Si, par contre, la valeur de AnsweredPacket est à 2, S_j vérifie si la valeur de l'horloge est comprise entre $3 * D_{a,j}$ et $5 * D_{j,x}$ (en supposant que S_a soit le site source de la requête). Si la condition est vérifiée, S_j entame un nouveau processus de retransmission vers S_a . Dans le cas où la valeur est inférieure à $3 * D_{a,j}$, S_j ignore le message requête reçu. Théoriquement, le temps $D_{a,j}$ suffit pour que le paquet réponse initialement diffusé par S_j parvienne à S_a , mais, par précaution, le seuil de vérification est élevé à $3 * D_{a,j}$, dans le cas possible d'un congestionnement du réseau. Pour mieux saisir ce phénomène, étudions l'exemple suivant :

3.5.4.3.0.1 Exemple : S_a diffuse sa requête et temporise $3 * D_{a,b}$ (S_b est le site source de la donnée requise) avant de réémettre la même requête. La requête parvient à S_a au bout de $D_{a,b}$. S_a , qui est le site le plus proche de S_b , temporise $D_{a,b}$ puis retransmet la donnée requise. Tout site en instance de retransmission de la même donnée reçoit la réponse de S_a accompagnée de la période d'inhibition des requêtes multiples ($5 * D_{a,b}$). Suite à un congestionnement du réseau, le temps de transfert entre S_a et S_b est multiplié par 5. $6 * D_{a,b}$ unités de temps se sont donc écoulées depuis l'émission de la requête par S_b qui entre temps a retransmis sa requête. Le doublon va parvenir à S_a et aux autres sites alors que la première réponse de S_a est en cours d'acheminement vers S_b . Aucune retransmission n'est utile dans ce cas, et le mécanisme adopté n'effectuera pas de retransmission. L'algorithme qui suit résume le cas de la retransmission d'une donnée suite à une requête doublon:

3.5.4.3.0.2 Algorithme :

```

si Horloge_ $S_a$  expire après réception de requête de  $S_b$ 
  alors  $S_a$  envoie paquet réponse
     $S_a$  temporise  $5 * D_{a,x}$  avant le traitement d'une requête doublon
     $S_i$  qui reçoit réponse de  $S_a$  positionne son horloge d'inhibition à  $5 * D_{a,x}$ 
fsi

si  $S_i$  reçoit requête de  $S_j$ 
  alors si  $S_i = S_a$ 
    alors si  $S_j = S_b$ 
      alors si  $(3 * D_{a,b} < \text{Horloge\_}S_a \leq 5 * D_{a,x})$ 
        alors retransmettre donnée requise
        sinon ignorer requête doublon
      fsi
      sinon si  $\text{Horloge\_}S_i > 5 * D_{i,x}$ 
        alors retransmettre donnée requise
        sinon ignorer requête doublon
      fsi
    fsi
  sinon si  $S_i$  a reçu la requête de  $S_b$ 
    alors si  $S_j = S_b$ 
      alors si  $\text{Horloge\_}S_i > 5 * D_{i,x}$ 
        alors ignorer requête doublon
        sinon retransmettre donnée requise
      fsi
    fsi
  sinon si  $\text{Horloge\_}S_i > 5 * D_{i,x}$ 
    alors ignorer requête doublon
    sinon retransmettre donnée requise
  fsi
fsi

fsi

```

3.5.5 Terminaison de transfert et consensus unanime

le transfert d'un message est terminé lorsque toutes les entités FMP du groupe ont reçu ce paquet. Cet événement est appelé : terminaison de transfert du message. La zone tampon associée à ce message peut être libérée. Le délai de terminaison de transfert conditionne la charge des entités FMP en terme d'utilisation des ressources tampons. Il est donc important de signaler la terminaison de transfert le plus tôt possible pour éviter le phénomène de congestion des récepteurs qui est, pour l'essentiel, à l'origine des pertes de paquets.

Le mécanisme de fiabilité implanté accomplit la livraison unanime (ou consensus unanime) de manière implicite. En effet, aucun délai maximal de terminaison de transfert n'est fixé au préalable. Le protocole assure uniquement, qu'à tout message perdu est associé un mécanisme de requête et de recouvrement dont la terminaison varie selon la distance tem-

porelle séparant le site retransmetteur du point de défaillance. Toutefois, un ajustement au niveau du protocole peut être réalisé pour permettre une livraison unanime explicite. Pour ce faire, la transmission du message au niveau applicatif est différée durant un laps de temps suffisant à la réception d'une éventuelle requête de retransmission. L'exemple suivant décrit la variante proposée :

3.5.5.0.3 Pertes perceptibles :

Soit un site A qui diffuse deux messages m_1 et m_2 . Un site B ne reçoit que le message m_2 . Un site E qui reçoit les deux messages, temporise un laps de temps avant de remettre les messages reçus. Ce laps de temps doit permettre à la requête sur m_1 d'atteindre E avant sa livraison des messages m_1 et m_2 . Dans le cas où aucune perte de message n'est survenue dans la diffusion de m_1 et m_2 , ces derniers sont livrés à l'application à l'expiration de T .

Dans le cas de perte cité ci-dessus, T ne doit expirer qu'après, l'expiration du temps cumulé de l'horloge de requête de retransmission de B , correspondant à $D_{A,B}$, de $D_{E,B}$, correspondant au temps d'acheminement de la requête jusqu'à E , et d'une constante Cte , correspondant au temps de traitement du message m_1 par B et au traitement de la requête de retransmission par E , soit :

$$- T = D_{B,A} + D_{B,E} + Cte, \text{ où :}$$

Comme E ne connaît pas à priori, l'identification du site ayant perdu les message, ni la distance de ce site par rapport à l'émetteur A , le mécanisme de stabilité proposé est basé sur le calcul d'un temps T_{max} supérieur ou égal à T et qui assure le même rôle.

Le calcul de T_{max} repose sur les mêmes hypothèses émises dans 3.5.3.3, à savoir des conditions de congestionnement des réseaux traversés par les paquets échangés, équivalentes. De même pour les topologies et débits de ces réseaux.

Soit S le site le plus éloigné de E . A partir des hypothèses précitées, nous pouvons écrire :

$$\begin{aligned} - D_{E,S} &\geq D_{E,B} \\ - D_{B,E} + D_{E,A} &\geq D_{B,A} \quad (1) \end{aligned}$$

$$\text{D'où la nouvelle équation : } T \leq D_{E,S} + D_{B,A} + Cte \quad (2)$$

La précaution de considérer la distance $D_{E,S}$ plutôt que $D_{E,B}$ vient du fait que E ne connaît pas l'identification du site ayant perdu les message. De ce fait, E se réfère au site qui lui est le plus éloigné. Dans les conditions fixées, la distance $D_{E,S}$ inclut $D_{E,B}$.

L'inequation (2) reste vérifiée dans les deux cas suivants :

- B est le seul site à manquer la réception de m_1 :

$$(1) \text{ devient : } D_{S,E} + D_{E,A} \geq D_{B,A}$$

$$\text{d'où : } D_{S,E} + D_{S,E} + D_{E,A} \geq D_{B,A} + D_{E,B}$$

$$\text{ce qui donne : } T \leq 2 * D_{S,E} + D_{E,A} + Cte$$

$$T_{max} = 2 * D_{S,E} + D_{E,A} + Cte$$

- B est le site le plus proche de A à manquer la réception de m_1 :

Deux cas peuvent se présenter :

- $D_{E,A} \geq D_{B,A}$

$$\text{on en déduit que : } D_{E,S} \geq D_{E,A} \geq D_{B,A}$$

$$\text{d'où : } T_{max} = 2 * D_{S,E} + Cte$$

- $D_{E,A} < D_{B,A}$

$$\text{comme } D_{B,A} \leq D_{B,S} + D_{S,A}, \text{ on en déduit que :}$$

$$D_{B,A} \leq 2 * D_{S,E}$$

$$\text{d'où la nouvelle valeur de } T_{max} : T_{max} = 2 * D_{S,E} + Cte.$$

Nous avons vu que dans tous les cas, le temps $T_{max} = 2 * D_{S,E} + D_{E,A} + Cte$ est suffisant pour que la remise des messages m_1 et m_2 reçus par tout site E du groupe ne précède pas la réception par E d'une requête sur m_1 ou m_2 .

Notons que ce mécanisme peut très bien être utilisé pour implémenter une *fiabilité majoritaire* parmi les sites récepteurs. En effet, à l'expiration de T_{max} , un site peut considérer que la plupart des récepteurs corrects ont reçus la liste de messages sur laquelle porte T_{max} et peut, ainsi, délivrer la liste reçue.

3.5.5.0.4 Pertes non perceptibles :

Considérons maintenant le cas d'une défaillance non perceptible par un site (B par exemple). Cela survient lorsqu'aucun des messages diffusés par A n'est réceptionné par B , ou lorsque B reçoit une séquence de messages de A , croissante mais non complète. Quel doit être alors le comportement des autres sites ayant correctement reçu les messages de A (m_1 et m_2)? Une nouvelle technique alternative à la première peut être proposée :

L'idée est que le site E temporise un temps T initialisé à $D_{E,A}$ ¹⁵ puis diffuse un message de contrôle, assimilé à un ACK, comportant le plus grand numéro de séquence de la série

15. pour éviter un effet d'implosion des messages d'acquittement

reçue de A (m_2). Ce message de demande d'acquiescement concerne les seuls sites n'ayant pas reçu le dernier message de A . Le site E positionne par la suite T à $2 * D_{E,S}$, en prévision de la réponse du site le plus éloigné par rapport à E . Lorsque la demande d'ACK de E est réceptionnée par un site ayant correctement reçu m_1 et m_2 , celui-ci l'ignore et annule son T_{max} relatif aux messages reçus de A . Par contre, le site B vérifie qu'il n'a pas une horloge de requête activée (dans le cas où A a diffusé de nouveaux messages avant l'arrivée du message d'acquiescement de E). Si l'horloge est active, la demande de E est ignorée. Si l'horloge est inactive, B entame un processus de demande de retransmission sur les messages perdus. A l'expiration du nouveau T , le site E délivre les messages m_1 et m_2 car, soit aucune perte concernant ces messages n'a eu lieu, soit, en cas de perte, celle-ci a été signalée aux sites concernés qui ont initié un processus de demande de retransmission qui aboutira, soit enfin, le site B n'a pas reçu la demande d'ACK de E , ni les messages de A et il est considéré, dans ce cas, comme défaillant¹⁶.

3.5.6 Reprise après défaillances de sites ou partitionnements

La conception de FMP repose sur un service de diffusion avec meilleur effort, tel que le Multicast IP. Dans ce service, les membres peuvent rejoindre et quitter les groupes de manière autonome et transparente. FMP ne devrait donc pas intégrer des mécanismes de détection et de reprise en cas de partitionnement ou de défaillance, sauf si les besoins de l'application l'exigent. Autrement dit, notre service de livraison fiable ne fait pas la distinction entre un départ volontaire d'un site membre, une panne matérielle de ce dernier ou un partitionnement du réseau impliquant ce même membre. FMP donne uniquement un moyen, grâce aux mécanismes de détection et de retransmission de pertes, mais aussi un moyen périodique, grâce aux paquets état, de détecter les défaillances. Pour les besoins des applications sensibles aux défaillances, ces moyens fournis par le service fiable FMP, peuvent être utilisés par des protocoles de fiabilisation de systèmes pour implémenter un mécanisme de recouvrement. Ce mécanisme est implémenté au dessus du service de livraison fiable proposé.

3.6 Service de livraison totalement ordonné (TODS)

3.6.1 Principe de TODS

La classe des protocoles à contrôle centralisé spécifie que tous les messages du groupe transitent par un serveur d'ordre qui assume la fonction de numération. Du fait de la

¹⁶. récepteur incorrect, non concerné par la propriété de consensus unanime.

convergence des messages vers le serveur d'ordre, ce type de protocoles présente le risque de goulot d'étranglement au niveau du séquenceur. Pour prévenir le risque de défaillance du séquenceur, le protocole TRP [CM84] utilise un anneau virtuel qui vise à répartir le rôle de serveur d'ordre entre les membres du groupe. Ce protocole présente une lourdeur liée à la construction et à la gestion de l'anneau. Dans la description du service de livraison fiable, nous avons montré que les propriétés réalisées par la structure d'anneau, telles que la résistance aux défaillances, la suppression de pertes et le respect de l'ordre, peuvent être réalisées par un schéma d'acquiescement négatif. Cela nous a conduit à implanter le service TODS au dessus du RDS qui assure les rôles de transport fiable des messages de QoS atomique et de détection de défaillances. Pour assurer la livraison globalement ordonnée, le service TODS intègre la technique du serveur d'ordre unique.

3.6.2 Fonctionnement normal du service

La diffusion atomique comprend trois phases : la diffusion d'un message vers ses destinataires, la détection de la stabilité du message et, enfin, la délivrance ordonnée du message à ses destinataires. Nous avons vu que le schéma d'acquiescement négatif ne réalise pas la stabilité, c'est-à-dire la connaissance d'un site de la réception correcte par tous les sites destinataires du message qu'il a diffusé. Nous avons, également, décrit dans comment ce problème est contourné en introduisant la notion de paquets état. Néanmoins, cette solution n'est pas sans inconvénients : le temps nécessaire à la stabilité d'un message peut être considérablement augmenté du fait de la période de temps séparant deux paquets état (diffusés durant les périodes d'inactivité de la session) qui de surcroît sont assujettis aux pertes. Pour cela, nous allons utiliser le schéma d'acquiescement positif pour accélérer la stabilité des messages de QoS atomique.

Le protocole de diffusion atomique que nous proposons, appartient à la famille des protocoles à contrôle centralisé. La livraison atomique utilise les NACKs pour assurer un service fiable et les ACKs pour assurer la stabilité des messages de QoS totalement ordonnée.

Tout paquet de QoS atomique comporte un champ d'exclusion mutuelle (ou verrou). Ce message est reçu par tous les sites corrects du groupe qui l'insèrent dans leurs files de réception respectives (*DataQueue*). Le site séquenceur, unique site à détenir la clé relative au verrou, place, quant à lui, ce paquet dans une file FIFO particulière, appelée *OrderQueue*. Son rôle est déterminant dans l'attribution de numéros ordre globaux. La clé est attribué par le processus créateur et gestionnaire de la session de groupe. Le service de diffusion fiable RDS assure, grâce au mécanisme de gestion et de recouvrement de pertes, la bonne réception du paquet par tous ses destinataires (particulièrement le séquenceur).

L'occurrence de l'un des deux événements suivant provoque la diffusion par le site séquenceur d'un paquet signifiant l'ordre de délivrance des paquets reçus : le premier événement est l'expiration d'une horloge associée a file *OrderQueue*, le second est la saturation de cette dernière liste. Ce paquet, diffusé avec la QoS fiable, est désigné par le terme de paquet ACK. En effet, il acquitte les paquets de QoS atomique selon leur ordre de réception par le séquenceur. Une fois le paquet ACK diffusé, le site séquenceur transfère le contenu de *OrderQueue* dans la file *DataQueue* pour les besoins du service RDS qui assure la retransmission des paquets perdus. Les sites qui reçoivent un paquet ACK, délivrent les paquets stockés dans leurs *DataQueue* respectives suivant l'ordre communiqué par le séquenceur. L'ordre de réception du séquenceur constitue, ainsi, l'ordre absolu selon lequel les paquets seront délivrés aux processus de l'application.

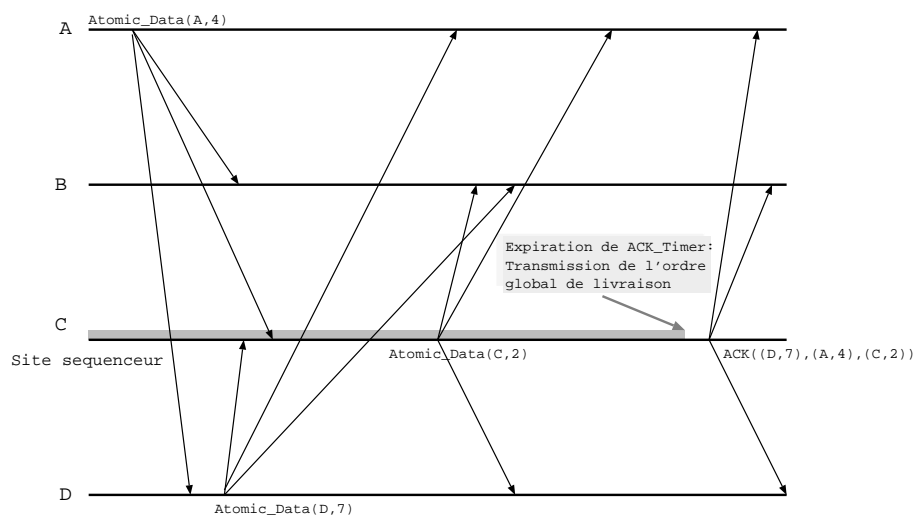


FIG. 3.6 - Exemple de livraison atomique

3.6.3 Fonctionnement lors de défaillances

Un paquet perdu sera requis selon le mécanisme de recouvrement décrit dans RDS. Le séquenceur classera le paquet perdu dans sa file *OrderQueue* une fois que ce dernier a été retransmis. Dans l'attente du paquet requis, le séquenceur bloque l'ordonnancement des paquets dont le numéro de séquence est supérieur au numéro du dernier paquet de la séquence perdue. Ces paquets seront stockés dans une file ...

A cause de l'asynchronisme inhérent aux systèmes répartis, l'arrivée d'un paquet ACK peut précéder celle d'une partie ou de la totalité de la séquence de paquets acquittés. Pour la même raison, la réception d'un ACK acquittant une séquence de paquets peut précéder celle d'ACK dont le numéro de séquence est antérieur. Dans le premier cas, les paquets acquittés qui peuvent être délivrés le sont. Le paquet ACK est ensuite inséré

dans file d'attente spécifique aux paquets ACK (*ACKQueue*). Toute nouvelle livraison ordonnée est bloquée jusqu'à réception ou recouvrement des paquets perdus. Lorsqu'un des paquets retardés ou perdus est reçu, la file *ACKQueue* est parcourue pour établir l'ordre de livraison de ce paquet. Lorsque tous les paquets de la liste ont été délivrés, le paquet ACK correspondant est supprimé de la file *ACKQueue*. Dans le deuxième cas, le paquet ACK reçu est sauvegardé dans la file *ACKQueue*. Il ne sera traité que lorsque deux conditions seront réunies : le paquet ACK qui le précède a été reçu et tous les paquets de la liste qu'il transport ont été délivrés. Dans les deux cas, une demande de retransmission sur les paquets ou ACKs non reçus est diffusée, selon le procédé préalablement décrit.

