Conception et réalisation d'un protocole de diffusion fiable pour réseaux locaux

Daniel Veillard

THÈSE

présentée par

Daniel Veillard

pour obtenir le titre de

Docteur de l'Univ**e**rsité
Joseph Fourier − Grenoble 1

spécialité : INFORMATIQUE

Conception et réalisation d'un protocole de diffusion fiable pour réseaux locaux

Soutenue le 1er février 1996

Composition du jury :

Président	Roland Balter
Rapporteurs	Jean−Marie Rifflet
	Paulo Veríssimo
Directeur	Sacha Krakowiak
Examinateurs	Jacques Mossière
	Enrico Clementi

Aux Hélènes ...

Introduction

L'évolution des systèmes informatiques est guidée par les deux facteurs principaux que sont l'amélioration des performances liée à l'évolution technologique, et les besoins des utilisateurs qui recouvrent une population de plus en plus diversifiée avec la démocratisation de l'outil informatique. L'influence de ces deux phénomènes conjugués s'est ressentie au niveau des systèmes d'exploitation, qui ont évolué depuis la gestion par lots, les systèmes interactifs, jusqu'aux systèmes en temps partagé utilisés actuellement. Une fois acquis les principes théoriques de la structuration des entités d'exécution sous la forme de processus, l'évolution s'est naturellement porté vers une meilleure intégration de cette structure dans le système et l'amélioration de l'interface offerte aux utilisateur.

La mise en place des réseaux informatiques est un des facteurs technologiques qui ont fortement modifié la structure des systèmes informatiques durant les deux dernières décennies. Nous pouvons constater une évolution similaire de l'intégration de services qu'ils procurent, d'abord sous la forme d'échanges par lots tels qu'UUCP, puis l'introduction de services de communication interactifs dont FTP et Telnet, et enfin l'intégration au niveau du système et des langages de programmation des primitives de communication ; on citera par exemple les systèmes de fichiers distants NFS et les mécanismes d'appels de procédure à distance.

La mise en place de mécanismes permettant le travail coopératif est aussi une des évolutions récentes de l'outil informatique moderne, provoquée par les besoins des utilisateurs. La présence de réseaux locaux permettent la réalisation d'outils facilitant le travail concerté de plusieurs personnes à une même tâche, au sein d'entités telles qu'une entreprise ou un groupe de travail. A une échelle plus large, les "News" d'Internet permettent à des personnes réparties sur l'ensemble de la planète de s'échanger des informations, formant par là même l'exemple le plus achevé de programme informatique coopératif, par le nombre de personnes concernées et le volume des informations échangées. Ce n'est pas pour autant un outil évolué, car on peut constater qu'il n'est ni interactif ni fortement intégré au système informatique. Nous sommes donc amenés à nous interroger sur les mécanismes permettant d'intégrer la notion de travail coopératif parmi la panoplie des outils informatique, en particulier les bases théoriques permettant de modéliser les mécanismes nécessaire, les possibilités d'intégration de ces mécanismes dans les systèmes informatiques et les méthodes à adopter pour rendre leur utilisation aisée.

Cette thèse porte sur les protocoles de diffusion, qui sont un des mécanismes conçus pour le support du travail coopératif dans les systèmes répartis, systèmes constitués d'un ensemble de machines connectées à un même réseau de communication. Il est basé sur la notion théorique de groupe de processus communiquants, le groupe étant le formalisme retenu pour représenter un ensemble d'entités actives collaborant à une même tâche. Les

protocoles ont pour but de faire communiquer l'ensemble des participants en diffusant les messages qui y sont envoyé à tous les membres du groupe.

Dans le premier chapitre, nous présentons l'état actuel de la théorie sur les protocoles de diffusion et un ensemble d'exemples et de réalisations issues de la recherche. Puis les principes de fonctionnement, les architectures employées et les utilisations visées sont présentées.

Le deuxième chapitre expose les choix de conception retenus pour la réalisation d'un protocole de diffusion. Le principe des groupes opaques retenu est présenté ainsi que diverses optimisations s'appliquant au protocole initial. Une étude théorique présente les gains que l'on est en mesure d'espérer.

Les troisième chapitre présente la mise en œuvre, les choix d'implantation retenus, et détaille la réalisation des protocoles. Nous présentons les différentes plate−formes systèmes sur lesquelles le protocole a été porté et les spécificité de chaque environnement système.

Dans le quatrième chapitre, nous présentons les méthodes de développement utilisées. Nous présentons les principes et l'interface d'un émulateur de réseau local qui a été développé spécifiquement pour la mise au point de protocoles.

Enfin les résultats expérimentaux sont présentés dans le cinquième chapitre. Nous y fournissons une analyse détaillée des temps d'exécution dans le noyau du système et des échanges avec le protocole en mode utilisateur. L'impact de divers paramètres sur les performances est présenté. Enfin une validation pratique des optimisations proposées a été réalisée.

La conclusion permet de faire le point sur les apports et conclusions de cette thèse. Une analyse des extensions possibles du protocole est fournie, ainsi qu'une analyse à moyen terme de l'utilisation des protocoles de diffusion dans le cadre du travail coopératif.

Chapitre I

État de l'art

Ce chapitre fait le point sur les divers protocoles de diffusion existants. Les mécanismes mis en œuvre sont détaillés, puis nous étudions les diverses architectures des protocoles et leur implantation au sein des systèmes existants. Enfin les principaux types d'applications visées sont présentés.

I.1 Généralités sur les protocoles de diffusion

L'utilisation de protocoles est nécessaire lorsque l'on veut faire communiquer des entités actives et que les mécanismes servant à l'échange d'informations ne sont pas parfaits. Les algorithmes de diffusion sont utilisés lorsque les transferts impliquent plus de deux entités (par opposition aux algorithmes point−à−point). Nous nous limitons dans cette étude aux seuls protocoles utilisés pour des réseaux informatiques, mais on peut constater que des protocoles existent aussi par exemple au sein de multiprocesseurs, qu'il soient ou non pourvus d'une mémoire commune. On se heurte alors au problème d'absence d'état global qui est la difficulté majeure de toute l'algorithmique répartie. Divers types de pannes sont possibles, en fonction des caractéristiques du réseau, et du comportement des sites qui prennent part aux échanges.