3.6.3.0.0.5 Défaillance du séquenceur :

Etant donné que le site séquenceur n'est pas exclusivement dédié à l'ordonnancement total, le protocole peut être rendu tolérant à une défaillance du séquenceur par l'élection d'un nouvel ordonnanceur parmi les sites du groupe. L'algorithme d'élection n'a pas été spécifié. Il devra, par exemple, choisir le site dont le dernier numéro de séquence de la file de délivrance *OrderQueue* est le plus élevé. Le procédé de détection de la défaillance du séquenceur (S_{token}) tient compte du fait, que l'identité du séquenceur est connue par les sites du groupe (elle accompagne tout paquet ACK). La détection est donc réalisée de la manière suivante : tout site S_i qui diffuse un message m de QoS atomique, positionne une horloge relative à ce message. La valeur de cette horloge est déterminée par la formule suivante :

$$- T_{failure} = 5 * (2 * D_{i,token} + T_{Queue}), \text{ où :}$$

- $2 * D_{i,token}$ correspond au temps nécessaire à un paquet pour faire l'aller-retour entre S_i et S_{token} ,
- 5 est une valeur aléatoire par laquelle le temps estimé de réception de l'ACK relatif à m est multiplié. Il s'agit, en fait, d'une garantie en cas de congestionnement du réseau séparant S_i et S_{token} .

Si à l'expiration de $T_{failure}$, le paquet ACK qui acquitte et estampille m n'est pas reçu, S_{token} est considéré comme défaillant. S_i initie alors une phase d'élection d'un nouveau séquenceur.

3.7 Service de livraison causale (CDS)

L'ordre causale stipule qu'un message m ne peut être remis à l'application que si tous les messages qui appartiennent au passé de m ont été remis à cette application. Par définition un message appartient au *passé causal* (ou ait historique) de m s'il a été remis avant l'émission de m .

Tous les protocoles de diffusion causale étudiés se déroulent en une phase : dès qu'un site S_i reçoit un message m dont le passé causal contient uniquement des messages déjà remis par S_i , alors S_i remet m . Cette remise immédiate peut en présence de défaillances être à l'origine d'un blocage. Considérons l'exemple suivant : S_i diffuse un message m_1 qui est reçu uniquement par S_j . S_j diffuse à son tour un message m_2 dépendant de m_1 . S_i et S_j défont par la suite. Aucun des sites du groupe de diffusion, à la réception de m_2 ne peut obtenir le message m_1 . Ces processus récepteurs sont alors bloqués. Un site S_i ayant délivré un message m peut, donc, être amené à le retransmettre tant qu'il existe un site S_j n'ayant pas reçu m . Il est, donc, nécessaire que S_i garde une copie des messages délivrés en prévision d'une éventuelle retransmission. Toutefois, le protocole doit permettre à un site de savoir quand la copie d'un message peut être détruite.

3.7.1 Motivations et définitions

L'étude des protocoles de diffusion causale existants (2.3.2.4) a permis de distinguer trois familles de protocoles, selon la représentation du passé causal d'un message diffusé :

- Dans la première famille, un message est transporté avec son historique, ainsi que le contenu de tous les messages appartenant à son historique.
- Dans la deuxième famille, un message est diffusé uniquement avec son historique.
- Dans la troisième famille, un message est diffusé avec un temps logique représenté par un vecteur temps ou une matrice temps.

Ces trois familles de protocoles se caractérisent par au moins l'un des inconvénients suivants :

- Le coût élevé en terme de tailles des messages. Les messages transportent une quantité importante d'informations additionnelles afin d'assurer la livraison causale. Cet inconvénient est illustré par les protocoles de [BJ87b] 2.3.2.4.1 [BSS91] 2.3.2.4.2 et [RST91] (2.3.2.4.3).

- Le calcul complexe qu’il est nécessaire d’effectuer à la réception d’un message afin de déterminer s’il peut être délivré ou non. Ceci est le cas, par exemple, pour les protocoles [PBS89] et [RST91].
- La taille mémoire importante nécessaire au stockage des messages (comme dans [BJ87b]) ou pour stocker les structures de données manipulées par le protocole, comme c’est le cas dans [PBS89].
- La redondance des traitements : ceci est le cas lorsqu’il arrive fréquemment que le récepteur de plusieurs messages soit informé plusieurs fois d’un même événement (émission ou livraison de messages), et ce, à travers les informations de contrôle contenues dans les différents messages qu’il reçoit. Cette redondance se traduit inévitablement par une perte plus ou moins importante de performances.

3.7.1.0.0.6 Hypothèses sur les défaillances Un autre aspect important qui n’a été que partiellement, ou pas du tout, pris en considération par les protocoles proposés, concerne le fonctionnement du protocole lors d’omissions, de défaillances ou de changements dans la composition du groupe.

Dans le protocole Psync [PBS89], les fonctionnalités de détection de défaillance, de mise à jour cohérente de la vue d’une session sur adhésions/retraits ou sur défaillances, sont implémentés par des protocoles de niveau supérieur. De plus, Psync suppose que l’ensemble des processeurs (ou sites) est connu, ce qui ne peut être envisagée si le service réseau utilisé est le protocole IP. Quant au protocole Fast CBCAST [BSS91], il suppose que les délais de communication sont bornés. Il est implémentée au dessus d’un service de communication fiable, tel que TCP, garantissant la non perte et le séquençement des messages échangés. Le protocole tolère la défaillance (définitive) d’un seul site par groupe de diffusion. Ces hypothèses adoptées ne sont pas adaptées aux systèmes asynchrones tolérants aux fautes. Enfin, le protocole implémenté dans [RST91] supposent les processeurs et le canal de communication fiables.

Le *service de livraison causale* (ou CDS : Causal Delivery Service) proposé a pour objectif d’atténuer les inconvénients précités. Il suppose que les sites peuvent être sujet à des défaillances temporaires (omission) ou définitives (comme pour le service RDS, assimilées à des retraits). De même le réseau de communication est non fiable (peut être sujet à des partitionnements) et ne garantit pas le séquençement des messages émis par un site.

L’idée est de limiter les informations de contrôle transportées par chaque messages à celles permettant de déduire les messages délivrés par l’émetteur depuis sa dernière diffusion et non pas depuis le début du calcul comme c’est le cas dans plusieurs protocoles proposés

dans la littérature. Par exemple, si un processus P délivre un seul message m entre deux diffusions m_1 et m_2 , les protocoles de [RST91] et [BSS91] nécessitent que le second message diffusé par P (i. e., m_2) contienne respectivement un vecteur d'entiers de taille n ou une matrice d'entiers de taille $n * n$ (n , étant la taille du groupe), alors que l'information devant être transportée par m_2 peut se limiter à un couple de deux d'entiers représentant l'identificateur du processus émetteur du message m et l'estampille de celui-ci.

Le CDS introduit la notion d'*ensemble ordonné d'émetteurs* (ou OSS : Ordered Set of Senders). Un OSS est un ensemble de couples d'entiers sur lequel s'appliquent les deux opérateurs suivants :

- *Premier* : qui donne le premier élément d'un OSS,
- *Reste* : qui donne un ensemble OSS diminué de son premier élément, soit :

$$\text{Premier}(OSS) \cup \text{Reste}(OSS) = OSS$$

3.7.2 Fonctionnement normal de CDS

Le service CDS repose sur trois structures de données localisées au niveau de chaque site FMP S_i :

- un vecteur $MsgDeliv_i$ de dimension n , n étant la taille maximale du groupe que peut supporter le service CDS. Cette valeur n'est pas encore fixée. $MsgDeliv_i$ est initialisé à 0. Nous donnons les définitions suivantes, relatives au vecteur $MsgDeliv$:
 1. L'élément $MsgDeliv_i[j]$, ($j \neq i$) représente le nombre de messages diffusés par S_j et délivrés par S_i depuis le dernier message, de S_j , délivré.
 2. $MsgDeliv_i[i]$ est l'estampille du prochain message causal diffusé par S_i .
- un ensemble OSS local, appelé $LastSenders_i$, qui contient une liste non ordonnée de couples d'entiers. Chaque couple identifie un numéro de site FMP, suivi du numéro du dernier message causal délivré de ce site.
- $n \Leftrightarrow 1$ files d'attentes où la k -ème file $FILE_k$ sert au stockage des messages reçus qui sont en attente d'autres messages diffusés par S_k .

Pour diffuser un message m , S_i y insère son estampille locale ainsi que son ensemble $LastSenders_i$ représentant le passé causal du message. Une fois le message diffusé, $LastSenders_i$

est remis à \emptyset . Lorsqu'un site S_j reçoit le message m deux cas sont possibles :

- $LastSenders_i = \emptyset$, ce qui signifie que l'émetteur n'a délivré aucun message (en dehors du message reçu) depuis la diffusion de son dernier message. Dans ce cas, le message est soit délivré, si $MsgDeliv_j[i] = MsgDeliv_i[i]$, soit inséré dans $FILE_i$. La seconde possibilité sous-entend qu'un ou plusieurs messages de S_i , précédant causalement m sont en attente d'être délivré (soient qu'il n'ont pas été réceptionnés, soit qu'il sont bloqués dans une des files d'attente). A chaque fois qu'un message du site S_i est délivré, S_j tente de délivrer m si la condition $MsgDeliv_j[i] = MsgDeliv_i[i]$ est vérifiée, sinon m est remis dans $FILE_i$.
- $LastSenders_i \neq \emptyset$, ce qui signifie que m dépend causalement d'autres messages non diffusés par S_i . S_j utilise alors la liste $LastSenders_i$ ainsi que le vecteur $MsgDeliv_j$ pour déterminer les messages manquants dans le chemin de causalité de m dont la livraison est retardée jusqu'à ce que tous ces messages soient délivrés.

Lorsque S_j délivre le message m , soit l'élément $(S_i, MsgDeliv_j[i])$ est rajouté à l'ensemble $LastSenders_j$, si S_i n'était pas représenté dans la liste. Soit la variable $MsgDeliv_j[i]$ de l'élément $(S_i, MsgDeliv_j[i])$ est mise à jour pour être égale à l'estampille du nouveau message délivré. Lorsque m est délivré, S_j incrémente $MsgDeliv_j[i]$, en prévision d'une prochaine réception de messages de S_i . S_j parcourt ensuite la file de messages $FILE_i$. Les messages s'y trouvant sont délivrés si le seul message qui manque à leur passé causal est le message qui vient d'être délivré. Dans le cas contraire, ces messages sont transférés vers la file d'attente correspondant à l'émetteur du message manquant qui suit dans les listes $LastSenders$ qui leur sont associées. L'exemple suivant illustre parfaitement le mécanisme décrit.

3.7.2.0.0.7 Exemple : Considérons 4 sites A, B, C et D . Supposons que D reçoit le message $(a_2, (B, 5), (C, 1))$. A cet instant, le vecteur $MsgDeliv_D$ est à $(1, 4, 0, 2)$, le premier élément correspondant à A et le dernier à D . B ne délivre pas a_2 car il dépend causalement de messages contenus dans la liste $LastSenders$ qui l'accompagne. B effectue l'opération : $Premier>LastSenders_A (= b_5)$. Le premier site dont dépend a_2 est donc le site B . a_2 est inséré dans la file $FILE_B$ car $MsgDeliv[2] < 5$.

Supposons que D reçoit (b_5, \emptyset) . Ce dernier ne va pas être délivré car $MsgDeliv_D[2] = 4$, ce qui indique que le prochain message attendu de B est b_4 . b_5 est donc inséré dans la file $FILE_B$. Lorsque D reçoit (b_4, \emptyset) , celui ci est délivré et $MsgDeliv_D[2]$ est incrémenté. D parcourt ensuite la file $FILE_B$ pour délivrer les éventuels messages qui dépendent causalement de b_4 . D effectue l'opération : $Premier(FILE_B) (= a_2)$. a_2 est remis en queue

de file $FILLE_B$ car il dépend, en premier de b_5 . D effectue ensuite: $Premier(FILLE_B)$ ($= b_5$). Ce dernier est délivré car $MsgDeliv_D[2] = 5$. D parcourt de nouveau $FILLE_B$ et effectue $Premier(FILLE_B)$ ($= a_2$). D effectue, alors, $Premier(LastSenders_A)$ ($= b_5$) qui a été livré. Dans ce cas, D effectue $Reste(LastSenders_A)$ qui a pour résultat de supprimer le couple $(B, 5)$. D effectue à nouveau $Premier(LastSenders_A)$ ($= c_1$) qui n'a pas été encore livré ($MsgDeliv_D[3] = 0$). Par conséquent, $(a_2, (C, 1))$ est transféré vers la file $FILLE_C$ pour en être extrait puis livré, une fois que le message c_1 a été délivré.

3.7.3 Fonctionnement lors de défaillances ou de changements de vue

Le paragraphe qui suit décrit les mécanismes proposés pour permettre le fonctionnement de CDS lors de l'occurrence d'évènements externes à son fonctionnement, comme la défaillance temporaire d'un site émetteur ou récepteur (conduisant dans les deux cas à une perte de messages), l'adjonction de nouveaux membres ou, finalement, le départ de membres (une défaillance définitive est assimilé à un départ involontaire du groupe).

3.7.3.1 Fonctionnement lors de défaillances temporaires de sites

Pour assurer une livraison fiable, le service CDS repose sur un schéma de détection de pertes et de retransmission proche de celui implémenté dans le service de livraison fiable (RDS). Lorsqu'un site détecte la perte d'un message causal, il construit un paquet requête dans lequel un champ spécial (*RequestTyp*) spécifie le mécanisme de retransmission à utiliser par les récepteurs de la requête. Le site le plus proche de l'émetteur de la requête, construira son message de retransmission en fonction du contenu d'un champ spécial. Nous énumérons les spécificités par lesquelles le service CDS se distingue de RDS :

1. Les numéros de paquets attribués aux messages de QoS causale n'ont pas la même source de numérotation que les paquets de QoS fiable ou atomique. La raison est qu'à la différence de la livraison fiable où les messages sont remis sans condition à l'application, ou de la livraison atomique où l'ordre de livraison est établi puis diffusé par le site séquenceur, indépendamment des numéros de séquence et de l'ordre dans lequel des paquets originaires d'un même site ont été émis, l'ordre causal est un ordre partiel qui définit un ordre de livraison respectant l'ordre dans lequel les messages ont été délivrés par leur site émetteur. L'exemple suivant montre comment une numérotation commune entre les messages de QoS causale et les autres types de messages ne permet pas d'atteindre cet objectif.

3.7.3.1.0.8 Exemple : Soient quatre sites A, B, C et D . Supposons que B diffuse trois messages de QoS fiable (b_6, b_7 et b_8) entre deux messages de QoS causale b_5 et b_9 . A reçoit les deux messages puis diffuse $(a_3, (B, 9))$, indiquant le dernier message causal reçu de B . D manque la réception de b_5 , mais reçoit b_9 puis a_3 . D ne peut pas délivrer b_9 (et par conséquent a_3 non plus). La raison est que D ne peut savoir au préalable si le message b_5 perdu est de QoS causale ou autre. D retarde alors la livraison de b_9 jusqu'à la réception de b_5 , ce qui peut s'avérer inutile dans le cas où b_5 est de QoS fiable uniquement. De même, si D manque b_7 , mais reçoit b_5 et b_9 , la délivrance de a_3 sera retardée, car D ne connaît pas, au préalable, la QoS de b_7 . La représentation adoptée pour l'ensemble $LastSenders_A$ ne permet pas, en effet, à D de savoir quels sont les messages, antérieurs à b_9 , desquels dépend la livraison de a_3 . Ce retard dans la livraison affecte les performances du service CDS et nous fait opter pour une numérotation spécifique aux messages de QoS causale. Dans ce cas, les numéros des messages b_5 et b_9 seraient consécutifs, permettant, ainsi, à tout récepteur de décider sans ambiguïté de la délivrance des messages reçus (tel que cela a été décrit dans 3.7.2).

2. Le mécanisme de requête et de retransmission relatif à CDS repose également sur le procédé de *slotting/damping* implémenté dans le cadre du service de livraison fiable (3.5). La différence avec le service RDS réside dans la composition des paquets de retransmission et son effet sur le mécanisme de recouvrement de pertes. Contrairement au service RDS, un paquet de retransmission de message causal, ne contient pas l'identifiant du site retransmetteur. Tout site qui répond à une demande de réparation sur un message causal, répond en retransmettant le message tel qu'il a été reçu (y compris l'ensemble $LastSenders$ de l'émetteur), sans aucune information additionnelle. Pour cela, tout message reçu est stocké dans une file de retransmission qui est mise à jour à chaque réception d'un message état (voir 3.5.1.3). La raison de ce choix est, principalement, de simplifier le mécanisme de réparation implémenté dans le cadre de RDS et qui, appliqué à CDS, conduirait à une lourdeur incompatible avec l'objectif recherché, à savoir fournir un service de livraison causale et fiable, et non pas accélérer le processus de détection et de correction de pertes comme c'est le cas dans RDS. De plus, le fait que chaque message transporte son passé causal, est un moyen supplémentaire de détection de pertes qui permet à CDS de se passer de la numérotation des messages réponse.
3. L'ensemble $LastSenders$ qui accompagne tout message causal est un élément supplémentaire de détection de perte. Dans l'exemple précédent, si le site D reçoit le message $(a_3, (B, 9))$ alors que $MSgDeliv[A] = 5$, D initialise son horloge de requête à $D_{D,B}$, à l'expiration de laquelle il diffuse une requête de retransmission sur les

messages b_6, b_7 et b_8 . Si, à présent, D reçoit $(c_1, (B, 8))$, il n'initie pas de processus de requête, car ce paquet a déjà été requis.

3.7.3.2 Fonctionnement lors de défaillances définitives ou de retraits de sites

Lorsqu'un site quitte le groupe ou subit une défaillance définitive, le site séquenceur met à jour puis diffuse la nouvelle vue de groupe. Quelle doit être, alors le comportement du site séquenceur, mais aussi celui d'un site du groupe à la réception de la nouvelle liste?.

Pour éviter une saturation des vecteurs $MsgDeliv$, le site séquenceur devrait affecter le GId du site partant à un nouveau site du groupe de diffusion causale. Cependant, il faut qu'il s'assure au préalable qu'aucun message estampillé par le site partant (soit, S_j) n'est attendu par les autres sites. Une façon d'atteindre cet objectif est que tout site S_i qui reçoit la nouvelle liste, parcourt ses files $FILE_k, k = 1..n, k \neq i$ pour détecter tout message en attente de délivrance qui contient dans son passé causal ($LastSenders_i$) un message provenant de S_j . Le mécanisme de recouvrement de pertes assure que les messages originaires de S_i et figurant dans le passé causal des messages détectés, ont été requis. Si la recherche dans les files $FILE_i$ aboutit, S_i déclenche une horloge positionnée à $D_{i,x}$, S_x étant le site le plus éloigné de S_i . A l'expiration de l'horloge, S_i parcourt à nouveau les files $FILE_i$ pour supprimer toute référence à site S_j dans le passé causal des messages détectés et provenant d'autres sites que S_j . Quant aux sites originaires de S_j et ne dépendant causalement que de messages provenant du même site, ils sont supprimés de la file $FILE_j$. Par contre, les messages des files $FILE_k$ originaires de S_j et dépendant de messages ne provenant pas de S_j , seront livrés une fois que leur ensemble $LastSender$ sera vide. Une fois que ces derniers messages seront livrés, S_i peut reinitialiser la valeur de $MsgDeliv_i[j]$ à 0, pour recevoir le premier message diffusé par un nouveau site auquel le séquenceur aura attribué le GId de S_j (GId_j).

Cependant, à la suite de mises à jour précitées, S_i peut recevoir un message de S_k transportant dans $LastSenders_k$ une référence à S_j après le départ de ce dernier. S_j ne dispose d'aucun moyen pour déterminer si la référence à S_j est relative à l'ancien ou au nouveau site. Devant cette ambiguïté, nous avons préféré ne pas attribuer le GId d'un site partant, même si cela présente le risque de saturation de l'ensemble des GId attribués par le séquenceur, en cas de départs ou d'adhésions trop fréquentes.

3.7.3.3 Fonctionnement lors d'adhésion de membres

Lorsqu'un processus utilisateur invoque pour la première fois le service CDS du site FMP local S_i , ce dernier diffuse un message contenant un champ d'exclusion mutuelle qui permet

au seul site séquenceur de le traiter. Le message émis est en fait une requête d’attribution d’un numéro de site global qui sera utilisé comme identificateur du site lorsque celui-ci diffusera des messages sollicitant le service d’ordonnancement causal. Le site jeton répond à la requête en retournant un identificateur global (ou GId) attribué. Pour émettre le GId , le séquenceur utilise le service de transport unicast TCP. Le site jeton incrémente par la suite le numéro d’ordre local pour une nouvelle attribution. L’incréméntation des numéros GId est limitée au nombre maximal de sites que peut supporter le service CDS. Actuellement ce nombre a été fixé, sans une étude de performance préalable. Si ce nombre est atteint, toute requête d’attribution de GId est suivie d’une réponse négative qui ne permet pas au site requéreur de diffuser des paquets de QoS causale. Le service CDS du site S_i diffuse, alors, la donnée remise par l’utilisateur en plaçant, en plus de son adresse IP, le GId attribué dans le message diffusé. Le GId de l’émetteur (GId_i) sera utilisé par le site récepteur (soit, S_j) pour accéder à l’élément du vecteur $MsgDeliv_j[GId_i]$. De même, avant de diffuser un message de QoS causale, S_j utilise GId_i (ainsi que tous les $GIds$ reçus des sites dont S_j a délivré des messages) pour construire son ensemble $LastSenders_j$.

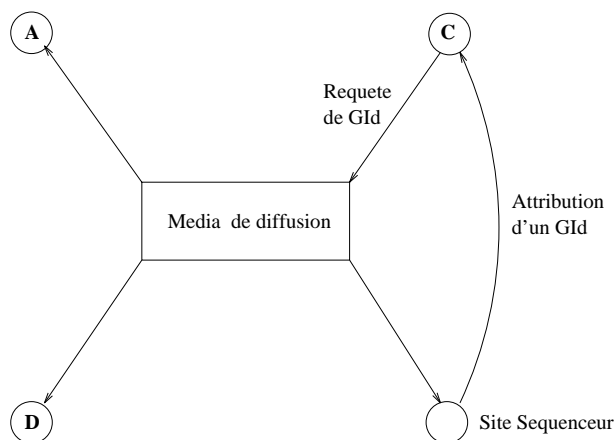


FIG. 3.7 - Attribution d’un numéro GId

Les inconvénients de cette première solution résident, d’une part, dans le risque de défaillance du site séquenceur. D’autre part, dans les “trous” ou lignes vides qui peuvent apparaître dans la structure de données $MsgDeliv$. La défaillance du site séquenceur est gérée, soit en prévoyant un site de reprise sur lequel toutes les informations du site séquenceur sont dupliquées, soit par un mécanisme d’élection d’un nouveau site séquenceur. Dans ce dernier cas, le site élu diffusera une requête à laquelle chaque site du groupe répond en renvoyant son GId . Pour cela, chaque site utilise le service de transport TCP, car l’information retournée ne concerne pas les autres sites. Pour éviter l’effet d’implosion des réponses au niveau du site séquenceur, chaque site S_i temporise un laps de temps égal à $D_{i,token-site}$ avant de retourner son GId . Une fois les $GIds$ collectionnés, le nouveau site séquenceur positionnera son numéro d’ordre au plus grand GId retourné, augmenté de

un.

Une autre solution alternative au site séquenceur, est l'utilisation de *tables de hachage*. Localement, chaque site gère une table de hachage permettant d'attribuer un *numéro local unique (LId)* correspondant à l'adresse IP du site émetteur d'un message d'ordre causal. A la différence du premier mécanisme décrit, le LId_i calculé par un site S_j relativement à un émetteur S_i , ne peut être utilisé que pour l'accès à l'élément $MsgDeliv_j[LId_i]$. Pour construire l'ensemble $LastSenders_j$, S_j y insère les numéros IP des sites émetteurs et non pas leurs $LIds$, étant donné que ces identificateurs sont locaux à S_j . Pour la même raison, l'identificateur du site émetteur de m_j se limite au numéro IP de S_j , LId_j étant un identificateur local. Lorsqu'un site S_k reçoit un message causal m_j de S_j , il calcule, non seulement, le LId correspondant à l'adresse de S_j , mais aussi les $LIds$ de chaque site mentionné dans $LastSenders_j$ diffusé avec M_j .

3.7.3.3.0.9 Avantages et inconvénients des deux solutions :

L'avantage du second mécanisme décrit est qu'il n'est pas dépendant du site séquenceur. La défaillance de ce dernier n'a donc aucune incidence sur l'attribution des $LIds$. Un autre avantage de ce mécanisme est apparent au niveau du vecteur $MsgDeliv$. La fonction de hachage attribue, en effet, les $LIds$ de manière séquentielle. Cela permet d'avoir une construction progressive de vecteurs $MsgDeliv$, tandis que le premier mécanisme crée des lignes correspondant aux $GIds$ transmis avec le message causal reçu. Supposons, par exemple, qu'un site S_i reçoit un premier message de S_j , dont le GId vaut 2, suivi d'un premier message de S_k dont le GId vaut 38. Le site S_i devra créer les vecteurs lignes 2 et 38, puis initialiser les valeurs d'un vecteur $MsgDeliv_i[38]$. D'un autre côté, le mécanisme des tables de hachage aurait généré un vecteur $MsgDeliv_i[3]$, avec la création des lignes $LId_{S_j} = 2$ et $LId_{S_k} = 3$. Cependant, comme l'attribution des GId par le séquenceur fait suite à une première diffusion causale du site requéreur, la non réception, par S_i , de $GIds$ compris entre 3 et 31, sous-entend un partitionnement du réseau ou un congestionnement du site S_i . Deux paramètres décident, alors, du comportement futur de S_i : le nombre de $GIds$ manqués et la politique de fiabilité adoptée par l'application. S_i devra, soit, diffuser, pour chaque GId manqué, une requête "spéciale"¹⁷ de retransmission, soit quitter le groupe, considérant qu'il a subi une défaillance trop longue ou que c'est un récepteur trop lent. Il est clair que le mécanisme des tables de hachage ne permet pas de détecter cette défaillance.

En outre, le mécanisme des tables de hachage présente deux inconvénients majeurs. Le

17. Le site le plus proche retransmet, pour chaque GId inclus dans le paquet requête, tous les messages d'ordre causal émis ou reçus, encore disponibles

premier est lié à la taille des messages de QoS causale, qui est élevée du fait de la nécessité de construire un ensemble *LastSenders* à partir d'adresses IP de sites, et non pas de *GIds* comme pour le premier mécanisme (32 bits pour une adresse IP et 8 bits pour un *GId*). Le second inconvénient est lié au fait que chaque récepteur de message causal, calcule systématiquement les *LIds* correspondant au site émetteur ainsi qu'à tout site mentionné dans l'ensemble *LastSenders* de celui-ci. Le second mécanisme est donc plus coûteux aussi bien en temps de traitement, qu'en taille des messages à délivrance causale, alors que le premier est sensible à la défaillance du site jeton. Dans notre recherche d'optimisation du flux de données et de la taille des messages, nous avons adopté le premier mécanisme pour sa convergence avec notre objectif.

3.7.3.4 Transmission du vecteur *MsgDeliv* à un nouveau membre

Comme pour le service de livraison fiable, un nouveau site FMP (soit, S_{new}) qui rejoint une session de groupe en cours, ne doit pas requérir la retransmission des paquets à livraison causale dont il a raté la réception. Pour cela, la réponse du site séquenceur à la requête d'attribution de GId_{new} , comprend aussi le vecteur *MsgDeliv* local ($MsgDeliv_{token_site}$). Il est, toutefois, plus adapté que le site transmetteur de la matrice *MsgDeliv* soit un site appartenant au réseau local du nouveau membre ou, du moins, au réseau local le plus proche. Il est, en effet, le plus à même à fournir un vecteur *MsgDeliv* proche de celui du site S_{new} si celui-ci était parmi les membres initiaux du groupe.

Cependant, pour garantir que seul le site FMP le plus proche réponde à la requête de S_{new} , chaque récepteur doit temporiser un délai correspondant à sa distance temporelle par rapport à l'émetteur de la requête. Or, en l'absence de diffusion de paquet état¹⁸ par S_{new} , cela n'est pas réalisable. Une fois que la distance $D_{i,new}$ est connue par tout site S_i du groupe, si S_{new} demande la retransmission d'un paquet causal dont il a manqué la réception, c'est le site S_j , le plus proche qui y répondra. S'il s'agit de la première requête¹⁹ de S_{new} reçue par S_j , ce dernier transmettra non pas le paquet causal requis mais son vecteur local $MsgDeliv_j$. Lorsque S_{new} recevra le vecteur $MsgDeliv_j$, il annulera sa requête et remplacera son vecteur $MsgDeliv_{new}$, initialement transmis par le site séquenceur, par $MsgDeliv_j$. De cette manière l'objectif du mécanisme proposé est atteint, à savoir, permettre à tout nouveau membre d'être synchrone, dans sa réception des messages à livraison causale, avec les membres de son réseau local ou, au moins, du réseau local le plus proche.

18. Le paquet état est utilisé pour le calcul des distances temporelles entre sites FMP (voir 3.5.1.3)

19. A la première invocation, la fonction qui construit les paquets requête sur un message causal, insère dans un champ spécial, un bit signalant que c'est la première requête

3.8 Conclusion

FMP est un protocole de niveau transport, conçu pour un environnement asynchrone où les délais de transmission ne sont pas bornés et où les processus de communication sont assujettis aux défaillances par arrêt ou par omission, alors que le réseau d'interconnexion peut être sujet aux partitionnements.

FMP fournit trois services de communication fiable sélectionnables à la demande. Le premier service assure une livraison fiable, le second assure une livraison atomique et le troisième assure une livraison respectant l'ordre causal établi entre les messages. Notre proposition a été motivé par le souci de fournir les qualités de service nécessaires pour permettre au protocole de supporter une aussi large que possible classe d'applications de groupe. En permettant la sélection de la qualité de service selon les besoins de l'utilisateur, nous avons voulu doter le protocole d'une flexibilité qui répond aux besoins subtiles de certains types d'applications, sans pour autant pénaliser les applications dont le besoin en service de communication est minimal.

Dans ce chapitre, nous avons décrit les mécanismes qui réalisent les trois services de diffusion fournis par FMP. Nous avons également décrit des mécanismes sous-jacents sur lesquels reposent les services décrits, tels que le contrôle de flux, la détection de défaillances, la terminaison de transfert, etc. Le chapitre suivant sera consacré à l'implémentation de FMP par l'intermédiaire d'un langage de spécification formelle.

Chapitre 4

Mise en oeuvre de FMP

4.1 Introduction

Différentes méthodes peuvent être utilisées pour implémenter des protocoles de communication. Elles vont de l'utilisation des langages naturels aux langages formels. L'utilisation de langages naturels a comme inconvénient de donner l'illusion de maîtriser la réalisation du protocole, mais peut conduire à une conception longue. De plus elle peut comporter des "ambiguïtés" qui rendent compliquée la phase de vérification de la complétude et de la correction du protocole [Bud92]. De ce fait, il est souhaitable d'utiliser une technique de description formelle (ou FDT : *Formal Description technique*) afin de simplifier la conception et de permettre la validation. Cette approche permet aussi de rendre l'implémentation et la procédure de tests plus efficace.

A la différence d'outils de spécification formelle orientés preuves formelles, tels que Lotos [?] ou SDL [SDL88], Estelle [ISO89] est connu pour être orienté implémentation. Il fournit, de ce fait, un outil d'implémentation réel tout en préservant l'aspect formel de la réalisation. Pour cela, et pour d'autres raisons précisées ultérieurement, notre choix a porté sur le langage Estelle pour implémenter le protocole FMP.

La section 1 traite des structures de données du protocole et des actions qui s'y rapportent. Le but recherché est de décrire les mécanismes sous-jacents aux services de livraisons fournis par FMP et que la description formelle ne précisera pas. La section 2 introduit la description formelle de FMP en représentant le protocole sous forme d'une machine à états finis. La section 3 donne un bref aperçu sur le langage Estelle. La section 4 est consacrée à la description formelle de l'architecture du système ainsi que des services fournis par le protocole. Dans la section 5, nous discutons notre choix sur l'outil FDT utilisé.

4.2 Structures de données et algorithmes

Différentes structures de données sont utilisées par les services du protocole FMP, nous allons présenter les plus pertinentes d'entre elles et décrire les algorithmes qui les manipulent.

4.2.1 Service RDS

Pour les besoins du service de livraison fiable, les structures de données suivantes sont utilisées :

4.2.1.1 La structure *StateTab*

Cette structure de donnée permet la mise à jour des informations relatives au site émetteur du paquet reçu, notamment, le prochain numéro de séquence attendu de cette source et le prochain numéro de séquence de cette source à délivrer.

Chaque élément de la table représente le début d'une liste FIFO d'éléments similaires. Lorsqu'un paquet (désigné par *ReceivPack*) de QoS atomique ou fiable est reçu, son entrée dans la table *StateTab* est référencée par le champ identificateur du site émetteur (désigné par *ReceivPack.SenderId*). Le sommet de la file correspondant à l'émetteur *SiteId* est, ainsi, désigné par *StateTab[ReceivPack.SenderId]*. L'ensemble des sommets de files constitue la table d'état contenue dans un message état diffusé. Un élément comprend trois champs : le premier représente un numéro de séquence (*SeqNum*), le second champ est un entier qui indique les numéros de séquence consécutifs, formant une suite décroissante de raison 1, à partir du numéro contenu dans le premier champ, *i.e.*, si les valeurs des champs 1 et 2 sont respectivement 7 et 3, l'élément de la table *StateTab* représente les paquets 7, 6 et 5. Le dernier champ composant est une adresse sur le prochain élément. Deux éléments adjacents sont le résultat d'une rupture de séquence dans la réception des paquets de l'émetteur correspondant à la file, *i.e.*, les éléments (7,3,next) et (2,2,nil) résultent de la séquence de réception suivante : 1,2,5,6,7.

Les algorithmes associés à *StateTab* permettent de détecter les pertes, de filtrer les paquets doublons et de construire la structure du paquet état.

4.2.1.1.1 Algorithme de détection de perte :

```

Function Loss_Detecetion()
  loss = flase
  Idf = StateTab[ReceivPack.SenderId]

  If (Idf.SeqNum leq (ReceivPack.SeqNum))
    loss = true
  End_If
  return(loss)
End_Function.

```

4.2.1.1.2 Algorithme de détection de paquets doublons :

Lors de la réception d'un paquet, cet algorithme a un double usage: la détection de paquets doublon et la mise à jour de la liste dont l'entrée dans *StateTab* est désignée par l'identifiant du site émetteur.

```

Function Update_StateTab()
  received = false
  Idf = StateTab[ReceivPack.SenderId]

  While ((Idf.SeqNum - ReceivPack.SeqNum) > Idf.Nbr)
    avancer dans la liste vers le prochain Idf
  End_While

  If ((Idf.SeqNum-Idf.Nbr) < ReceivPack.SeqNum ≤ Idf.SeqNum)
    received = true
  Else
    If (Idf.SeqNum leq (ReceivPack.SeqNum))
      insérer nouveau élément dans StateTab[ReceivPack.SenderId]
      New_Idf.SeqNum = ReceivPack.SeqNum
      New_Idf.Nbr = 1
    Else
      If (Idf.SeqNum == ReceivPack.SeqNum-1))
        Idf.SeqNum = ReceivPack.SeqNum
        Idf.Nbr = Idf.Nbr+1
      Else /* Idf.SeqNum = (ReceivPack.SeqNum+Idf.Nbr) */
        Idf.Nbr = Idf.Nbr+1
        If ((Idf.SeqNum-Idf.Nbr) == Next_Idf.SeqNum)
          Idf.Nbr = Next_Idf.Nbr
          supprimer l'élément Next_Idf de la liste
        End_If
      End_If
    End_If
  End_If
  return(received)
End_Function.