Les propriétés sémantiques qui peuvent être offertes par un protocole de diffusion sont sensiblement plus complexes que pour une communication point−à−point. En effet, le protocole doit garantir, si possible, que tous les sites qui participent aux communications ont une vision cohérente de l'état d'avancement du protocole. Différents types de propriétés peuvent être offertes suivant les besoins des utilisateurs, et la composition des diverses caractéristiques permet d'obtenir plusieurs classes de protocoles répondant à des besoins spécifiques.

La multiplicité des propriétés, des classes de protocoles et les diverses réalisations possibles ont motivé plusieurs projets de recherche sur la mise en œuvre de protocoles, la décomposition de ceux−ci en diverses couches et leur assemblage pour obtenir un protocole aux propriétés requises.

I.1.1 Les groupes

La notion de groupe est l'abstraction utilisée pour désigner le mécanisme logique de communication entre plusieurs entités actives. C'est en quelque sorte l'équivalent du concept de connexion utilisée pour les communications point−à−point, mais étendue au cadre des communications multi−points. La notion de groupe est toutefois plus complexe dès

que l'on prend en compte la possibilité de connexion ou de déconnexion dynamique de membres participant à un groupe existant.

Les protocoles utilisés dans ce domaine sont de deux types :

- les protocoles de transport de données, en règle générale on considère que tout message envoyé à un groupe est reçu par tous ses participants valides. Ce sont les protocoles de diffusion, la terminologie anglaise distingue deux notions différentes, le "broadcast" qui envoie les données à tous les sites valides, et le "multicast" qui introduit la possibilité de plusieurs groupes simultanés, et désigne l'envoi à l'un d'entre eux seulement.
- les protocoles servant à maintenir une vision cohérente de la composition du groupe. Il faut en particulier détecter la disparition de participants, et propager la composition du groupe lors des changements vers tous les participants valides.

Ces deux fonctions peuvent être gérées simultanément par un protocole unique, mais la tendance actuelle est à la séparation de ces deux fonctionnalités, les protocoles de transport s'appuient sur la description des groupes que leur fournit le protocole d'appartenance.

Les entités qui forment les groupes peuvent être différentes, on peut par exemple considérer des groupes de machines ou des groupes de processus. Dans le dernier cas, il est possible d'avoir plusieurs membres d'un groupe présents sur un même site, voire tous les membres. Il faut alors prendre en compte les opérations de démultiplexage nécessaires lors de la délivrance d'un message.

I.1.2 Modèles de réseaux

Les réseaux d'interconnexion informatiques ont pour but d'échanger des informations entre des machines physiquement distinctes. Une première classification repose sur les distances couvertes par ces réseaux :

- un LAN (Local Area Network) permet de connecter directement un ensemble de machines comprenant jusqu'à une centaine d'éléments situés dans une zone n'excédant pas quelques centaines de mètres. Typiquement un tel réseau satisfait aux besoins d'interconnexion pour un bâtiment, les technologies standards sont le réseau Token−Ring en anneau et Ethernet qui est constitué de segments interconnectés par des ponts.
- un MAN (Metropolitan Area Network) augmente significativement la distance couverte pour satisfaire aux besoins d'une agglomération. La distance couverte représente alors plusieurs dizaines de kilomètres. Il peut servir soit à connecter des machines isolées, soit permettre des échanges entre réseaux locaux. Le standard dans ce domaine est FDDI, qui est formé par un anneau doublé pour limiter l'effet des pannes.
- enfin un WAN (Wide Area Network) permet d'interconnecter un ensemble de réseaux locaux et métropolitains qui peut couvrir un pays voire le monde entier. La géométrie de type de réseau est très variable, ainsi que les technologies utilisées pour faire cheminer l'information (satellites, câbles, fibre optique ...).

Les propriétés des protocoles de diffusion sont a priori indépendantes du type du réseau et de la topologie du réseau d'interconnexion utilisé, mais les caractéristiques techniques influent énormément sur les possibilités pratiques d'implantation des protocoles. En particulier, les normes de transmission, la qualité de service (délais, bande passante et taux de pertes), et le types de pannes encourues rendent de tels protocoles très complexes lorsque le nombre de sites mis en œuvre et la taille du réseau croissent.

Pour formaliser le comportement "idéal" d'un système distribué, c'est à dire l'ensemble des machines et le réseau d'interconnexion, on se réfère au modèle synchrone qui satisfait aux trois critères suivants :

- les temps de transmission sont bornés. Cela signifie que l'intervalle de temps absolu nécessaire à l'émission d'un message, sa transmission via le réseau, puis sa réception ne dépassera jamais une limite connue.
- tous les sites possèdent une horloge interne, dont la dérive peut être bornée.
- toutes les étapes d'un calcul réparti ont un temps d'exécution qui peut être borné.

Plusieurs remarques peuvent être faites concernant ces trois propriétés. Le premier point ne signifie pas l'absence de perte de messages, mais qu'en l'absence de perte on est certain de pouvoir borner les délais de transmissions. Le deuxième point permet alors de décider de manière formelle si un message a été perdu par le réseau après expiration d'un délai de garde. Enfin le troisième point exige que les systèmes installés sur les divers sites aient un comportement temporel déterministe, ce qui n'est garanti que pour des systèmes de type temps−réel, afin d'éliminer les délais supplémentaires liés par exemple à l'augmentation de la charge pour un système géré en temps partagé.

S'il est possible de construire des systèmes distribués synchrones pour répondre aux besoins de certaines applications industrielles, la majorité des systèmes distribués ne permettent pas de garantir des temps de transmission ou d'exécution bornés. On les désigne sous le nom de systèmes asynchrones. Un des résultats majeurs concernant les systèmes asynchrones est l'impossibilité théorique de fournir un protocole garantissant un consensus. L'énoncé du problème du consensus est le suivant : le système est composé par un ensemble de processus, chacun suggère une valeur et à la fin de l'algorithme tous les processus valides se sont accordés pour choisir l'une d'elles. Ce type de protocoles qui est d'un usage très fréquent lors de l'implantation de systèmes répartis ne peut pas être formellement prouvé si le système est asynchrone. Toutefois, cela n'empêche pas l'existence de tels algorithmes, et leur utilisation dans le cadre de systèmes asynchrones car l'impossibilité théorique est en pratique limitée à des cas extrêmes, statistiquement peu probables, mais non impossibles.