```

4.2.1.2 La structure *DataQueue*

Lorsqu'un paquet donnée de QoS fiable est reçu, il est mis dans une file FIFO pour une éventuelle retransmission. De même, un paquet de changement de vue ou un paquet donnée de QoS atomique est mis dans la même file pour une actualisation de la liste locale de groupe pour le premier et pour une délivrance ultérieure pour le second. Cette liste représente la structure de données *DataQueue*. Les trois opérations associées à cette liste sont :

1. opération d'insertion lors de la réception de paquets : les types d'éléments suivants sont concernés par les opérations d'insertion :
 - un paquet donnée ou un paquet réponse dont la donnée transportée n'a pas été reçue antérieurement.
 - un paquet spécial identifiant la liste des paquets qui ont été requis et non encore recouverts.

L'ordre d'insertion dans la liste ne comporte aucune priorité autre que l'ordre d'arrivée des paquets.

2. opération de suppression réalisée sur :
 - tout paquet donnée ou réponse qui a été acquitté par l'ensemble des sites connus par le récepteur.
 - tout paquet spécial dont l'ensemble des paquets référencés ont été reçus.
3. essai de délivrance de tout paquet qui en a les qualifications. La tentative de délivrance réussit si l'élément réunit certaines conditions basées sur le type de paquet et sur la QoS qui lui est attribuée.
4. opération de détection de requêtes multiples ou de requêtes doublons (émises deux fois par le même site). L'algorithme correspondant est donné plus bas.

Pour réaliser les opérations décrites ci-dessus, à chaque élément de la liste, est associé un *état* qui peut être :

- paquet reçu (*D*) : réception correcte du paquet. Il s'agit de paquets de QoS atomique qui doivent être délivrés à la réception du paquet ACK les acquittant,
- paquet délivré (*O*) : le paquet a été correctement reçu puis remis à l'application,

- paquet retransmis (*A*): le rôle de cet état est nécessaire pour le traitement des requêtes multiples. Il indique que la donnée stockée est un paquet retransmis par un autre site.
- liste de paquets requis (*R*): les paquets identifiés sont manquants et un NACK (ou requête de retransmission) a été envoyé trois fois.

Etat du paquet	Information contenue	Champs optionnels		
D,O	Paquet donnée	Champ Answer		Champ RequestSource
		valeur=0: paquet non retransmis	valeur=1: paquet retransmis	valeur=2: paquet retransmis par un autre site
A	Paquet donnée retransmis	Requete.Source		
R		DataSource: Site destinataire de la requête	First: First: premier paquet requis	Last: Last: dernier paquet requis

TAB. 4.1 - *SDD de la file de réception*

4.2.1.2.1 Algorithme de traitement des requêtes multiples :

Lors de la réception d'un paquet requête, la file *DataQueue* est parcourue pour déterminer d'une part, si un paquet réponse doit être retourné ou non, et d'autre part si un protocole de détection de défaillance doit être initié ou non. L'algorithme suivant s'applique à tous les éléments de la liste. Un élément de la liste est désigné par le terme *Item*. Le parcours s'arrête dès qu'un des traitements (issu de l'algorithme) correspondant à l'événement réception de paquet requête est effectué. Le principe de l'algorithme a été décrit dans 3.5.3.3.

```

Void Multi_Request()
  If (DataQueue.Item.State == D)
    If (DataQueue.Item.Answ ≠ 0)
      If (InhibitionTimer non expirée) // activée après émission du paquet réponse
        ignorer la requete reçue
        If (Request.Source == DataQueue.Item.ReqSrc)
          Détection de défaillance du site Request.Source
        End_If
      Else
        processus de retransmission du paquet requis
      End_If
    Else
      processus de retransmission du paquet requis
      mettre Request.Source dans DataQueue.Item.ReqSrc
      DataQueue.Item.Answ = 2) // indique que c'est le site courant qui a retransmis
    End_If
  Else
    If (DataQueue.Item.State == A)
      If (InhibitionTimer non expirée) // activée après réception du paquet réponse
        ignorer la requete reçue
        If (Request.Source == DataQueue.Item.ReqSrc)
          Détection de défaillance du site Request.Source
        End_If
      Else
        processus de retransmission du paquet requis
        mettre Request.Source dans DataQueue.Item.ReqSrc
      End_If
    Else
      If (DataQueue.Item.State == R)
        If (Request.Source == DataQueue.Item.To) and
          (DataQueue.Item.To.Min ≤ Request.SeqNum ≤ DataQueue.Item.To.Max)
          ignorer la requete reçue
        Else processus de retransmission du paquet requis
        End_If
      End_If
    End_If
  End_If
End_Function.

```

4.2.2 Service TODS

Deux listes FIFO sont utilisées par le service de livraison globale: *DataQueue* et *OrderQueue*. La première liste est commune aux services RDS et TODS. Elle sert de structure de stockage des paquets pris en charge par l'un des deux services, selon l'état du site récepteur. Un site non séquenceur y insère les paquets de QoS atomique pour une délivrance ultérieure. *DataQueue* est également utilisée pour les mécanismes de recouvrement de pertes. Par ailleurs, l'usage de la liste *OrderQueue* dépend de l'état du site récepteur et du type des paquets reçus. Le site séquenceur y insère les paquets de QoS atomique, alors qu'un site non séquenceur y insère les paquets ACK, de QoS fiable,

diffusés par le séquenceur. Chaque élément de chacune des deux types de liste comporte une priorité fixée par les estampilles sur les paquets.

L'ordre de réception du site séquenceur est transmis dans une structure de donnée linéaire (*DelivOrder*) dont chaque élément correspond à l'estampille du site source du paquet qu'il représente. Cette structure de donnée est complétée par deux informations : la taille de la structure (*DelivOrder_size*) et un pointeur entier (*next*), initialisé à 0 et désignant la prochaine estampille de la structure à traiter.

Les algorithmes associés aux structures décrites ci-dessus sont utilisés pour insérer, supprimer des éléments de la liste, pour la délivrance de paquets ou, enfin, pour la construction de paquets ACK. Chaque algorithme est basé sur l'état courant du site de traitement (site séquenceur ou site non séquenceur).

4.2.2.1 Traitement relatif au site séquenceur

Deux algorithmes s'excluant mutuellement manipulent simultanément les structures de données *DataQueue* et *OrderQueue*. L'exclusion mutuelle est réalisée par la technique des sémaphores. La variable d'exclusion mutuelle est initialisée à 0 (*verrou = 0*) :

- Lors de la réception d'un paquet avec la QoS atomique :

```

Void packet_reception()
  While (verrou == 1)
    attendre
  End_While
  verrou = 1
  If (OrderQueue pleine)
    construire paquet ACK à partir de OrderQueue
    diffuser paquet ACK
    transférer OrderQueue dans DataQueue
    mettre paquet reçu dans OrderQueue
    If (ACK_Timer active)
      désactiver ACK_Timer
    End_If
  Else
    mettre paquet reçu dans OrderQueue
    If (ACK_Timer non active)
      activer ACK_Timer
    End_If
  End_If
  verrou = 0
End_Function.

```

- Lors de l'expiration de *ACK_Timer* :

```

Void ACK_expiration()
  While (verrou = 1)
    attendre
  End_While
  verrou = 1
  If (OrderQueue vide)
    construire paquet ACK à partir de OrderQueue
    diffuser paquet ACK
    transférer OrderQueue dans DataQueue
  End_If
  verrou = 0
End_Function.

```

4.2.2.2 Traitement relatif à un site non séquenceur

4.2.2.2.1 La structure *ACKRefTab* : La structure de donnée *ACKRefTab* est une table de hachage maintenue au niveau de tout site non séquenceur. Elle attribue à toute estampille contenue dans *DelivOrder* et formée par le couple : identificateur de Site FMP et numéro de séquence, l'adresse, dans la liste *DataQueue*, du paquet ACK correspondant. Cette table est utile pour le traitement de messages dont la réception est postérieure à celle du paquet ACK les acquittant. Les deux algorithmes qui suivent, décrivent de quelle manière cette structure est utilisée :

- Lors de la réception d'un paquet de QoS atomique (*ReceivPack*) :

```

Void packet_reception()
  mettre le paquet reçu dans DataQueue
  If ((addr == ACKRefTab[ReceivPack.stamp]) ≠ NIL)
    If (addr.DelivOrder[addr.next] == ReceivPack.SeqNum)
      next = next + 1
      délivrer ReceivPack
      supprimer l'entrée ACKRefTab[ReceivPack.stamp]
      Repeat
        If (paquet estampillé addr.DelivOrder[addr.next]
            est présent dans DataQueue)
          next = next + 1
          délivrer le paquet
          supprimer l'entrée ACKRefTab[packet.stamp]
        Else
          sortir de la boucle
        End_If
      Until (addr.next == addr.DelivOrder_size)
      If (addr.next == addr.DelivOrder_size)
        supprimer de OrderQueue paquet ACK dont l'adresse est addr
      End_If
    End_If
  End_If
End_Function()

```

- Lors de la réception d'un paquet ACK :

```

Void ACK_reception()
  Repeat
    If (paquet référencé par ACKpacket.DelivOrder[ACKPacket.next]
        est présent dans DataQueue)
      délivrer paquet
      next = next + 1
    Else
      sortir de la boucle
    End_If
  Until (ACKPacket.next == ACKPacket.DelivOrder_size)
  If (ACKPacket.next == ACKPacket.DelivOrder_size)
    supprimer ACKPacket
  Else
    insérer ACKPacket dans OrderQueue
    addr = position de ACKPacket dans OrderQueue
    For (i = ACKPacket.next To ACKPacket.DelivOrder_size)
      ACKRefTab[ACKPacket.DelivOrder[i]] = addr
    End_For
  End_If
End_Function.

```

4.2.3 Service CDS

Deux types de données sont utilisées pour la livraison causale : il s'agit du vecteur *MsgDeliv* et de la file FIFO *FILE*. Le chapitre précédent (3.7.2) décrit de manière détaillée l'utilisation de ces deux structures pour assurer le service CDS. Dans ce paragraphe, nous reprenons les principes énoncés dans 3.7.2, en donnant l'algorithme d'ordonnement causal.

4.2.3.0.2 Algorithme de diffusion d'un paquet causal :

Soit le site S_i , en instance de diffuser un message m_i . S_i insère dans m_i :

- la valeur courante de son estampille : $MsgDeliv_i[i]$
- l'ensemble $LastSenders_i$ représentant l'historique des messages délivrés par S_i depuis sa dernière diffusion.

Après diffusion de m_i , S_i exécute :

- $MsgDeliv_i[i] = MsgDeliv_i[i] + 1$
- $LastSenders_i = \emptyset$

4.2.3.0.3 Algorithme de livraison causale :

Lorsqu'un site S_j reçoit le message $m_i(S_i, MsgDeliv_i[i], LastSenders_i)$:

```

Void Causal_Delivery( $m_i$ )
  If ( $LastSenders_i = \emptyset$ )
    If ( $MsgDeliv_j[i] = MsgDeliv_i[i]$ )
      délivrer( $m_i$ )
    Else
      insérer  $m_i(S_i, MsgDeliv_i[i], LastSenders_i)$  dans  $FILE_i$ 
    End_If
  Else
    If ( $MsgDeliv_j[i] = MsgDeliv_i[i]$ )
      Repeat
         $first = Premier(LastSenders_i)$ 
        If ( $MsgDeliv_j[first.SiteSrc] < first.SeqNumber$ )
          insérer  $m_i(S_i, MsgDeliv_i[i], LastSenders_i)$  dans  $FILE_{first.SiteSrc}$ 
        Else
           $LastSenders_i = Reste(LastSenders_i)$ 
        End_If
      Until ( $(LastSenders_i = \emptyset)$  or ( $MsgDeliv_j[first.SiteSrc] < first.SeqNumber$ ))

      If ( $LastSenders_i = \emptyset$ )
        délivrer( $m_i$ )
      End_If
    Else insérer  $m_i(S_i, MsgDeliv_i[i], LastSenders_i)$  dans  $FILE_i$ 
    End_If
  End_If
End_Function.

Void délivrer( $m_i$ )
  délivrer  $m_i$  à tous les processus locaux
   $MsgDeliv_j[i] = MsgDeliv_j[i] + 1$ 
  If ( $S_i \in LastSenders_j$ )
     $Old_MsgDeliv_i[i] = MsgDeliv_i[i]$ 
  Else
     $LastSenders_j = LastSenders_j \cup (S_i, MsgDeliv_i[i])$ 
  End_If
  Foreach ( $m_k(S_k, MsgDeliv_k[k], LastSenders_k) \in FILE_i$ )
    If ( $LastSenders_k = \emptyset$ )
      If ( $S_k \neq S_i$ ) or ( $(S_k = S_i)$  and ( $MsgDeliv_k[k] = MsgDeliv_i[i]$ ))
        supprimer  $m_k$  de  $FILE_i$ 
        exécuter la fonction délivrer( $m_k$ )
      End_If
    Else
      retirer  $m_k$  de  $FILE_i$ 
      exécuter la fonction Causal_Delivery( $m_i$ )
    End_If
  End_Foreach
End_Function.

```

4.3 Représentation du protocole sous forme d'automate

Pour faciliter l'effort de vérification et pour spécifier rigoureusement le protocole FMP, chaque entité FMP est représentée sous la forme d'une machine à états finis. A tout moment du déroulement de la communication, un état est associé à chaque entité FMP active. Collectivement, les membres du groupe forment l'état global du protocole qui est la composition des états associés à chacune des entités FMP.

La machine à états est commandée par des événements. Ces événements correspondent à une invocation locale du service FMP, à l'arrivée de paquets et à l'expiration de délais de garde, et au concours de conditions établies que l'on explicitera dans les paragraphes suivants. Les différents événements et leur description sont montrés par la figure suivante. Pour un événement particulier, la machine à états peut comporter des transitions multiples. Le choix d'une transition est, alors, défini par des conditions établies que l'on explicitera dans les paragraphes suivants. Dans un but de simplification, les tables d'états que nous allons présenter ne reportent pas le cas de réception et de suppression de paquets doublon.

Evènement	Description
Data	Réception d'un paquet donnée
Answer	Réception d'un paquet réponse
NACK	Réception d'un paquet requête de retransmission
State	Réception d'un paquet état
ACK	Réception d'un paquet d'acquiescement, imposant un ordre global de livraison
Req_Vue	Réception d'un paquet de demande d'insertion (ou de retrait) dans le (du) groupe
Change_Vue	Réception d'un paquet transportant la nouvelle vue du groupe
GId_Req	Réception d'une requête d'attribution de GId
Token	Réception d'un paquet d'attribution du rôle de séquenceur
Req_Timer	Expiration du délai de garde contrôlant l'émission d'une requête
Resp_Timer	Expiration du délai de garde contrôlant la retransmission d'une donnée
State_Timer	Expiration du délai de garde contrôlant l'émission d'un paquet état
ACK_Timer	Expiration du délai de garde contrôlant l'émission d'un paquet ACK d'ordre global
OrderQueue	Saturation de la file <i>OrderQueue</i>

TAB. 4.2 - *Table des Etats et événements*

4.3.1 Fonctionnement normal du protocole

Une opération normale de FMP peut être décomposée suivant les 5 états suivants. Chaque site du groupe se trouve dans un de ces états durant le déroulement des opérations normales. Notons que l'émission de paquets données est entièrement asynchrone, elle ne

dépend pas de l'état courant du site. Par ailleurs, pour simplifier la représentation des tables d'états, nous supposons que tout paquet doublon est détecté et supprimé par les sites récepteurs.

4.3.1.1 Site non séquenceur

Il s'agit de l'état où un site réceptionne les paquets diffusés. Le comportement du site et sa transition vers un nouvel état dépendent alors du type de paquet réceptionné :

- paquet donnée : la remise du paquet dépend de la qualité de service qui l'accompagne :

Si la QoS relative au paquet reçu est non fiable ou fiable, le paquet est directement remis à l'application. Si la QoS est atomique, le paquet est placé dans la file Order-Queue. Si la QoS est causale, le paquet est délivré ou placé dans la file relative au premier paquet de QoS causale dont il dépend. Si la QoS est fiabilité majoritaire, le site applique le mécanisme de livraison décrit dans 3.5.5.0.0.4. Une fois le paquet livré, le site reste dans son état d'origine.

Si une perte est détectée, le site initie une demande de retransmission et passe de l'état courant à l'état *En instance de requête*. A ce stade, le site maintient une liste de paquet requis identifiée par le site source du paquet donnée reçu. Si la requête est émise, le site passe du nouvel état à celui de *Attente de paquets*.

- paquet d'attribution du rôle de séquenceur : le site réalise les traitements établis par le protocole d'élection qui sont préalables à sa nouvelle fonction (comme la réception de tous les paquets en attente). Le site attend ensuite la réception de variables de contrôles qui lui permettent d'assurer le rôle de site séquenceur. Il s'agit de variables comme le *GId* actuel et la clé sur les verrous d'E.M transportés par les paquets de QoS atomique. C'est alors qu'il passe dans son nouvel état de *site séquenceur*.

- paquet requête : Si le paquet requis est disponible, le site passe à l'état *En instance de retransmission* et procède à la retransmission du paquet requis selon le mécanisme de temporisation décrit dans le service RDS. Le paquet réponse construit dépend du champ *RequestTyp* de la requête (voir 3.7.3.1).

Si, par contre, le paquet requis n'a pas été reçu, le site procède à une demande de retransmission et se place à l'état *En instance de requête*.

- paquet réponse : si la donnée retransmise provoque la détection d'une nouvelle perte, le site procède au traitement, décrit plus haut, aboutissant à une transition vers l'état *En instance de requête*.

De plus, si la QoS du paquet réponse est fiable, l'estampille du site retransmetteur est consultée par le récepteur pour détecter une éventuelle perte. Si tel est le cas, le site passe à l'état *En instance de requête*.

- paquet état : selon le mécanisme décrit dans 3.5.1.3.1, le site passe à l'état *En instance de retransmission* si une perte est détectée. Dans le cas contraire, le site reste dans le même état. De plus le site récepteur procède à la suppression des messages de sa file de réception selon le mécanisme décrit dans 3.5.1.3.2
- réception d'un paquet ACK : les messages acquittés reçus au préalable, sont livrés selon l'ordre indiqué par le paquet ACK. Si tous les identifiants contenus dans le paquet ACK ont été délivrés, le site se maintient dans le même état. Dans le cas contraire, il est placé dans la file *ACKQueue* et un processus de retransmission est initié, faisant passer le site à l'état *En instance de requête*.

Evènement	Condition	Action	Etat de transition
Data	QoS	paquet remis à l'application selon la QoS associée	Site non séquenceur
Data	QoS \neq non fiable Perte détectée	processus de demande de retransmission	En instance de requête
Token	-	acquisition des variables de contrôles	Site séquenceur
NACK	paquet requis disponible	processus de retransmission	En instance de retransmission
NACK	paquet requis indisponible	processus de demande de retransmission	En instance de requête
Answer	perte relative au site retransmetteur	processus de demande de retransmission	En instance de requête
Answer	site retransmetteur \neq site source & perte relative au site source	processus de demande de retransmission	En instance de requête
State	-	calcul du RTT	Site non séquenceur
State	perte détectée	processus de retransmission	En instance de retransmission
ACK	-	livraison ordonnée des messages acquittés	Site non séquenceur
ACK	paquet acquitté non reçu	processus de demande de retransmission	En instance de requête

TAB. 4.3 - Table d'états d'un site "non séquenceur"

4.3.1.2 Site séquenceur

L'état du site inclut celui d'un site normal du groupe avec un comportement spécifique relatif à l'occurrence des événements suivants :

- saturation de la file *OrderQueue*: le paquet ACK représentant la file *OrderQueue* est construit puis émis vers les sites du groupe qui vont délivrer les paquets de QoS atomique selon l'ordre communiqué par le paquet ACK.
- expiration de l'horloge ACK : le même traitement décrit ci-dessus est réalisé.
- réception d'un paquet requête de changement de vue de groupe : le séquenceur calcule la nouvelle vue de groupe, qu'il diffuse vers le groupe avec la QoS atomique. Le paquet ACK diffusé dans les conditions citées ci-dessus, indiquera l'ordre de prise en compte de la nouvelle liste.
- réception d'un paquet requête d'attribution d'un identificateur global *Gid* utilisé dans le cadre du service d'ordre causal. Il attribue, alors, un *Gid* unique qu'il retourne au site émetteur en utilisant le service de transport TCP. Le site jeton incrémente, par la suite, le numéro d'ordre *Gid* pour une prochaine attribution.

Evènement	Condition	Action	Etat de transition
ACK_Timer	file <i>OrderQueue</i> non vide	émission du paquet ACK	Site séquenceur
<i>OrderQueue</i>	-	ré-initialisation de ACKTimer émission du paquet ACK	Site séquenceur
Req_Vue	-	calcul et émission de la nouvelle vue de groupe	Site séquenceur
Gid_Req	-	émission d'une réponse	Site séquenceur

TAB. 4.4 - Table d'états du site "séquenceur"

4.3.1.3 En instance de requête

Les événements suivants sont attendus et traités par un site se trouvant dans cet état :

- paquet donnée: le paquet est délivré selon la QoS qu'il transporte. De plus, si ce paquet est inclus dans la liste des paquets requis, le site procède à la mise à jour de la liste, en supprimant l'identificateur du paquet reçu. Si, après mise à jour, la liste devient vide, le site interrompt le processus de requête et revient à l'état *Site non séquenceur* si aucune perte concernant le site source du paquet reçu n'est détectée.

Si, par contre, une perte est détectée, alors si la source du paquet reçu correspond à l'identificateur de la liste de requête localement maintenue, la liste des identifiants des nouveaux paquets perdus est insérée à la fin de l'ancienne file. Si le site source du paquet ne correspond à aucune liste de requête maintenue, une nouvelle liste identifiée par le site source est alors créée. Une fois la mise à jour effectuée, le site reste dans l'état *En instance de requête*.

- paquet réponse: le comportement relatif à la donnée retransmise est identique à celui correspondant à la réception d'un paquet donnée. En outre, si la QoS du paquet est fiable, alors une vérification de perte est effectuée sur l'estampille du site retransmetteur. De même, si une perte est détectée, le même traitement de mise à jour de la liste de requêtes est réalisé.
- paquet requête: la liste de requête est mise à jour à chaque fois qu'un identifiant de la liste de paquets requis reçue correspond à un identifiant de la liste de requête maintenue par le site. Effectivement, tout paquet requis ne doit plus l'être par les sites récepteurs de la requête. Si, après mises à jour, cette dernière liste est vide, le site interrompt le processus de requête et passe à l'état Site non séquenceur.
- paquet ACK: le procédé de livraison des messages acquittés par le paquet ACK est identique à celui décrit dans l'état *Site non séquenceur*. Si le paquet ACK est placé dans la file *ACKQueue*, avant d'initier une demande de retransmission, le site vérifie pour chaque identifiant contenu dans le paquet ACK inséré, si celui-ci correspond à l'identifiant du site source de la liste de requête maintenue. Si tel est le cas, soit aucun processus de requête n'est entrepris si l'identifiant est contenu dans la liste de requête, soit l'identifiant est inséré en fin de liste. Le site reste alors dans le même état. Le cas contraire n'est traité que si le site est dans l'état Site non séquenceur.

Evènement	Condition	Action	Etat de transition	
Data	QoS	paquet remis à l'application selon sa QoS	En instance de requête	
Data	non détection de perte & donnée reçue incluse dans Req_list	mise à jour de la liste de requête	si Req_list vide En instance de requête	si Req_list non vide Site non séquenceur
Data	détection de perte & source du paquet = Identifieur de Req_list	ajout du paquet perdu en fin de Req_list	En instance de requête	
Answer	-	même traitement que pour l'évènement Data	En instance de requête	
NACK	paquets requis inclus dans Req_list	mise à jour de la liste de requête	si Req_list vide En instance de requête	si Req_list non vide Site non séquenceur
NACK	paquets requis non inclus dans Req_list	ajout des paquets perdus en fin de Req_list	En instance de requête	
ACK	paquet acquitté non reçu	paquet ACK placé dans file ACKQueue processus de requête	si source paquet = Identifieur Req_list (*)	sinon En instance de requête
ACK	(*)			
	paquet inclus dans Req_list	pas de nouveau processus de requête	En instance de requête	
	paquet non inclus	paquet inséré en fin de Req_list	En instance de requête	
Req_Timer	Req_list non vide	émission d'une demande de retransmission	En attente de paquets	

TAB. 4.5 - Table d'états du site "En instance de requête"

4.3.1.4 Attente de paquets

Un site se maintient dans cet état tant qu'il n'a pas reçu les paquets manquants. A la réception d'un paquet donnée ou d'un paquet réponse requis, le site supprime l'identifiant correspondant dans la liste de requête maintenue. Si après mise à jour, la liste est vide, le site passe à l'état *Site non séquenceur*. La remise du paquet reçu dépend du service de livraison sollicité.

Evènement	Condition	Action	Etat de transition	
Data, Answer	paquet reçus inclus dans Req_list	mise à jour de Req_list	si Req_list vide En attente de paquet	si Req_list non vide Site non séquenceur

TAB. 4.6 - Table d'états du site "En attente de paquets"

4.3.1.5 En instance de retransmission

Les événements traités par un site se trouvant dans cet état, sont :

- paquet donnée : le paquet est délivré selon le service de livraison sollicité. Le site reste dans le même état.
- paquet réponse : si l'identifiant du paquet retransmis est contenu dans la liste de réponse, celui-ci en sera retiré. Si après mise à jour, la liste de réponse est vide, le processus de retransmission est annulé et le site passe à l'état *Site non séquenceur*.

Evènement	Condition	Action	Etat de transition	
Answer	paquet reçu inclus dans Resp_list	mise à jour de Resp_list	si Resp_list vide	si Resp_list non vide
			En instance de retransmission	Site non séquenceur
Resp_Timer	Resp_list non vide	émission d'un paquet réponse	Site non séquenceur	

TAB. 4.7 - Table d'états du site "En instance de retransmission"

4.3.2 Fonctionnement lors de changements dans le groupe

En plus des trois états présentés ci-dessus, trois autres états ont été introduits pour les besoins du mécanisme de gestion des membres d'un groupe. Ces états sont :

- *Non dans le groupe*
- *En instance de joindre le groupe*
- *Quittant le groupe.*

De nouveaux événement ont, également, été introduits. Il s'agit de :

- *Requête de changement de vue*
- *Nouvelle liste de membres*
- *Requête d'adhésion*
- *Horloge de départ*

L'état *Non dans le groupe* est l'état initial de chaque membre. L'état *En instance de joindre le groupe* est un état de transition dans lequel se trouve un site, en attendant que le site séquenceur établisse la nouvelle liste des membres qui inclut ce site.

Comme nous l'avons mentionné dans le chapitre précédent, l'implémentation actuelle de FMP ne prévoit pas la gestion de retrait d'entités FMP. Néanmoins, nous présentons ci-dessous la table d'états correspondant à une version de FMP intégrant, pour les applications l'exigeant, la QoS "gestion d'adhésion/retrait" : l'état *Quittant le groupe* correspond à un état d'attente d'un site pendant que le groupe enregistre le départ de ce site et qu'une nouvelle liste est établie par le gestionnaire.

L'événement "Requête de changement de vue" agit d'une façon analogue à l'arrivée d'un paquet donnée. Il est émis de façon asynchrone. Relativement à l'événement précédent, l'événement "Nouvelle liste de membres" agit comme la réception d'un paquet ACK.

Quand un site FMP fait une demande d'adhésion au groupe¹, l'état du site transite vers l'état *En instance de joindre le groupe* et diffuse un paquet requête de changement de vue qui fait office de demande d'adhésion. Si après un certain nombre de retransmission, aucune réponse ne parvient au site, il se considère comme le premier site du groupe et transite directement vers l'état *site séquenceur*. Cette stratégie permet de déterminer de façon simple le site séquenceur. Cependant, si le site séquenceur existe déjà, le site reçoit, en retour à sa demande, une nouvelle liste qui constitue sa vue de groupe.

Quand un site veut se retirer du groupe, il diffuse une "Requête de changement de vue". Avant la réception d'une nouvelle liste qui confirme sa suppression du groupe, le site continue de fonctionner de façon normale. Aussi, si le site en question est le site séquenceur, il continue de fonctionner jusqu'à ce qu'il ait passé le rôle de site séquenceur à un autre site. Si les deux conditions précédentes sont réunies, le site transite vers l'état *Quittant le groupe*. Le site peut rester dans cet état jusqu'à ce qu'il s'assure qu'aucun site ne demande ou ne pourrait demander un paquet qu'il détient. Le site positionne, par la suite, une horloge de départ dont l'expiration fait transiter le site de l'état courant vers l'état *Non dans le groupe*. Ceci pour éviter qu'une défaillance du réseau de communication ne bloque le site dans l'état *Quittant le groupe*.

4.4 Regard sur le langage Estelle

Estelle est un langage de description formelle basé sur un modèle de transition d'états étendu, *i.e.*, un protocole spécifié en Estelle peut être vu comme une machine à états

1. Le site postulant connaît au préalable l'adresse multicast du groupe invoqué.

finis, dont le comportement interne est décrit par des fonctions écrites en langage Pascal. Cette définition d'Estelle lui acquiert la particularité, parmi les outils FDT existants, d'être orienté implémentation. En effet, il permet de modéliser les services d'un protocole sous forme de composantes modulaires communicantes, chaque module agissant comme un automate. Ces modules peuvent s'exécuter parallèlement et peuvent communiquer par l'échange de message ou par le partage, d'une façon restreinte, de variables. La définition d'un module peut inclure de nouvelles définitions de modules "descendants". Appliquée de manière récurrente, cela conduit à une structure arborescente de modules formant la spécification Estelle.

Le comportement d'un module est spécifié par un ensemble de transitions que le module peut exécuter, mais aussi par la définition de sous-modules (modules fils) en même temps que leurs interconnexions.

Un module peut être muni de *points d'interactions* par lesquels il peut communiquer avec les autres modules via des canaux de communication possédant les propriétés de fiabilité et d'ordre séquentiel (FIFO). Un canal de communication relie deux points d'interaction de modules de même niveau hiérarchiques ou liés par la relation père-fils. Un canal de communication est typé, ce qui permet de spécifier les messages qui peuvent y être échangés.

4.4.1 Spécification Estelle

Une spécification Estelle fournit une modularité qui définit une encapsulation de données et de comportements (ou méthodes) permettant la conception de protocoles par la combinaison dynamique d'associations entre modules et comportements. Sémantiquement, un module Estelle correspond à une classe dans la terminologie objet. Une association "module-comportement" définit une instance de classe, appelée objet ou *tâche*². Nous résumons les entités Estelle composantes d'une spécification :

- *les tâches* : composées par l'association d'un comportement (*BODY* et d'un module (*MODULE*). Un module est créé par son père. Lorsqu'un père initialise (instruction *INIT*) son fils, il lui associe un comportement,
- *les points d'interactions* ou ports (*ip*) : constituent les interfaces d'un module avec son environnement externe,
- *les messages* : interactions échangées par les tâches sur les canaux reliant leurs ports,

2. Dans la suite du chapitre, nous désignerons indifféremment un objet par les termes de tâche ou module

- *transitions* : actions atomiques effectuées par les tâches. L'ensemble des transition d'une tâche constitue l'automate de contrôle qui lui est associé. Une transition est composée d'une partie des clauses suivantes Estelle suivantes :
 - clause *FROM* : vérifie si le module est dans l'état de contrôle voulu,
 - clause *TO* : donne l'état de contrôle résultant du tir de la transition,
 - clause *WHEN* : évaluée à vrai si et seulement si le premier message de la file associée au port désigné dans la clause correspond à l'identificateur de message spécifié,
 - clause *PROVIDED* : dépend des valeurs des variables internes au module, ou des paramètres passés dans le message de clause *WHEN*,
 - clause *DELAY* : spécifie un intervalle de temporisation préalable au tir de la transition. Cette clause ne peut être utilisée que dans les transitions *spontanées*, *i.e.*, ne possédant pas de clause *WHEN*,
 - clause *PRIORITY* : permet d'associer une priorité sur le tir de transitions concurrentes.

L'exécution d'une transition constitue une étape de l'exécution du système. La sélection d'une transition parmi l'ensemble des transition franchissables du système est gérée par les deux règles suivantes :

- le principe de la priorité ancêtre/descendant : le fait qu'une transition d'un module est franchissable inhibe la sélection des transitions de tous ses descendants,
- si un module n'a pas de transitions franchissables, la sélection des transitions dépend de l'attribut parallélisme qui lui est assigné.

4.4.1.0.1 Parallélisme dans la spécification :

Un module assigné *SYSTEMACTIVITY* définit un comportement asynchrone, *i.e.*, l'ensemble des transitions franchissables est soit vide, soit consiste en une seule transition, parmi les transitions franchissables des module fils, sélectionnée d'une façon non déterministe. Par opposition, un module assigné *SYSTEPROCESS* définit un comportement synchrone, *i.e.*, l'ensemble des transitions franchissables est soit vide, soit il comporte au plus une transition par module fils. S'il y a plus de deux transitions dans cet ensemble, leur exécution est synchronisée.

Une présentation détaillée sur les mécanismes du langage de spécification Estelle peut être consultée dans [ISO89].

4.5 Spécification formelle de FMP

L'objectif de cette section est de spécifier le comportement de FMP et les services qu'il offre, en utilisant le langage Estelle qui fournit une modularité et une encapsulation des données et des comportements, permettant, ainsi, la construction de FMP par la combinaison des descriptions d'objets réutilisables. La spécification FMP est composée d'une collection d'objets communicants. Chaque objet représente une instance de module défini dans Estelle. Différents modules réalisent les fonctions qui peuvent être exécutées en parallèle. La communication entre les modules permet à ces derniers d'échanger des données et de synchroniser, si nécessaire, leur comportement.

La spécification est accomplie en deux étapes : dans la première étape, nous déterminons tous les évènements auxquels une entité FMP réagit, ainsi que le comportement correspondant. La seconde étape consiste en une translation des fonctionnalités définies par le comportement FMP vers l'ensemble des transitions Estelle.

4.5.1 Architecture du système

L'architecture de la spécification, observée dans la figure 4.1, est composée de trois parties : l'entité application, l'entité FMP qui réalise les services du protocole, et l'entité module de communication qui représente la couche de transport OSI. Rappelons qu'une entité FMP représente un noeud du groupe supportant l'exécution d'un ou plusieurs processus de l'application. En conformité avec le système réel modélisé, le mode d'exécution entre entités FMP est asynchrone, alors que le comportement à l'intérieur d'une entité FMP est synchrone.

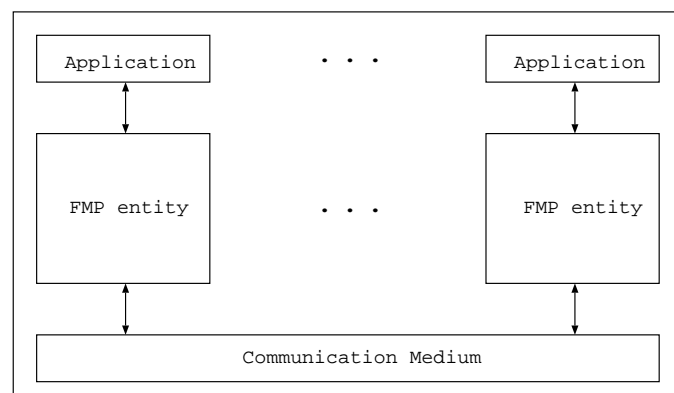


FIG. 4.1 - *Architecture de la spécification*

Une entité FMP est organisée autour d'une module racine et d'une collection de modules fils (figures 4.2 et 4.3). Une partie des modules fils (*i.e.*, *ThreadS*) a comme fonction de

diffuser les messages des processus applicatifs lorsque ceux-ci invoquent le service FMP local. L'autre partie des modules fils (*i.e.*, *ThreadR*) est en charge de réceptionner les messages transmis par l'entité média de diffusion. ThreadR et ThreadS sont créés et détruits de manière dynamique. Comme Estelle ne permet pas qu'un module s'arrête de lui même, tout module ayant accompli la tâche pour laquelle il a été créé, envoie un message à son père qui réalise l'opération de destruction (*i.e.*, *release*).