I.1.3 Différents types de pannes

Divers types d'erreurs et de pannes peuvent être rencontrées lors de l'utilisation de systèmes répartis. On peut les classer par type logique, ce qui permet de distinguer :

- les erreurs et pannes matérielles, dont le spectre couvre aussi bien la perte d'un paquet par le réseau d'interconnexion, que l'arrêt complet d'un site suite à la panne d'un élément critique.

- les erreurs et pannes logicielles, qui sont en général plus subtiles tant par la difficulté de leur détection que par leurs conséquences sur le fonctionnement du système. On peut noter par exemple la saturation de tampons conduisant à la perte de messages, jusqu'à l'arrêt complet des communications suite à une erreur de programmation.

Dans le cadre spécifique de la mise en œuvre de protocoles pour les systèmes répartis, nous dressons une liste des problèmes reconnus, classés approximativement par ordre de gravité croissante :

- la perte de paquets isolés : ce type d'incident est fréquent et doit être obligatoirement pris en compte. En effet, les réseaux d'interconnexion ne garantissent pas l'absence de pertes, celles−ci peuvent aussi se produire sur le site émetteur, récepteur ou les routeurs lors d'engorgements dus à une augmentation du débit.
- la perte de paquets successifs peut aussi être dûe à la saturation de tampons en émission ou en réception. Ce type d'erreur est plus difficile à gérer, car il faut conserver plus longtemps les informations transmises pour pouvoir annuler l'effet de cette panne. Si cette situation se prolonge, on risque aussi de conclure hâtivement à la panne des sites non accessibles.
- l'impossibilité d'émettre est une forme aiguë du problème précédent, et peut survenir par exemple lors d'une panne du sous−système d'accès au réseau (pilote ou matériel).
- l'arrêt de fonctionnement d'un site survient lors de la panne d'un élément vital. On peut distinguer l'arrêt dit "fail−stop" où la machine cesse toute activité, mais où il est possible de consulter son état et de diagnostiquer de manière sûre son arrêt, de l'arrêt silencieux qui ne peut être distingué d'une impossibilité d'émettre prolongée.
- le partitionnement du réseau se produit lors d'un dysfonctionnement du réseau conduisant à l'existence de deux réseaux corrects mais ne pouvant plus communiquer entre eux. Ce type de problème est très difficile à gérer car chaque partie peut considérer les autres sites comme stoppés et continuer à fonctionner de manière isolée. On obtient alors deux systèmes indépendants qui vont agir de manière divergente, rendant ainsi très difficile voire impossible le retour à un état stable unique lors de la reprise des communications.

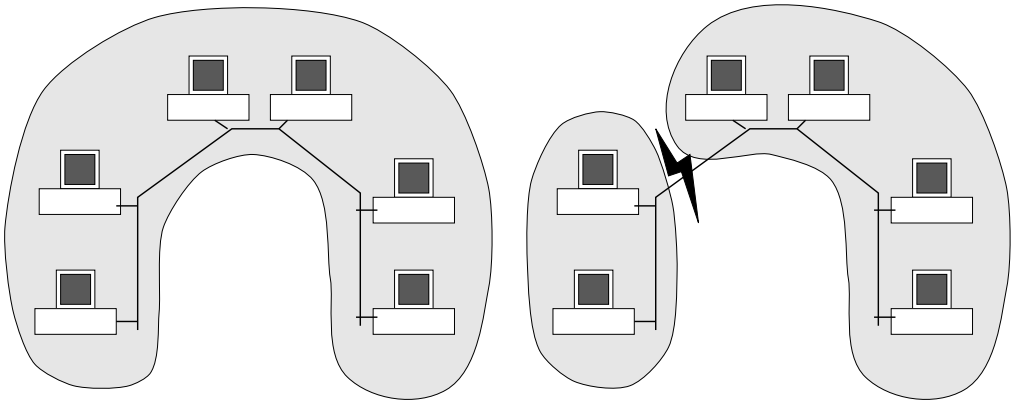


Fig. 1.1 : exemple de partitionnement

- les erreurs byzantines : ce terme recouvre tous les dysfonctionnements liés à un comportement arbitraire d'un composant fautif. Cela recouvre par exemple l'envoi de messages erronés lié au dérèglement de composants logiciels, ou un comportement hors norme d'un composant matériel.
- les fautes intentionnelles qui ne sont pas classifiées comme des pannes mais recouvrent aussi des fonctionnements anormaux, par exemple lorsqu'un des sites du système réparti tente d'induire les autres sites en erreur pour accéder à des données ou des privilèges auxquels il n'a pas normalement accès.

Tout protocole se doit de gérer les pertes occasionnelles de messages pour pouvoir donner lieu à une utilisation pratique. En fonction du contexte d'utilisation prévu, il pourra alors être nécessaire de pallier les pannes de sites ou le silence prolongé de sites si on se place dans le cadre de systèmes asynchrones. L'utilisation du protocole sur des réseaux à grande échelle nécessite de prendre en compte les partitions qui peuvent par exemple survenir lors de la panne de routeurs, par contre les cas de partitions sont plus rares dans le cadre de réseaux locaux. Dans le cas d'un segment Ethernet on peut constater qu'une rupture de connexion annule toute transmission sur le segment, et il n'y a pas de partition mais un ensemble de sites isolés. Enfin l'utilisation du protocole dans des environnements critiques (matériel embarqué ou gestion de données vitales) nécessite de prendre en compte les cas d'erreurs byzantines, et entraîne l'emploi de matériel spécifique

Après cette présentation générale des réseaux informatiques et des différentes pannes auxquels ils sont exposés, nous détaillons dans la suite de cette section les différentes propriétés que peuvent assurer les protocoles de diffusion.

I.1.4 Fiabilité

Un protocole est dit *fiable* s'il assure lors de la réception d'un message que :

− soit tous les destinataires du message le reçoivent,

− soit aucun ne le reçoit.

Cette propriété est essentielle pour assurer la cohérence d'une transaction mettant en œuvre la duplication de données. Cependant elle n'impose rien quand à l'ordre de réception de

plusieurs messages successifs. Toutes les propriétés suivantes concernant l'ordonnancement supposent que le protocole est fiable.