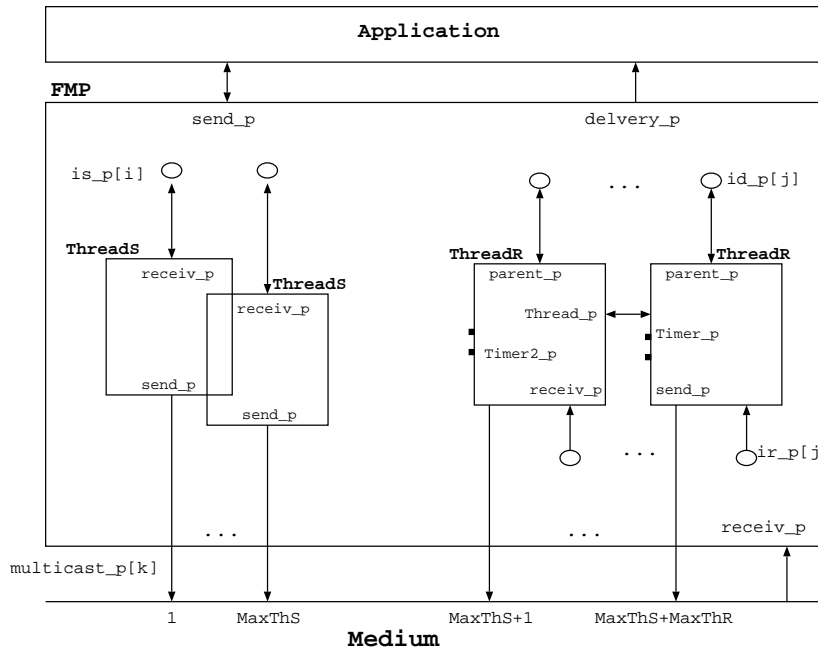


FIG. 4.2 - Architecture du module FMP

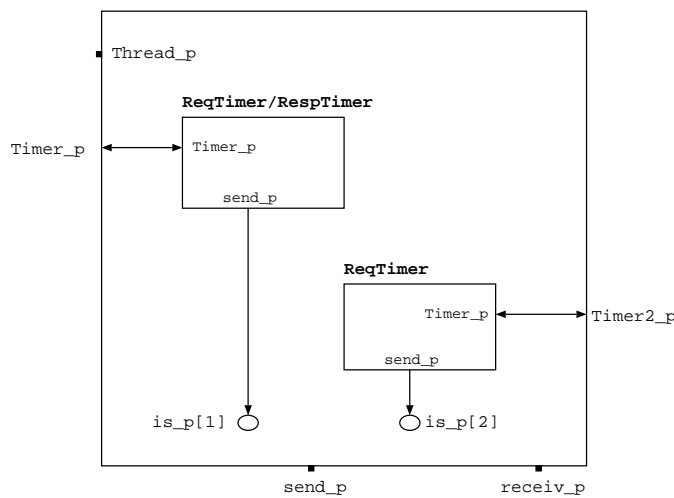


FIG. 4.3 - Architecture du module ThreadR

4.5.2 Comportements des modules de la spécification

Dans ce paragraphe, nous allons décrire le comportement de chacun des modules composants de l'architecture décrite ci-dessus. Nous nous limiterons au comportement relatif au service de livraison fiable (RDS), le comportement relatif aux services TODS et CDS a été longuement décrit dans les sections précédentes (3.6, 3.7.2, 4.2.2 et 4.2.3).

4.5.2.1 Module FMP

Le rôle du module racine (module FMP) consiste à :

- créer un module ThreadS qui traite les demandes de diffusion émanant du module application et communiquées via le port externe send_p. Toute donnée utilisateur est transmise par le module FMP au module fils ThreadS par le port interne is_p[2].
- créer un module ThreadR lorsqu'il reçoit, via le port externe receiv_p un paquet diffusé par un autre module FMP. Le paquet est transmis à ThreadR via l'un des ports internes ir_p[i]. Afin de ne pas surcharger le processeur local, le nombre de ThreadR pouvant s'exécuter en parallèle a été fixé (arbitrairement) à 5. Lorsque ce nombre maximum est atteint, tout message arrivant sur le port receiv_p est placé en file d'attente jusqu'à la libération d'un port ir_p[i] correspondant à un nouveau ThreadR créé.
- délivrer à l'application toute donnée remise par un module ThreadR, à travers un des ports internes id_p[i],
- détruire le module fils ThreadS ou ThreadR ayant transmis, via is_p[2] pour ThreadS et id_p[i] pour ThreadR, une message de fin de tâche,
- créer un module fils ThreadS chargé exclusivement de l'envoi périodique de messages état qui lui sont transmis par le port interne is_p[1].

4.5.2.2 Module ThreadS

Dans la perspective de diffusion de la donnée utilisateur transmise, le module ThreadS construit un message protocole, place une copie du message dans la file de réception *DataQueue* (pour une possible retransmission) puis le transmet au média de diffusion. Par ailleurs, le module ThreadS chargé de construire le paquet état place dans la partie donnée du message, un tableau représentant l'état de réception du site émetteur. Chaque

élément du tableau correspond au dernier numéro de séquence reçu du module FMP dont l'identité correspond de manière unique à la position de l'élément dans le tableau.

4.5.2.3 Module ThreadR

Le rôle d'un module ThreadR est de traiter tout message protocole transmis par le module racine. Le comportement de ThreadR dépend du type du message transmis et de la qualité de service l'accompagnant.

S'il s'agit d'un paquet donnée, un des traitements effectués par ThreadR est la détection de pertes. Dans l'affirmative, ThreadR construit une liste d'identificateurs correspondant à la séquence croissante de paquets perdus appelée *Req_list*. ThreadR crée, par la suite, un module fils, nommé ReqTimer à qui il transmet la liste *Req_list* via le port *timer_p*. le module ReqTimer active une horloge (*Req_timer*) initialisée à la distance séparant les entités FMP émettrice et réceptrice. A l'expiration de l'horloge, ReqTimer transmet au ThreadR père, un message requête contenant une *Req_list* éventuellement mise à jour durant la période précédant l'expiration de *Req_timer*. Ainsi, *Req_list* est composée uniquement des identificateurs des messages non reçus à l'expiration de *Req_timer*. La figure 4.4 décrit le comportement de ThreadR à la réception d'un paquet donnée.

La réception d'un paquet réponse, peut générer deux modules ReqTimer: le premier correspond à une perte dont l'origine est le module FMP émetteur du paquet réponse. Le second ReqTimer correspond à une perte dont l'origine est le site source de la donnée retransmise lorsque cette dernière n'est pas retransmise par son site originaire. Le second module est lié à ThreadR via le port *timer2_p*. Cette double création de module ReqTimer est une conséquence de l'application de la QoS fiable, aussi bien sur les paquets donnée que sur les paquets réponse (3.5.4).

Un traitement commun aux paquets donnée et réponse consiste à ce que ThreadR détermine s'il existe un module frère ThreadR' qui contrôle un module ReqTimer maintenant une liste *Req_list* dans laquelle se trouve l'identificateur du message reçu. Dans l'affirmative, ThreadR transmet à ThreadR' un message de mise à jour de *Req_list* à travers le port *Thread_p*. Ce dernier message est transmis au module ReqTimer cible via le port *timer_p* le reliant à ThreadR'. Dans le cas, d'un paquet réponse, ThreadR procédera à une double vérification (portant sur le site source et le site retransmetteur), si les sources du paquet réponse et de la donnée retransmise sont différentes. La figure 4.5 décrit le traitement relatif à la réception d'un paquet réponse.

A la réception d'un paquet requête, ThreadR commence par vérifier si les paquets identifiés par la liste *Req_list* transportée sont disponibles dans la file *DataQueue*. Dans l'affirma-

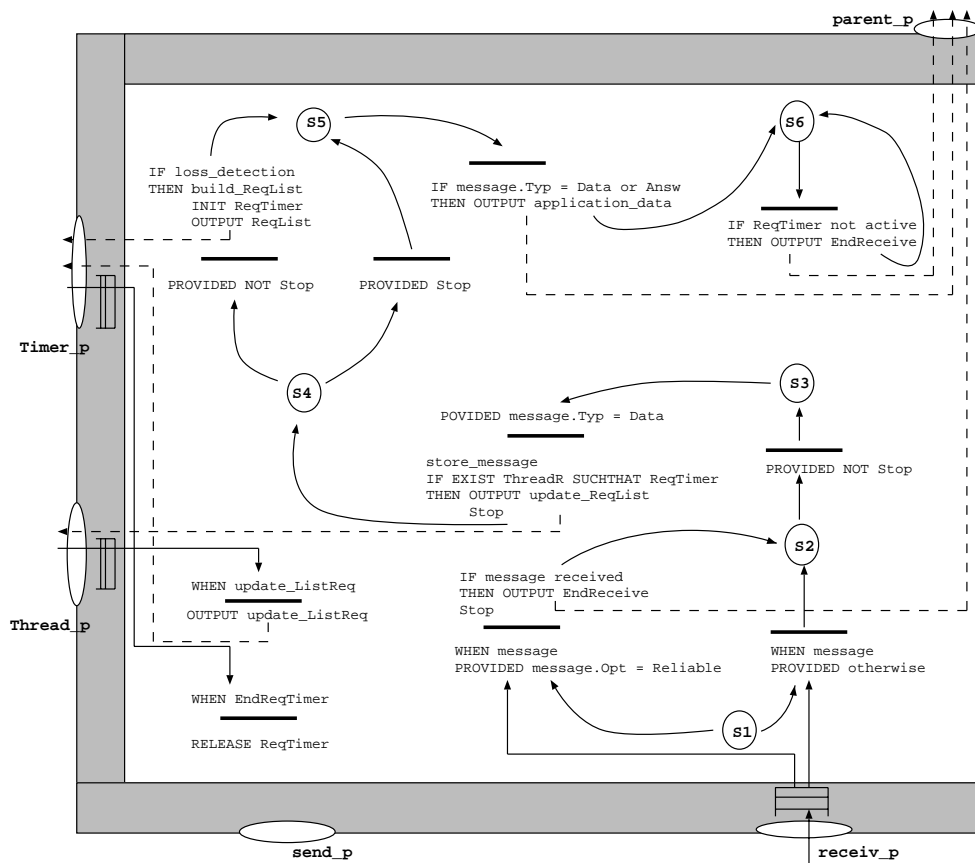


FIG. 4.4 - Comportement de ThreadR à la réception d'une donnée

tive, il vérifie s'il n'existe pas de modules ThreadR' en instance de retransmettre tous les paquets requis. Si c'est le cas, il ignore le paquet reçu. Dans le cas contraire, il crée un module fils, nommé RespTimer, à qui il transmet la liste Req_list reçue diminuée d'une part, des identifiants des messages requis mais indisponibles dans DataQueue, et d'autre part, des identifiants des messages en instance de retransmissions par des modules ThreadR'. La liste résultant de la mise à jour est désignée par le terme Resp_list. Le module Resptimer active une horloge (Resp_timer) initialisée à la distance entre les entités FMP émettrice et réceptrice. A l'expiration de l'horloge, RespTimer diffuse les paquets correspondant aux identifiants de Resp_list éventuellement mise à jour. En effet, si, avant l'expiration de Resp_timer, un module ThreadR" reçoit un paquet réponse contenant une donnée dont l'identificateur est inclus dans Resp_list, il envoie à ThreadR un message de mise à jour de Resp_list. Ce message est transmis par ThreadR, via le port timer_p, à son sous-module RespTimer qui supprime de Resp_list l'identificateur communiqué par ThreadR". Cette mise à jour est, ainsi, une conséquence directe d'un traitement supplémentaire effectué par un ThreadR à la réception d'un paquet réponse. Il vérifie si la donnée retransmise a déjà été reçue ou non. Dans l'affirmative, il vérifie si aucun module frère n'est en instance de retransmettre la donnée contenue dans le paquet réponse reçu. Si un tel module existe,

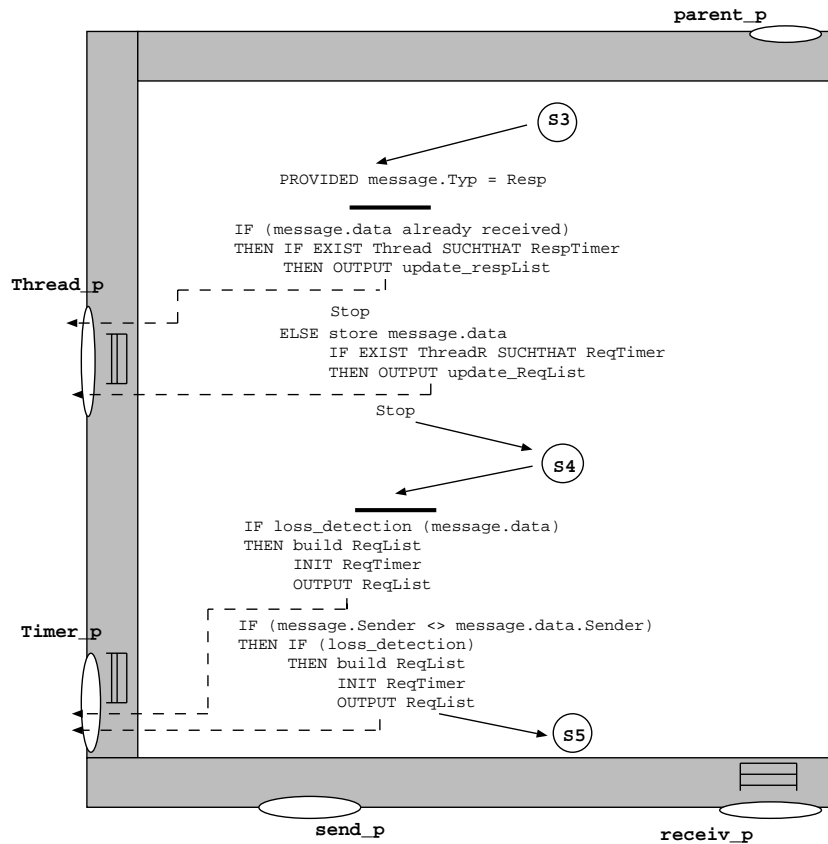


FIG. 4.5 - Comportement de ThreadR à la réception d'une réponse

il lui communique un message de mise à jour de la liste `Resp_list`. Dans le cas négatif, il insère simplement le paquet reçu dans `DataQueue`.

Un dernier traitement est effectué par ThreadR à la réception d'une requête. Il correspond au cas où une partie des messages identifiés par la liste `Req_list` transportée n'a pas été reçue. Dans ce cas, ThreadR vérifie s'il existe un module ThreadR' qui contrôle un sous-module ReqTimer dont la liste `Req_list'` contient au moins un identificateur des paquets non reçus. Dans ce cas, il transmet à ThreadR' un message de mise à jour de la liste `Req_list'`. Une fois ce message reçu, ThreadR' le transmet au module fils ReqTimer' qui masque le ou les identificateurs transmis par ThreadR, de la liste `Req_list`. La mise à jour de `Req_list'` doit intervenir avant l'expiration de l'horloge Req_timer associée à la liste. L'objectif de ce mécanisme est d'éviter la diffusion de requêtes doublons. Dans le cas contraire, ThreadR crée un module RespTimer comme cela a été décrit plus haut. La figure 4.6 décrit le comportement de ThreadR lors de la réception d'un paquet requête.

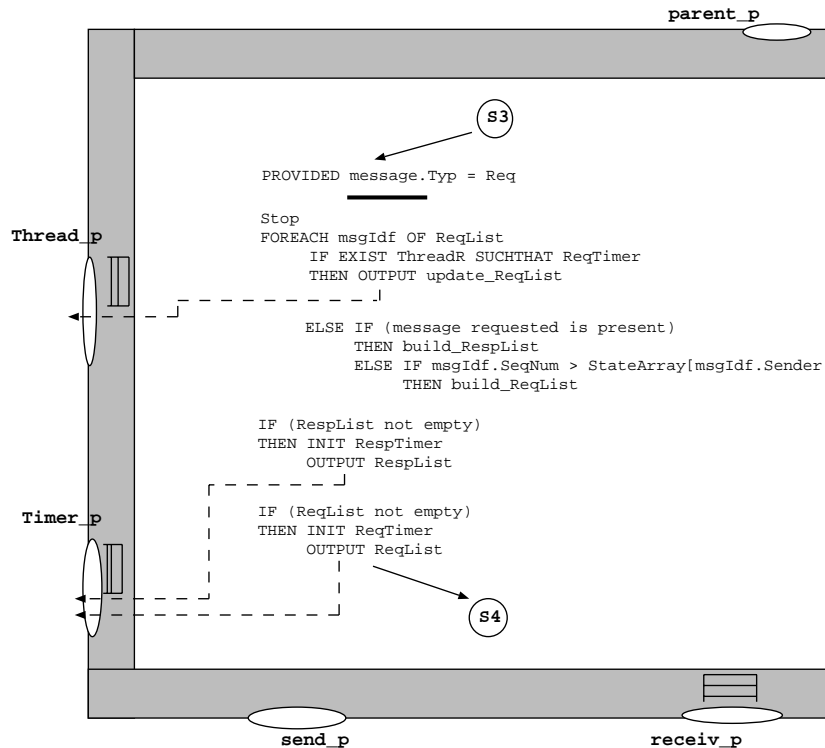


FIG. 4.6 - Comportement de ThreadR à la réception d'une requête

4.5.2.4 Modules ReqTimer et RespTimer

Le dernier niveau de modules de la spécification est représenté par les modules ReqTimer et RespTimer. Leurs comportements respectifs ont été décrits ci-dessus. Précisons que le module ReqTimer transmet le paquet requête construit à travers un des deux ports internes à ThreadR : `is_p[i]`. Cela est dû au fait qu'un module ThreadR peut contrôler deux modules ReqTimer. ThreadR diffuse par la suite la requête par son port externe `send_p`.