I.1.5 Ordre FIFO

Cela consiste à assurer que si deux messages A et B sont émis successivement depuis un même site et sont tous deux reçus sur un autre site, l'ordre de réception des messages coïncide avec leur ordre d'émission.

La figure suivante utilise la représentation standard utilisée en algorithmique répartie pour représenter des processus communiquant par le biais de messages. Chaque processus est représenté par un axe temporel sur lequel chaque événement lié au processus est représenté. Les messages sont symbolisés par des flèches reliant les axes, le point de départ est l'événement associé à l'émission du message, le point d'arrivée est la réception dudit message. Les messages en diffusion sont symboliquement représentés comme un ensemble de flèches ayant un même point de départ, car il n'y a qu'une émission.

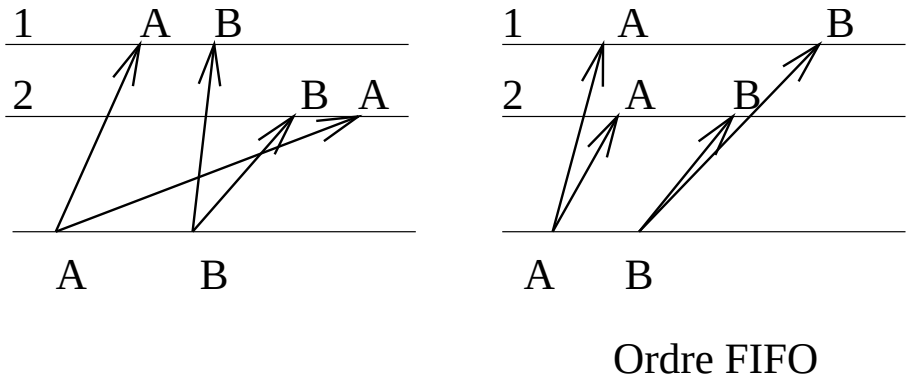


Fig. 1.2 : ordre FIFO

Dans la figure Fig. 1.2 on remarque une rupture de l'ordre FIFO sur le site 2 qui reçoit les messages B puis A ce qui ne correspond pas à leur ordre d'émission.

Pour illustrer par l'exemple les diverses propriétés des protocoles nous utiliserons par la suite l'exemple de la diffusion des "News" d'Internet. C'est sans doute le protocole de diffusion informatique le plus massivement utilisé, et il est aisé d'y puiser des exemples significatifs de dysfonctionnements.

Supposons qu'un participant diffuse un premier message, puis se rendant compte d'une erreur, fournisse une correction ou une addition au message initial. Si le protocole de diffusion utilisé est FIFO, il garantit que la correction relative au premier message ne sera délivrée aux lecteurs qu'après le message initial. Il s'agit d'une propriété qui assure la délivrance des messages dans un certain contexte.

I.1.6 Ordre causal

Malheureusement il est généralement impossible de comparer les dates d'émission de messages provenant de sites différents. Pour garantir la validité du contexte de délivrance des messages on utilise l'*ordre causal* qui est un ordre partiel induit par la fermeture transitive des deux règles de dépendance suivantes :

- la réception d'un message *dépend* de son émission.
- sur un même site, deux événements sont liés par l'ordre total de la dépendance temporelle (l'ordre total temporel ne peut pas en pratique être reconstruit si plusieurs sites sont en jeu).

Cet ordre, définit par Lamport [1], correspond à la notion intuitive de causalité c'est−à−dire une dépendance temporelle naturelle (l'émission d'un message précède forcément sa réception et l'ordre d'émission sur un site) qui est augmentée par une succession de dépendances induites.

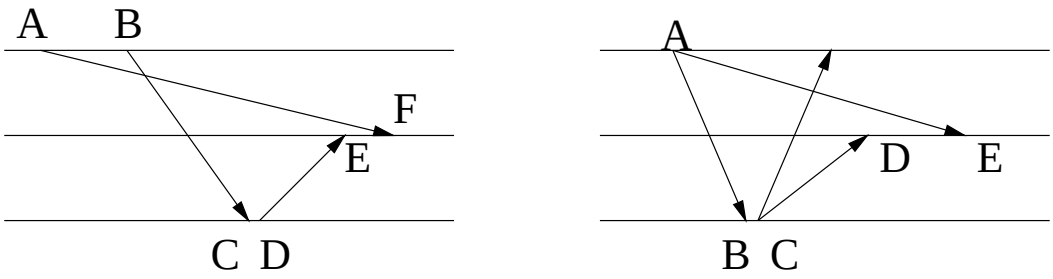


Fig. 1.3 : violations de l'ordre causal

Dans le cas de la figure précédente, deux exemples de violation d'ordre causal sont présentés. La figure de gauche illustre une violation d'ordre causal lors de l'utilisation de messages point−à−point. Le message reçu en E dépend causalement du message émis en A suivant la chaîne A−B−C−D−E, et sa réception antérieure à celle du message initial en F viole la causalité. La figure de droite montre un exemple du même type mais utilisant des messages en diffusion ; dans cet exemple c'est la délivrance en D, donc antérieure à E, d'un message dépendant causalement du message émis en A qui pose problème.

Pour reprendre l'exemple de la distribution des news, l'ajout d'un filtrage forçant une délivrance en ordre causal assurerait que pour tout message lu, les messages auxquels il fait référence ont déjà été délivrés. Le principe fondamental de l'ordre causal est d'assurer que le contexte dans lequel un message est délivré reste cohérent malgré les temps de transmission variables.

I.1.7 Atomicité

Un protocole assure l' *atomicité* s'il garantit que deux messages reçus dans un certain ordre sur un site ne pourront pas être reçus dans un ordre différent sur un autre site. Cette propriété est aussi connue sous le nom d'ordre total.

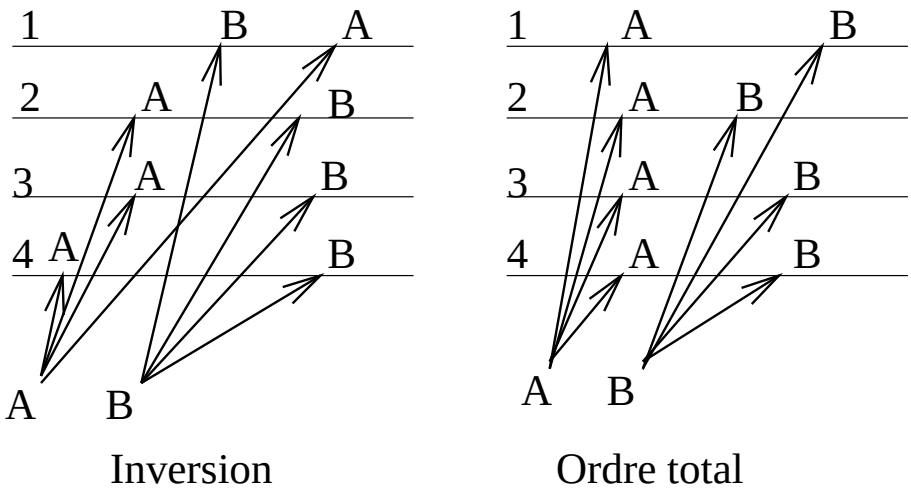


Fig. 1.4 : inversion et diffusion atomique

Par exemple, sur le premier schéma de la figure Fig. 1.4, on constate que l'atomicité n'est pas respectée. En effet, le site 1 reçoit la séquence "B puis A" alors que le site 2 reçoit la séquence "A puis B".

Si la distribution des news était atomique, il serait possible d'énumérer les messages en fonction de leur ordre d'arrivée. L'atomicité assurerait que sur tous les sites, la séquence des messages et leurs estampilles associées soient identiques, il serait alors possible de référencer un article par son seul numéro.

I.1.8 Résistance aux pannes de sites

Le protocole doit aussi assurer la résistance aux pannes, c'est−à−dire garantir la sémantique des envois (diffusion fiable et ordonnée à tous les membres actifs), même en cas de défaillance prolongée de l'émetteur ou des destinataires. La plupart des algorithmes permettent de choisir un facteur de résistance de la forme : "algorithme résistant à la défaillance de K membres", c'est−à−dire permettant de reformer le groupe constitué des machines qui ne sont pas en panne si moins de K pannes de site se produisent simultanément, tout en conservant les propriétés d'atomicité et d'ordonnancement des messages.

I.1.9 Composition

Il est possible de construire des protocoles de diffusion répondant à un ensemble des critères décrits dans les sous−sections précédentes. Il faut juste remarquer que fournir un ordonnancement causal garantit implicitement l'ordre FIFO. La figure ci−après illustre l'ensemble des services possibles :

	Fiable	FIFO	Causal
Fiable	Fiable	FIFO	Causal
Ordre Total	Atomique	FIFO Atomique	Causal Atomique

Fig. 1.5 : Différents ordonnancements possibles

Après cette présentation générale des protocoles de diffusion, leurs propriétés et des différentes sémantiques liées à l'ordonnancement, la section suivante présente les principaux protocoles et projets de recherche sur le sujet.

I.2 Exemples

Le domaine des protocoles de diffusion et leur utilisation dans les systèmes répartis est un sujet de recherche actif depuis plus d'une dizaine d'années. La littérature et les exemples de prototypes sont donc foison et il est impossible de faire une couverture exhaustive de l'ensemble des publications disponibles. Nous nous bornons donc à présenter les algorithmes et les réalisations les plus significatives, suivant un ordre chronologique :

I.2.1 Protocole de Chang et Maxemchuck

Ce protocole, présenté dès 1984 [2], est basé sur l'utilisation d'un serveur de séquencement appelé séquenceur. Il offre un mécanisme de diffusion fiable atomique et résistant aux pannes de site.

La base du protocole consiste en l'atomicité garantie à l'aide d'un site séquenceur. Le site émetteur envoie le message à diffuser au séquenceur, puis le séquenceur diffuse le message après lui avoir accordé une estampille unique. Le séquenceur acquitte alors positivement le message qu'il a reçu et le conserve dans un historique avec son estampille. Les receveurs peuvent découvrir la perte d'un message en remarquant un saut dans l'estampillage des messages reçus ; ils peuvent alors demander au séquenceur le ou les messages manquants. La figure suivante illustre ce principe de base.

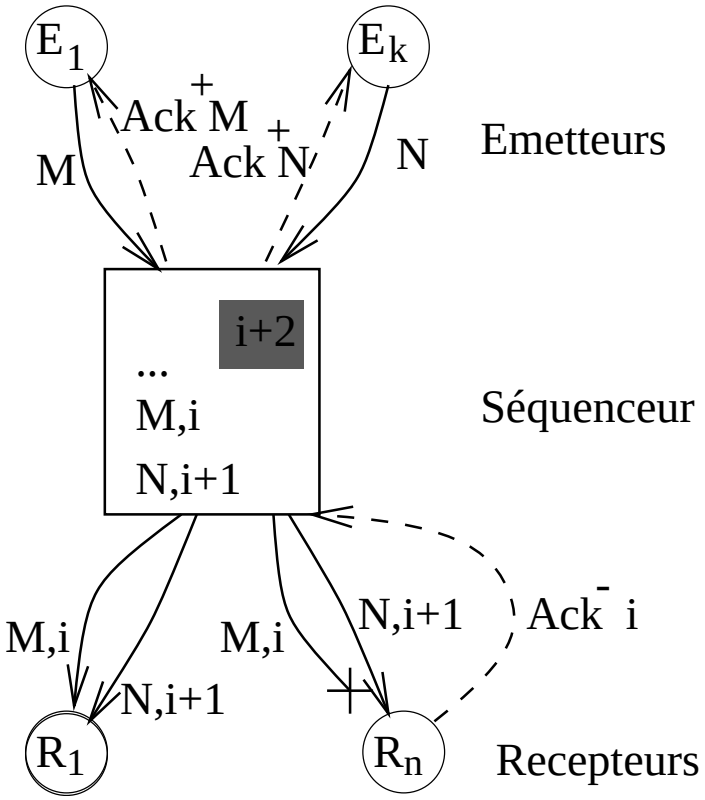


Fig. 1.6 : rôle du séquenceur

A la réception du message N assorti de l'estampille i+1, le site Rn remarque qu'il a perdu le message i : il bloque momentanément la délivrance de N et envoie un acquittement négatif au séquenceur qui lui renverra le message M.