Une autre précision concerne le comportement de ReqTimer, une fois qu'il a transmis le paquet requête au module ThreadR. Il ré-initialise son horloge Req_timer au triple de la valeur initiale en prévision de la réception des messages requis. A l'expiration de la nouvelle valeur de Req_timer, ReqTimer diffuse un nouveau message requête contenant une liste Req_list éventuellement mise à jour par la réception du module racine FMP d'une partie des messages requis. Un autre évènement peut, également, provoquer un comportement similaire de ReqTimer : si après mises à jour de Req_list, tous les éléments de la liste sont masqués, ReqTimer triple la valeur initiale de l'horloge Req_timer et diffère ainsi l'émission de son propre message de requête. Après un troisième report de diffusion, le module ReqTimer ôte le masque sur la liste Req_list et diffuse un message de requête. Ce mécanisme permet à une nouvelle entité FMP de prendre le relais dans la requête de

message perdu, dans le cas où un partitionnement empêche le premier émetteur de la requête de recouvrer les messages perdus. Deux conditions déterminent la terminaison de ReqTimer lors de l'expiration de la valeur de Req_timer courante : la suppression de tous les éléments de la liste Req_list initiale ou une troisième diffusion d'un message requête. la figure 4.7 décrit le comportement du module ReqTimer.

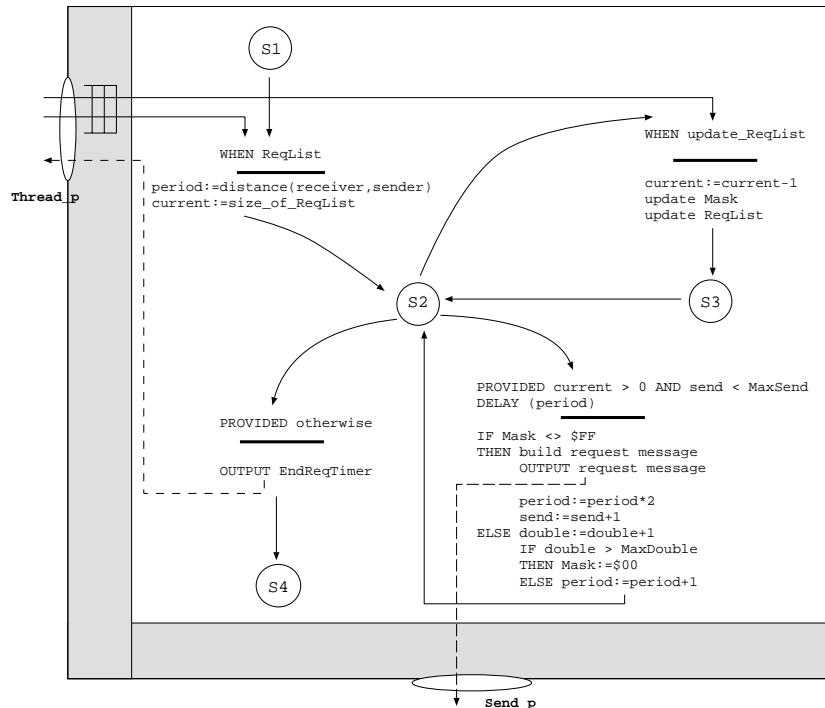


FIG. 4.7 - *Comportement de ReqTimer*

Notons qu'un module ThreadR ayant délivré le message transmis, ne transmet pas le message de fin de traitement tant qu'il contrôle un module ReqTimer ou RespTimer. Il a, en effet, la responsabilité de transmettre tout message de mise à jour de liste Req_list (ou Resp_list) au module ReqTimer (ou RespTimer) qu'il contrôle.

4.6 Questions d'implémentation

Un aspect positif du langage Estelle est qu'il inclut les mécanismes nécessaires à la réalisation du comportement spécifié du protocole FMP. De plus, il permet d'exécuter la spécification et de produire un prototype d'implémentation ou de simulation directement par l'utilisation de compilateurs et de simulateurs. Enfin, un des plus importants aspects d'Estelle est le parallélisme : il définit des modes d'exécution synchrones et asynchrones parmi les modules d'une spécification donnée.

4.6.0.4.1 Restrictions Estelle :

Cependant, les restrictions et contraintes imposées par Estelle introduisent une certaine lourdeur dans la spécification. En particulier, il n'est pas possible d'utiliser une variable globale commune à plusieurs modules. Cette restriction est due à l'inexistence de mécanisme qui gère les accès concurrents à cette variable. Cette contrainte peut, toutefois, être contournée de deux manières qui permettent de respecter l'intégrité de la variable globale : la première consiste à déclarer cette variable au niveau du module racine de la spécification. Les accès à cette variable par les modules descendants se font par l'envoi de message au module racine qui effectuera le traitement sur la variable et retournera le résultat. La seconde manière offerte par Estelle, consiste à utiliser une procédure externe (implémentées, par exemple, en C ou C++) dans laquelle la variable globale est déclarée. Cette procédure peut être invoquée par plusieurs modules, assimilés à des processus concurrents s'exécutant sur un, voir plusieurs processeurs. La procédure doit par conséquent implémenter un mécanisme d'exclusion mutuelle permettant le partage cohérent de la variable globale. Ainsi, l'utilisation de variables partagées implique deux mécanismes orthogonaux de communication entre les modules de la spécification : un mécanisme Estelle asynchrone basé sur les files associées aux ports et un mécanisme de synchronisation basé sur les primitives Unix/IPC.

Une autre contrainte vient du fait que si un module exporte une variable, celle-ci n'est accessible que par le module père. Un module de même niveau ne peut consulter cette variable. La raison est que si un module est assigné "PROCESS", ses fils coopèrent dans un environnement synchrone. Cela pose le problème d'exclusion mutuelle qu'Estelle ne gère pas. Cette restriction nous a conduit à ajouter différents messages et les traitements associés qui alourdissent la spécification. Par exemple, lorsqu'un ThreadR veut vérifier si un paquet donnée reçu est attendu par un module analogue ThreadR'. ThreadR envoie un message au module père FMP qui consulte une variable d'indication exportée par ThreadR' puis connecte les deux ports `thread_p` de ThreadR et ThreadR' pour que le premier communique au deuxième l'identificateur du message reçu.

Un aspect conceptuel important que nous avons eu à considérer dans la spécification de FMP, est la nécessité d'avoir des variables et des structures de données globales partagées par des modules de différents niveaux. Si les variables jouent un rôle important dans le comportement de FMP, alors leurs définitions et les traitements associés ne doivent pas être masqués dans des primitives externes, sinon l'utilisation d'un FDT ne serait pas justifiée. Notre politique a été donc d'utiliser les primitives externes uniquement dans les cas où les variables globales définies ainsi que les comportements associés ne jouent pas un rôle direct dans la spécification (exemple : la mise à jour de la table *StateTab*). Par contre, si le rôle d'une structure de donnée est important dans la spécification, elle est déclarée

au niveau du module FMP (c'est le cas, par exemple, des files de réception *DataQueue* et *OrderQueue*). Les accès concurrents des modules descendants à la structure se font par envois de messages au module FMP. Les accès sont sérialisés grâce aux listes FIFO associées aux ports du module FMP.

4.6.0.4.2 Outil de compilation :

La sémantique d'Estelle n'inclut pas la notion de processus ou de thread. L'isomorphisme "module Estelle-processus Unix" dépend entièrement de l'outil de compilation utilisé. Parmi les outils disponibles, nous nous sommes intéressés à deux générateurs de code pour deux raisons antagonistes liées à la notion de processus. D'un côté, l'outil Pet-Dingo [SS91] selon les principes suivants :

- lorsque deux modules sont dans le même processus Unix, ils sont sérialisés et il n'est pas nécessaire d'utiliser de mécanisme pour le contrôle d'accès concurrents,
- des modules qui sont dans des processus différents utilisent des échanges de messages pour se synchroniser et communiquer. La version de Pet-Dingo distribuée supporte deux modes de communication : Sun RPC (une ancienne version) et une librairie basée sur les sockets. Lorsque les processus sont sur la même machine, ils utilisent des sockets dans le "domaine" Unix (ce qui se traduit par des "pipes"). Lorsqu'ils sont sur des machines différentes, ils utilisent des sockets dans le domaine "Unix/Inet" (TCP/IP),
- la répartition des modules par processus et par machine se fait par l'insertion de commentaires spéciaux dans le code Estelle. Lorsqu'un module est créé (*INIT statement*), on peut spécifier s'il est sur un processus différent, et sur quelle machine il est. Un module de type `SYSTEMPROCESS` démarre nécessairement un nouveau processus.

La sémantique de la spécification FMP correspond à une hiérarchie de modules s'exécutant sur un même processeur : le module FMP constitue le niveau (1), les modules ThreadS et ThreadR le niveau (2) alors que les modules ReqTimer et RespTimer constituent le niveau (3). Puisque les modules (2) et (3), assignés `PROCESS` (au sens Estelle), ont plusieurs instances, celles-ci peuvent s'exécuter tels que des processus concurrents. Ce cas correspond à une des possibilités où Pet-Dingo génère du code distribué. Pour permettre le partage de variables entre les processus, nous avons implémenté des primitives externes utilisant soit les sémaphores (comme pour la table *StateTab*), soit les sockets du domaine Unix (comme pour les variables horloges). Pet-Dingo a cependant l'inconvénient d'avoir une interface de debugage pauvre.

D'un autre côté, EDT [Bud92] est un générateur plus performant que l'ensemble des outils existants. La version testée³ ne gérait pas les processus concurrents; toutes les instances de modules sont transposées en de simples structures de donnée et fonctions interprétées comme un processus unique. EDT dispose, cependant, d'une interface de debugage puissante. Pour cette raison nous avons également utilisé EDT pour les besoins de mise au point du protocole.

4.7 Conclusion

Nous avons décrit les structures de données essentielles ainsi que les algorithmes associés. L'ensemble constitue les mécanismes de base utilisés par les services de livraison décrits dans le chapitre précédent. Nous avons par la suite décrit le protocole sous forme de machine à états finis pour éliminer les risques d'ambiguïté et introduire la spécification formelle de FMP qui constitue la partie essentielle de notre travail. A travers cette démarche nous avons mis l'accent sur les solutions que nous avons apportées pour fournir un service de diffusion multipoint fiable qui d'une part renforce les techniques de fiabilité et de contrôle de flux réalisées par des protocoles tels que XTP [XTP92] et SRM [FJM95] et d'autre part, optimise les techniques de recouvrement de défaillances réalisées également par ces protocoles. Les solutions implémentant un service d'ordre total et causal basé sur le service de fiabilité ont également été décrites. En outre, nous avons montré comment nous avons adapté la spécification aux mécanismes assurant les différents services. De même, nous avons adapté l'outil de compilation à la spécification Estelle.

Par ailleurs, Estelle simplifie la conception et la spécification de protocoles de communication. Néanmoins, les contraintes propres au langage ainsi que les performances modestes des outils de compilation font qu'Estelle (et les autres FDTs) n'est pas réellement adapté pour une implémentation performante. Nous avons discuté les caractéristiques influentes des outils de compilation ainsi que certaines des restrictions du langage, notamment celles relatives à l'utilisation de variables partagées qui peut masquer une partie de la spécification.

3. une version récente d'EDT permet de générer des implémentations distribuées

Chapitre 5

Evaluation et perspectives

5.1 Evaluation

5.1.1 Protocoles existants

Différents objectifs et différentes définitions de fiabilité ont été décrits dans les diverses architectures multipoint réalisées. Notre étude a porté sur l'implication de ces différences dans les propriétés d'extensibilité, de robustesse, de gestion des groupes dynamiques et d'ordre.

RBP [CM84] constitue le premier travail sur les protocoles multipoint fiables. Il utilise un schéma centralisé organisé autour d'un anneau virtuel rassemblant l'ensemble des membres du groupe. Ce schéma repose aussi sur un privilège "tournant" attribué à un élément de l'anneau (le site jeton), lui permettant d'assurer une livraison fiable et totalement ordonnée. Ce schéma exige la reformation de l'anneau à chaque adjonction ou départ d'un membre du groupe. D'une part, cette gestion de l'anneau induit une charge supplémentaire influençant les performances du protocole, d'autre part, elle enlève au protocole la propriété d'extensibilité, notamment pour les très grands groupes. Par opposition, FMP utilise un schéma de fiabilité totalement réparti, augmentant ainsi, la robustesse et la disponibilité du service fiable. Il utilise l'efficacité du serveur d'ordre, tout en évitant la gestion d'un quelconque anneau. Pour ce faire, le service d'ordre repose sur le service fiable réparti. Enfin, la gestion du groupe multipoint¹ est déléguée à la couche réseau sous-jacente disposant du module Mulicast IP. Cela permet une extensibilité du protocole à un très grand nombre de sites.

RMP [MWK95] est une implémentation récente de RBP avec l'introduction de paramètres

1. Nous avons souligné au préalable que la notion de groupe fait référence aux entités FMP de niveau transport, et non pas au processus de niveau application

de QoS dans chaque transfert de donnée, ainsi qu'une gestion plus fine des changements de liste de groupe. Néanmoins, la gestion de l'anneau reste le principal inconvénient d'une telle approche.

Le protocole SRM [FJM95] a été conçu dans un esprit complètement opposé. Il repose sur le principe de ALF (Application Level Framing), selon lequel la meilleure façon de satisfaire les divers besoins d'une application est de laisser autant de fonctionnalités que possible à l'application. Pour cela, les algorithmes de SRM sont conçus pour satisfaire la définition minimale d'un multicast fiable. Malgré la généralité voulue, SRM reste particulièrement adapté aux applications multimédia interactives sensibles aux délais de transfert et impliquant un très grand nombre de participants. La réalisation de FMP est guidée par le même principe établi dans SRM, mais introduit les paramètres de QoS de niveau application, définissant la sémantique de fiabilité et d'ordre à appliquer sur le paquet à diffuser. Le schéma de fiabilité utilise les techniques de temporisation (slotting) et de l'échantillonnage (damping) pour assurer la fiabilité absolue dans les diffusions. Nous avons voulu améliorer divers aspects du schéma de fiabilité de SRM, soit dans le but de rendre le service fiable RDS plus robuste pour les applications qui le requièrent (numérotations des paquets réponse, schéma de gestion de groupe pour éviter le départ prématuré de sites, etc.), soit dans le but de réduire le flux de paquets de contrôle (regroupement des listes de paquets perdus ou à retransmettre par threads et mises à jour continues de ses listes). De façon optionnelle, FMP assure l'ordre dans la livraison grâce à deux modules d'ordre partiel et d'ordre total au dessus du service de remise fiable proposé. Nous résumons dans les paragraphes suivants les principales propriétés et fonctionnalités de FMP décrites dans les deux chapitres précédents.

Enfin, le système de communication Horus [RBM96] propose une architecture en couches qui permet à des groupes ayant des besoins différents de coexister dans le même système. Le modèle de communication repose sur une collection de "micro-protocoles" configurables qui confère au système la propriété de flexibilité. Ainsi, Horus conforte notre approche selon laquelle des mécanismes tels que l'ordre causal et l'ordre total peuvent, à la demande, être empilés au dessus d'un service fiable, lui-même dominant un service de livraison de type best-effort.

5.1.2 Propriétés intrinsèques de FMP

5.1.2.1 Caractéristiques générales

- Pour répondre aux besoins divers des applications de groupe, quatre services de diffusion sont fournis par le protocole FMP. Ces services vont de la diffusion non

fiable à une diffusion totalement ordonnée. Pour diffuser une donnée (SDU), un utilisateur invoque le service approprié qui assure la qualité de service voulue sur le paquet remis à l'entité FMP locale.

- Le service fiable peut avoir trois déclinaisons : la fiabilité absolue exigeant une remise fiable à tous les membres, selon ou non l'ordre d'émission, la fiabilité majoritaire (k -fiabilité, k représentant un nombre majoritaire de sites),
- L'ordre partiel et l'ordre total peuvent être accomplis de manière efficace et optimale en distribuant les procédés d'ordonnement à travers tous les membres du groupe.
- Une spécification formelle du protocole permet de lever les ambiguïtés et restrictions liées aux implémentations utilisant un langage naturel. De plus, le langage de spécification utilisé fournit les moyens d'une conception concurrente du protocole, permettant d'une part une meilleure disponibilité du protocole, et d'autre part, de réaliser efficacement le schéma de fiabilité conçu ; l'association d'un thread par processus de recouvrement de perte permet la mise à jour des listes de requête ou de réponse en instance d'émission, la gestion des horloges de contrôle et de gérer le flux en évitant les diffusions inutiles de requêtes et de données retransmises.
- Pour permettre une suppression cohérente de messages des buffers de réception, mais aussi un ordonnancement respectant l'ordre causal, le protocole construit et maintient une liste des membres du groupe dupliquée au niveau de chaque entité FMP. La cohérence de la liste est assurée grâce au mécanisme de synchronisme virtuel que fournit le service de diffusion atomique TODS.

5.1.2.2 Service RDS

- Paquets de recouvrement de QoS fiable : étant donné que n'importe quel membre du groupe peut retransmettre un paquet perdu, le schéma de fiabilité s'applique aussi bien aux paquets de type donnée qu'aux paquets de recouvrement (ou paquet réponse). Ainsi, un site qui reçoit un paquet réponse a le moyen de détecter une perte de messages qui sont originaires du site qui a retransmis, même s'il avait correctement reçu la donnée retransmise.
- Fiabilité majoritaire : utilisation d'horloges associées à chaque séquence de message reçue. A l'expiration de l'horloge si aucune requête n'est parvenue sur les messages associés, ces derniers sont délivrés.
- Grâce à la limitation du nombre de tentatives de recouvrement de paquets perdus

par un site, le service permet le recouvrement d'une donnée malgré le partitionnement du site original de la requête.

5.1.2.2.1 Contrôle de flux :

- Limitation des paquets de contrôle et de recouvrement en ajoutant aux techniques classiques de temporisation et d'amortissement des mécanismes basés sur les horloges locales et sur la technique des threads qui permettent d'atteindre les objectifs suivants : éviter les requêtes et réponses inutiles sur des données déjà requises ou retransmises par l'actualisation des listes de contrôle, éviter les requêtes et les retransmissions multiples par la technique d'inhibition, permettre le recouvrement d'une donnée malgré le partitionnement du site original de la requête,
- En associant un thread au traitement d'un paquet reçu, toute détection de perte est associée à un thread unique. De cette façon, la liste de paquets perdus (`Req_list`) construite par un thread ne correspond qu'à un site source unique. Par conséquent, les listes des paquets à retransmettre sont aussi associées au même site source. Ce procédé rend aisé les mises à jour des listes de paquets requis (`Req_list`) ou de paquets à retransmettre (`Resp_list`), évitant, ainsi, les requêtes et les retransmissions inutiles.
- Un aspect du contrôle de flux est également abordé en ne permettant la diffusion de paquets état que durant les périodes d'inactivité du groupe, afin de ne pas encombrer la bande passante par les messages état. Pour ce faire, dès que le nombre de threads `ThreadRs` est réduit à 0, l'horloge associée au paquet état est activée. Son expiration provoque l'émission du message état, si le nombre de `ThreadRs` est toujours nul,

5.1.2.3 Service TODS

5.1.2.3.1 Fiabilité : Le schéma de fiabilité du service TODS repose entièrement sur le service RDS. De ce fait, à moins d'une défaillance totale, tout membre du groupe délivre dans un même ordre global exactement le même nombre de messages que les autres membres de son groupe. Ainsi, le service TODS garantit la stabilité d'un message sans recourir à un schéma d'acquittement positif qui présente les risques d'implosion d'ACKs entraînant un congestionnement du réseau.