Tel quel, ce protocole souffre de deux défauts majeurs :

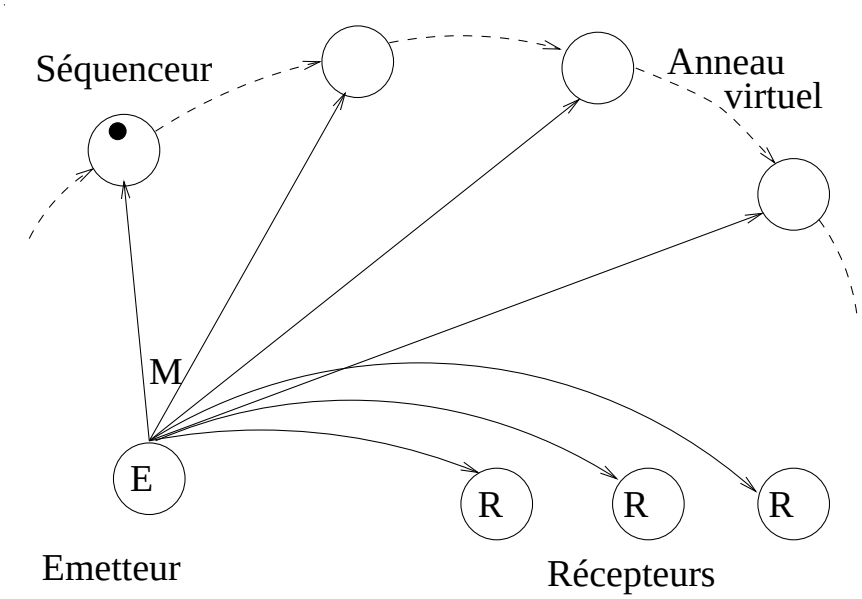
- avec le mécanisme d'acquittement négatif des receveurs, le séquenceur n'a pas le moyen de détecter quand tous les destinataires d'un message l'ont reçu. Il ne peut donc pas décider quand détruire un message dans son historique ; c'est le problème du "buffer infini".
- si le séquenceur disparaît, tous les messages qui n'ont été reçus par aucun des récepteurs et dont le séquenceur a envoyé l'acquittement à l'émetteur sont perdus.

Ces problèmes disparaissent lorsque l'on fait circuler le site séquenceur parmi tous les sites valides. Il s'agit d'un mécanisme à jeton où le site qui détient le jeton joue le rôle du séquenceur. L'ensemble des sites valides forme un anneau virtuel et le jeton circule sur cet anneau toujours dans le même sens. Le jeton est transmis avec un acquittement positif et le site qui le reçoit ne l'accepte que s'il est à jour pour les messages reçus. Le problème du "buffer infini" disparaît alors car quand un site accepte le jeton, il sait que tous les messages antérieurs à sa dernière acceptation du jeton ont été reçus par tous les sites, il peut donc les supprimer de son historique. De plus, si on n'envoie l'acquittement d'un message à son émetteur qu'après que le jeton ait été accepté par L sites, on est assuré que L + 1 sites

possèdent une copie du message dans leur historique : le protocole résiste alors à la panne de L sites.

Le protocole définitif est donc formé de trois phases :

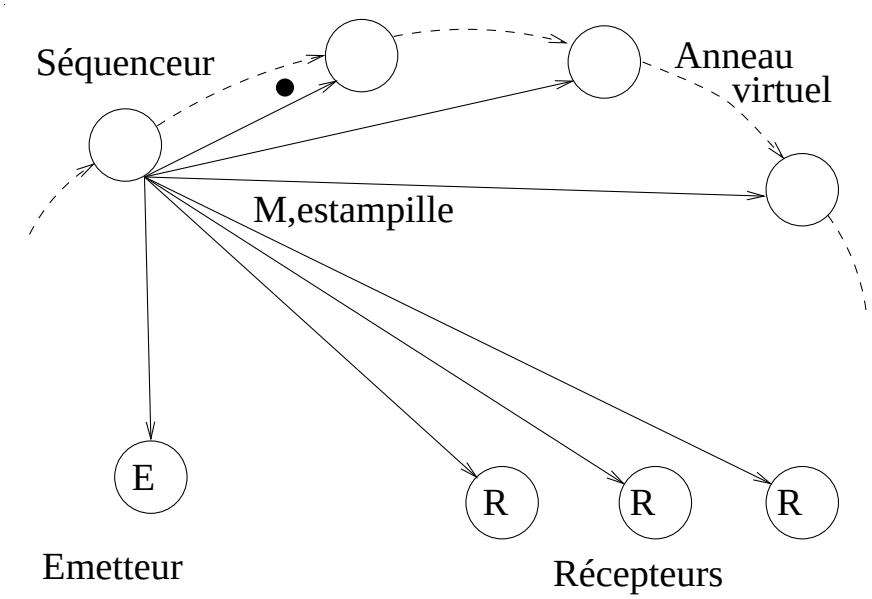
- L'envoi du message en diffusion physique :



Emission en diffusion physique du message

L'émetteur attend une réponse du site séquenceur ; si nécessaire, le message est renvoyé après un délai d'attente.

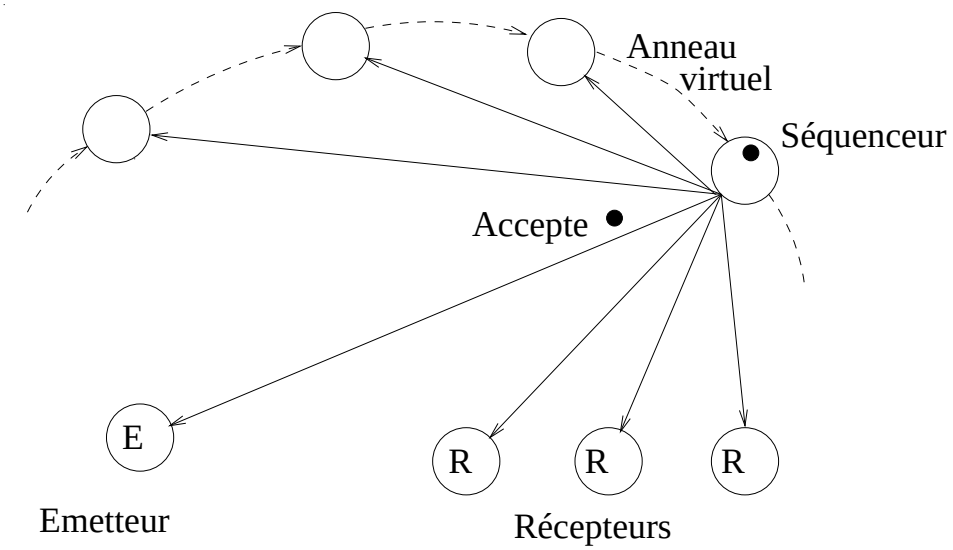
- L'estampillage du message par le séquenceur : le message, envoyé en diffusion physique, sert aussi à effectuer l'acquittement positif vers l'émetteur. Il est aussi utilisé pour transmettre le jeton ou acquitter sa bonne réception.



Estampillage en diffusion physique du message

Les sites qui reçoivent ce message le conservent dans un historique avec l'estampille associée.

- La délivrance du message a lieu lorsqu'au moins L passages de jeton ont été acceptés (dans la figure suivante L vaut 3).



Délivrance des messages après L transmissions du jeton

Le déplacement du jeton peut être suivi car la réception du jeton est validée par un acquittement positif placé dans un message à diffusion (par exemple un

estampillage). C'est lors de la réception du jeton qu'un site peut supprimer de son historique les messages reçus par tous les autres sites.

La détection d'une panne de site peut facilement être réalisée lors du passage du jeton à ce site ou lorsque le site séquenceur ne répond plus. L'ensemble des sites entre alors dans une phase de récupération où l'anneau virtuel est reformé et où un nouveau site séquenceur est élu.

Comme on peut le constater, une famille de protocoles est décrite par les valeurs choisies pour le facteur de résistance aux pannes de sites et la fréquence de rotation du jeton. Le choix du facteur de résistance aux pannes de sites est un compromis entre la fiabilité du protocole et la rapidité de transmission ; en effet, si on augmente L, les sites receveurs devront attendre plus longtemps avant de délivrer le message. Une vitesse de rotation du jeton faible ralentit le fonctionnement du protocole si L est non nul et augmente la taille de l'historique. Par contre, une vitesse de rotation trop rapide ralentit aussi le protocole et risque de saturer la bande passante du réseau à cause des messages de contrôle.

Nombre de rotations du jeton	Nombre de messages de contrôle	Taille de l'historique
1 par acquittement	1 ou 2	N - 1
1 pour K acquittements	tend vers 1 quand K $\rightarrow \infty$	K * (N - 1)
K par acquittement	K ou K + 1	(N - 1) / K

Fig. 1.7 : différentes familles du protocole

Ce protocole, surtout connu pour des raisons historiques s'avère en fait assez lent car la circulation constante du jeton et le mécanisme de résistance aux pannes sont coûteux.

I.2.2 ISIS

ISIS [3] [4] [5] [6] est un ensemble d'outils destinés à faciliter la programmation dans le cadre d'un système réparti, en particulier dans les domaines de l'algorithmique distribuée et de la tolérance aux pannes. L'ensemble des mécanismes de communication repose sur deux notions essentielles :

- Le principe de *groupes de processus virtuellement synchrones*, c'est−à−dire un groupe de processus coopérant à une même tâche.
- Plusieurs protocoles de diffusion permettant à un groupe de processus de communiquer de manière fiable suivant des sémantiques différentes.

Les deux protocoles suivants ont la particularité d'être basés sur un mécanisme d'estampillage vectoriel permettant de garantir l'ordonnancement causal. Chaque processus

possède un vecteur de temps VT, et chaque message m envoyé au groupe porte une estampille vectorielle Vm. Un vecteur V est formé de *n* estampilles où *n* est le nombre de processus appartenant au groupe. Les règles d'utilisation de ces vecteurs sont les suivantes :

- Le vecteur VT(i) d'un processus i est initialisé à zéro.
- Lors de l'émission d'un message *m* par un processus i, la composante VT(i)[i] est incrémentée.
- Un message émis par un processus i est daté par VT(i) (après incrémentation de la composante de l'émetteur).
- Lors de la délivrance d'un message m à un processus j, le vecteur de temps du processus receveur VT(j) est réactualisé en prenant le maximum composante à composante de VT(j) et Vm :

  $\forall k \in [1 .. n]$, $\quad VT(j)[k] = Max (VT(j)[k], Vm[k])$.

On définit alors les relations d'ordre partiel sur ces vecteurs par :

$$VT1 \leq VT2 \text{ ssi } \quad \forall k \in [1 .. n] \quad VT1[k] \leq VT2[k] \qquad (1)$$

$$VT1 < VT2 \text{ ssi } \quad VT1 \leq VT2 \quad \text{et} \quad \exists i \text{ tq } VT1[i] < VT2[i] \qquad (2)$$

On peut alors vérifier que si deux messages m et m' portent respectivement les estampilles Vm et Vm' et si m' dépend causalement de m au sens de I.1.6 alors VT(m) < VT(m'). Ce mécanisme, initialement conçu dans le but d'assurer le respect de l'ordre causal pour des communications point−à−point, est utilisé ici avec des messages à diffusion.

I.2.2.1 CBCAST

Le protocole CBCAST (pour Causal Broadcast) garantit le respect de l'ordonnancement causal pour des messages à diffusion à l'aide d'horloges vectorielles. Soit un message M émis par un site s, daté par une estampille VM et reçu sur un site i possédant un vecteur temps VT(i). La délivrance du message M est bloquée tant que l'une des deux conditions suivantes est vraie :

- $\exists j \in [1 .. n]$ avec $j \neq s$ et $VM[j] > VT(i)[j]$
- $VM[s] > VT(i)[s] + 1$

La première condition signifie que le site receveur n'a pas encore délivré un message dont dépend causalement M et émis depuis un site j différent de s. La deuxième signifie qu'un des messages émis précédemment par s n'a pas été reçu sur le site i.

La figure suivante Fig. 1.8 explique le fonctionnement de ce mécanisme.