5.1.2.3.2 Contrôle de flux : En absence d'avalanches ou de défaillances du réseau de communication, le service RDS évite le phénomène de congestion des feedbacks grâce

à la technique du slotting/damping, mais aussi évite les paquets de requêtes doublons qui peuvent générer des retransmissions inutiles.

Un autre facteur de régulation de flux, est la taille des paquets d'acquittements de messages atomiques. Un paquet ACK ne contient qu'une liste de taille limitée d'estampilles relatives aux paquets à ordonner. La taille maximale de la liste est fixée par le site séquenceur à la création du groupe. Ces paquets ont été, soit reçus par les sites destinataires du paquet ACK, soit le seront par l'action du schéma de fiabilité de RDS.

5.1.3 Service CDS

5.1.3.0.3 Fiabilité : Le schéma de fiabilité du service CDS repose aussi sur le service RDS dont la caractéristique est d'assurer (dans un délai non borné, cependant) une livraison de messages unanime. La différence réside dans les paquets de recouvrement qui sont transmis sans aucune information additionnelle par rapport aux paquets donnée perdus. Ils ne sont, de ce fait, pas numérotés pour simplifier le procédé de livraison de paquets retransmis.

5.1.3.0.4 Contrôle de flux : Comme pour le service TODS, le flux de messages est réduit par l'utilisation des acquittements négatifs avec la politique de temporisation et d'amortissement. De même, la taille des paquets a été un critère prépondérant dans la conception du mécanisme de livraison causal. Pour un site donné, l'historique d'un message de QoS causale émis ne contient, pour tout site émetteur, l'estampille du dernier message délivré de ce site, depuis la dernier message émis. De cette façon un aspect du contrôle de débit est pris en compte.

5.1.4 Limitations de l'étude réalisée

5.1.4.1 Niveau conception

- utilisation répétées des horloges système. Cette utilisation de la ressource processeur peut induire, à terme, des temps de traitement lents pouvant avoir une conséquence sur le comportement global du système qui verrait son délai de latence augmenter.
- détermination des périodes d'inactivité dans le groupe pour éviter que les messages état encombrant la bande passante. Cet aspect important de la fiabilité mérite une étude plus conséquente car il permet de gérer le problème de congestionnement des récepteurs.

- mécanisme de gestion des défaillances : un tel mécanisme permet l'exclusion de sites ayant subi une défaillance totale, la reformation du groupe (ou de groupes fonctionnels) et la détermination de la fin de session lorsqu'un nombre majoritaire de membres est défaillant.

5.1.4.2 Niveau implémentation

- Etude de performances : bien qu'étant orienté implémentation, Estelle a été conçu que pour spécifier rigoureusement les fonctionnalités des protocoles. L'objectif de réaliser des protocoles de performances élevées ne peut être atteint qu'avec des langages de haut niveau. C'est la principale raison pour laquelle l'étude de performance n'a pas été réalisée sur le protocole spécifié.
- Vérification : A la différence de langages FDT tel que Lotos, très peu d'outils permettent de réaliser des preuves sur une spécification Estelle. L'outil (Vesar [ACD⁺93]) reste à l'état de prototype et comporte des restrictions, notamment le fait que la spécification ne doit pas créer les modules dynamiquement, ce qui ne nous a pas incités à procéder à la vérification des services spécifiés.

L'intérêt d'avoir utilisé Estelle est que c'est un langage qui permet de décrire, de manière modulaire, des systèmes distribués qui communiquent par envoi de messages (tel qu'en réalité). Il fournit, en effet, une modularité et une encapsulation des données et des comportements, permettant la construction de protocoles par la combinaison des descriptions d'objets hiérarchisés et réutilisables. La spécification est, ainsi, composée d'une collection d'objets simples communicants dont le comportement peut être décrit rigoureusement par des automates.

5.2 Perspectives

L'étude réalisée peut être étendue par les perspectives suivantes :

- Un niveau de flexibilité supplémentaire peut être atteint en introduisant un mécanisme de résilience total ou majoritaire modulable selon les besoins des applications. Le nombre minimal de sites corrects est fixé par le processus créateur de l'application de groupe lors du dialogue de connexion à l'entité FMP locale.
- Un protocole de gestion de groupe adapté aux spécificités du protocole décrit, notamment la propriété d'extensibilité qui ne doit pas être altérée par un mécanisme de gestion explicite contraignante.

- Contrôle de congestion : Le contrôle de congestion reste un problème majeur et un axe de recherche actif dans les protocoles multipoint. Très peu de protocoles implémentent un mécanisme de contrôle explicite. Les mécanismes de contrôle de congestion requis pour un protocole multipoint dépendent étroitement des services de gestion de ressources disponibles au niveau du service réseau. Nous donnons en annexes des éléments de réponse sur les techniques de contrôle de congestion qui font l'objet d'investigations dans le domaine du multipoint.

Bibliographie

- [AABK88] M. Ahamad, M. H. Ammar, J. M. Bernabeu, and M. Y. Khalidi. **Using Multicast Communication to Locate Resources in a LAN-based Distributed System.** *Proc. of the 13th Conference on Local Computer Networks*,, pages 193–202, October 1988.
- [AB85] M. Ahamad and A. J. Bernstein. **An Application of Name Based Addressing to Low Level Distributed Algorithms.** *IEEE Trans. On Software Engineering*, SE-11(1):59–67, January 1985.
- [AC93] G. Agha and C. J. Callsen. **ActorSpace: An Open Distributed Programming Paradigm.** *ACM SIGPLAN Notices*, 28(7):23–32, July 1993.
- [ACD⁺93] B. Algayres, V. Coelho, L. Doldi, H. Garavel, Y. Lejeune, and C. Rodriguez. **VE-SAR: a pragmatic approach to formal specification and verification.** *Computer Networks and ISDN Systems*, 25:779–790, 1993.
- [AFM92] S. Armstrong, A. Freier, and K. Marzullo. **Multicast transport protocol.** *Internet Request for Comments RFC*, (1301), February 1992.
- [AN95] N. Adly and M. Nagi. **Maintaining Causal Order in Large Scale Distributed Systems Using a Logical Hierarchy.** *Proc. Of IASTED Innsbruck, Austria*, February 21-23 1995.
- [Bir86] K. P. Birman. **ISIS: A System for Fault-Tolerant Distributed Computing.** Technical Report 86-744, Dept. Of Computer Science , Cornell University, Ithaca, New York, April 1986.
- [Bir94] K. P. Birman. **Response to Cheriton and Skeen’s Criticism of Causal and Totally Ordered Communication.** *Operating Systems review*, 28(1):11–21, January 1994.
- [BJ87a] K.P. Birman and T. A. Joseph. **Exploiting Virtual Synchrony in Distributed Systems.** *Proc. Of the 11th ACM Annual Symposium on Operating Systems Principles*, pages 123–138, 1987.
- [BJ87b] P. P. Birman and T. A. Joseph. **Reliable Communication in the Presence of Failures.** *ACM Trans. On Computer Systems*, 5(1):47–76, February 1987.

- [BOG⁺94] C. Bormann, J. Ott, H.-C. Gehrcke, T. Kersch, and N. Seifert. **MTP-2: Towards Achieving the S.E.R.O. Properties for Multicast Transport**. *International Conference on Computer Communication and Networks ICCCN*, 1994.
- [BSD88] O. Babaoglu, P. Stephenson, and R. Drummond. **Reliable Broadcast and Communication Models: Tradeoffs and Lower Bounds**. *Distributed Computing*, 2:177–189, 1988.
- [BSS91] K. P. Birman, A. Schiper, and P. Stephenson. **Lightweight causal and atomic group multicast**. *ACM Trans. On Computer Systems*, 9(3):272–314, August 1991.
- [Bud92] S. Budkowski. **Estelle development toolset (EDT)**. *Computer Networks and ISDN Systems*, 25:63–82, 1992.
- [CAS85] F. Cristian, H. Aghili, and R. Strong. **Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement**. *Proc. Of the 15th Symposium on Fault-tolerant Computing, Ann Arbor, Michigan, USA*, pages 200–206, June 19-21 1985.
- [Che88] D. R. Cheriton. **The V distributed System**. *Communications of the ACM*, 31(3):314–333, March 1988.
- [CM84] J. Chang and N.F. Maxemchuk. **Reliable Broadcast Protocols**. *ACM Trans. On Computer Systems*, 2(3):251–273, August 1984.
- [CM88] D. Cheriton and T. P. Mann. **Decentralizing a Global Naming Service for Improved Performance and Fault Tolerance**. *ACM Transactions on Computer Systems*, 7(2):147–273, May 1988.
- [Coo90] E. C. Cooper. **Programming Language Support for Multicast Communication in Distributed Systems**. *Conference on Distributed Computing Systems, Proc. Of the 10th Int ,Paris, France*, pages 450–457, May 28-June 1 1990.
- [Coo94] R. Cooper. **Experience with Causally and Totally Ordered Communication Support: A Cautionary Tale**. *Operating Systems Review*, 28(1):28–31, January 1994.
- [Cor81] Cornafon. *Systemes Informatiques Repartis, Concepts et Techniques*. Ed. Dunod, 1981.
- [CP88] J. Crowcroft and K. Paliwoda. **A Multicast transport Protocol**. *ACM Computer Communication Review SIGCOMM*, 18(4), August 1988.
- [Cri91] F. Cristian. **Understanding Fault-tolerant Distributed Systems**. *Communications of the ACM*, 34(2):56–78, February 1991.
- [CS93] D. R. Cheriton and D. Skeen. **Understanding the Limitations of Causally and Totally Ordered Communication**. *Operating Systems Review*, 27(5):44–57, December 1993.

- [CZ85] D. Cheriton and W. Zwaenpoel. **Distributed Process Groups in the V kernel**, *ACM Trans. On Computer Systems*, 3(2):77–107, May 1985.
- [Dan89] P. Danzig. **Finite Buffers and Fast Multicast**. *Proc. Of the ACM Conf. On Measurement and Modelling of Computer Systems, ACM SIGMETRICS*, 1989.
- [Dee89] S. Deering. **Host Extensions for IP Multicasting**. Technical Report 1112, RFC, August 1989.
- [Dio94] C. Diot. **Reliability in Multicast Services and Protocols**. *International Conference on Local and Metropolitan Communication Systems, Chapman Hall Editor, Kyoto*, December 1994.
- [DK93] W. Dabbous and B. Kiss. **A Reliable Multicast Protocol for a White Board Application**. Technical Report 2100, INRIA, Sophia-Antipolis, France, November 1993.
- [DTH92] S. Dupuy, W. Tawbi, and E. Horlait. **Protocols for High-Speed Multimedia Communications**. Technical Report No. 92/82, Laboratoire MASI, France, November 1992.
- [FJM95] S. Floyd, V. Jacobson, and S. Mccanne. **A Reliable Multicast Framework for Light-weight Session and Application Level Framing**. *ACM SIGCOMM 95*, pages 342–356, August 95.
- [FT92] G. Florin and C. Toinard. **A New Way to Design Causally and Totally Ordered Multicast Protocols**. *Operating Systems Review*, 26(4):77–83, October 1992.
- [Gro96] M. Grossglauser. **Optimal deterministic timeouts for reliable scalable multicast**. *Proc. of the IEEE Infocom, San Francisco, CA*, pages 1425–1432, March 1996.
- [GS91] H. Garcia and A. Spauster. **Ordered and Reliable Multicast Communication**. *ACM Trans. On Computer Systems*, 9(3):242–271, August 1991.
- [Hug91] L. Hugues. **Identifying Migrated Objects Using Multicast Addresses**. *Computer Communications*, 14(7):423–431, September 1991.
- [ISO89] **Estelle: A Formal Protocol Description Technique Based on an Extended State Transition Model**. Information Processing System - osi, 1989.
- [Jac88] V. Jacobson. **Congestion Avoidance and Control**. *ACM, SIGCOMM Proceedings*, pages 314–328, 1988.
- [KP91] P. Karn and C. Partridge. **Improving Round-trip Time Estimates in Reliable Transport Protocols**. *ACM Trans. On Comp. Syst.*, 9(4):364–373, November 1991.

- [Kra91] S. Krakowiak. *Construction des Systèmes d'Exploitation Répartis*, pages 4–15 – 4–34. INRIA, Collection Didactique, 1991.
- [KTH89] M.F. Kaashoek, A. S. Tanenbaum, and S. F. Hummel. **An Efficient Reliable Broadcast Protocol**. *ACM Operating System Review*, 23(4):5–19, October 1989.
- [KZ96] A. Koifman and S. Zabele. **Ramp: A reliable adaptive multicast protocol**. *Proc. of the IEEE Infocom, san francisco*, pages 1442–1451, March 1996.
- [Lam78] L. Lamport. **Time, Clocks and the Ordering of Events in a Distributed System**. *Communications of the ACM*, 21(7):558–565, July 1978.
- [LCN90] L. Liang, S. T. Chanson, and G. W. Neufeld. **Process Groups and Group Communications: Classifications and Requirements**. *IEEE Computer*, 23(2):56–66, February 1990.
- [LOT89] **LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observation Behaviour**. *ISO IS*, (8807), 1989.
- [LP96] J. C. Lin and S. Paul. **Rmtp: A reliable multicast transport protocol**. *Proceedings of the IEEE INFOCOM*, pages 1414–1424, March 1996.
- [MA91] P. Minet and E. Anceaume. **What atomic protocol?** *Proc. Of the ERCIM Workshops INESC, Lisbon, Portugal*, pages 55–59, November 14-15 1991.
- [Mau93] A. Maurand. **Conception Formelle, Implementation et Mesures de Protocoles de Communication Haute Vitesse**. Technical Report 93516, Rapport LAAS-CNRS France, December 1993.
- [Mil92] D.L. Mills. **Network Time Protocol (version3)**. *RFC*, (1305), March 1992.
- [MMA⁺96] L. E. Moser, P. M. Melliar, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. **Totem: A Fault-Tolerant Multicast Group Communication System**. *Communications of the ACM*, 39(4):54–63, April 1996.
- [Moc83] P. V. Mockapetris. **Analysis of Reliable Multicast Algorithms for Local Networks**. *In Proceeding of the Eighth Data Communications Symposium, USA, IEEE computer soc. Press.*, pages 150–157, October 1983.
- [MWK95] T. Montgomery, B. Whetten, and S. Kaplan. **A high performance totally ordered multicast protocol**. *Theory and Practice in Distributed Systems, Springer Verlag, LCNS 938*, pages 33–57, 1995.
- [Ngo91] L. H. Ngoh. **Multicast Support for Group Communications**. *Computer Networks and ISDN systems*, 22:165–178, 1991.
- [Pal88] K. Paliwoda. **Transactions Involving Multicast**. *Computer Communications*, 11(76):313–318, December 1988.

- [PBS89] L. Peterson, N. Bucholz, and R. Schlichting. **Preserving and Using Context Information in Interprocess Communication**. *ACM Transactions on Computer Systems*, 7(3):217–246, August 1989.
- [Raj92] B. Rajagopalan. **Reliability and Scaling Issues in Multicast Communication**. *ACM Computer Communication Review*, 22(4):188–198, August 1992.
- [Ram90] K. V. S. Ramarao. **Efficient Fault-tolerant Broadcasts**. *Journal of Systems Software*, 11:131–141, January 1990.
- [RB91] A. M. Ricciardi and K. P. Birman. **Using Process Groups to Implement Failure in Asynchronous Environments**. *Proc. Of the 14th Annual ACM Symposium on Principles of Distributed Computing*, pages 341–351, 1991.
- [RBM96] R. V. Renesse, P. K. Birman, and S. Maffei. **Horus: A Flexible Group Communication System**. *Communications of the ACM*, 39(4):76–83, April 1996.
- [Ren94] R. V. Renesse. **Why bother with CATOCS?** *Operating Systems Review*, 28(1):22–27, January 1994.
- [RST91] M. Raynal, A. Schiper, and S. Toueg. **The Causal Order Abstraction and a simple way to implement it**. *Inf. Proc. Letters*, 39:125–148, 1991.
- [SDL88] *CCITT Specification and Description Language SDL*. CCITT Blue Book, 1988.
- [SF92] H. Santoso and S. Fdida. **Transport Layer Multicast : XTP Bucket Error Control and its Enhancement**. Technical Report 92/62, Laboratoire Masi, France, October 1992.
- [Sia96] A. Siafa. **Protocoles de Groupe comme Support de Communication pour Applications Réparties**. *JDIR, ENT, Paris*, pages 1–6, September 1996.
- [Sia97] A. Siafa. **Using Estelle to implement a Flexible Multicast Protocol**. *IEEE ICCCN , Las Vegas, USA*, pages 170–174, 22-25 September 1997.
- [SS91] R. Sijelmassi and B. Strausser. **The Distributed Implementation Generator: an overview and user guide**. Technical Report NCSL/SNA-91/3 MD 20899, National Institute of Standards and Technology, Gaithersburg, January 1991.
- [TR85] A. Tanenbaum and R. V. Renesse. **Distributed operating systems**. *Computing surveys*, 17(4):419–470, December 1985.
- [WZZ91] X. Wang, H. Zhao, and J. Zhu. **GRPC: A Communication Cooperation Mechanism in Distributed Systems**. Technical Report 12 pages, Northeast University of Technology, Shenyang, China, 1991.
- [XTP92] *XTP Protocol Definition*. Protocol Engines Inc., 3.6 edition, Jan 1992.

- [YGS95] R. Yavatkar, J. Griffioen, and M. Sudan. **A reliable dissemination protocol for interactive collaborative applications.** *Proceedings of the ACM Multimedia*, 1995.