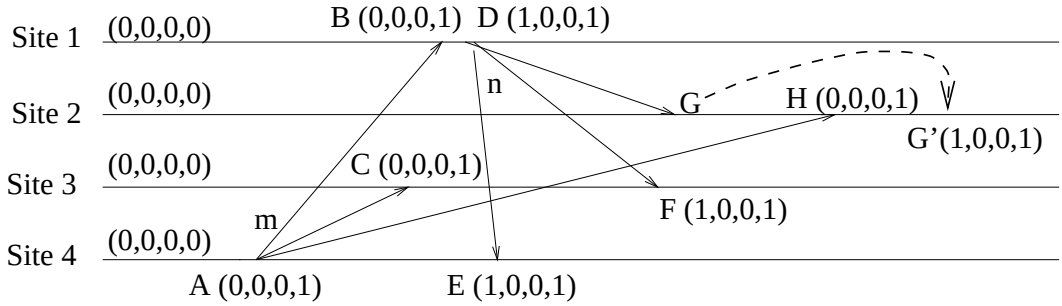


Fig. 1.8 : utilisation d'horloges vectorielles

Dans cet exemple, le site 1, après avoir reçu le message m (0,0,0,1) en provenance du site 4 émis à l'instant A(0,0,0,1), émet à l'instant D (1,0,0,1) un message à diffusion n estampillé par (1,0,0,1). D'après les règles définissant la dépendance causale, le message n dépend donc causalement du message m. Le site 2, qui n'a pas encore reçu le message m, détecte qu'il ne peut pas délivrer immédiatement le message n en G car la quatrième composante du vecteur temps en G est 0 alors que celle de l'estampille du message n est 1. Le site 2 peut en déduire qu'il n'a pas encore reçu un message en provenance du site 4 dont dépend causalement n, à savoir m. La délivrance de n est alors retardée et a lieu en G' après la réception de m en H.

I.2.2.2 ABCAST

ABCAST est un protocole garantissant une délivrance atomique et résistant aux pannes de sites. La dernière réalisation de ce protocole est basée sur CBCAST qui peut être utilisé pour construire un protocole garantissant l'atomicité de la diffusion. La technique utilisée consiste en un protocole à deux phases :

- le site émetteur diffuse le message m à l'aide du protocole CBCAST ; toutefois, m n'est pas délivré immédiatement.
- un site privilégié diffuse alors à l'aide de CBCAST un message donnant l'ordonnancement des derniers messages ABCAST reçus et non délivrés.
- à la réception du message d'ordonnancement, les sites peuvent alors délivrer les messages ABCAST ainsi ordonnés.

L'originalité de ce protocole est d'assurer à la fois l'atomicité de la diffusion des messages et de préserver l'ordre causal lors de l'utilisation simultanée de CBCAST et ABCAST. L'utilisation d'un site privilégié délivrant un estampillage unique des messages est la méthode la plus courante pour garantir l'ordonnancement total. Toutefois, le protocole de Melliar−Smith, Moser et Agrawala (I.2.3) présente une démarche originale visant à obtenir l'atomicité sans recourir à cette technique.

I.2.2.3 GBCAST

Le protocole GBCAST (Group Broadcast) est un protocole supplémentaire qui permet d'effectuer des envois de messages qui paraissent atomiques du point de vue des deux

autres protocoles ABCAST et CBCAST. Le principal intérêt de ce protocole est le maintien de la cohérence de la composition d'un groupe sur les divers sites qui y participent. Par exemple après la détection de la mort d'un site, le protocole GBCAST est utilisé pour mettre à jour la nouvelle composition du groupe, de manière atomique par rapport aux autres protocoles utilisés pour la diffusion de messages au sein du groupe. La figure suivante illustre l'emploi de GBCAST :

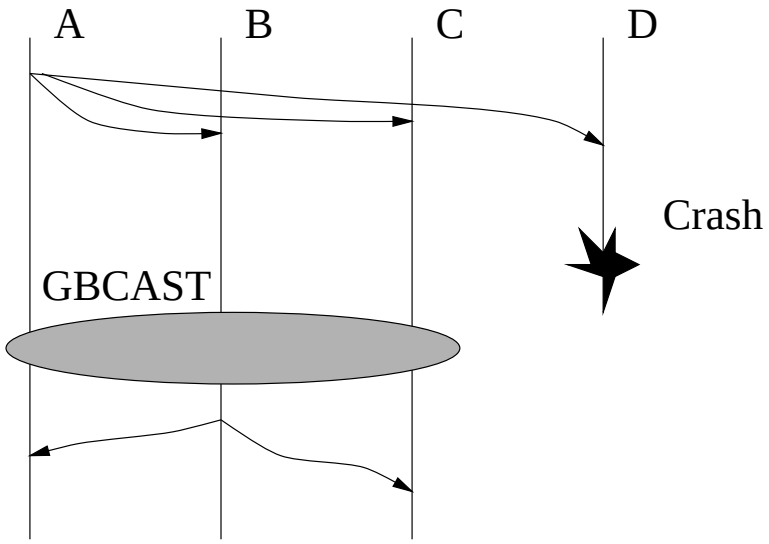


Fig. 1.9 : Utilisation de GBCAST pour le maintien des informations de groupe

Une fois la mort du site D détectée par les autres membres, la mise à jour de la nouvelle liste des membres du groupe est diffusée de manière atomique à l'aide de GBCAST.

Ce protocole n'est plus cité dans les papiers récents : en effet le mécanisme de maintien de la liste des membres du groupe a évolué fortement, en partie pour pouvoir supporter des groupes à grande échelle [7], c'est−à−dire pouvant comporter un grand nombre de sites ou des sites distribués sur un grande échelle géographique.

I.2.3 Algorithme de Melliar−Smith, Moser et Agrawala

La majorité des protocoles de diffusion existants nécessite de nombreux échanges de messages pour garantir la bonne réception et un ordonnancement cohérent des messages. Les deux protocoles suivants "Trans" et "Total" [11] sont exclusivement basés sur l'envoi de messages en diffusion physique. Le principe fondamental de fonctionnement consiste, lors de l'envoi d'un message en diffusion, à y adjoindre des informations concernant les messages reçus précédemment. L'ensemble des informations ainsi collectées permet d'extraire un graphe de dépendance entre messages utilisé pour déterminer si un message peut être délivré localement.

Le fonctionnement des protocoles est basé sur un mécanisme d'acquittements positifs et négatifs. Un acquittement est transitif : si un site S acquitte un message M, il n'a pas besoin d'acquitter directement tous les messages qu'acquittait M. Pour illustrer ce fonctionnement

considérons un ensemble de quatre stations A, B, C et D. Soit la séquence de messages (à diffusion) suivante :

$$B_1 \ ; \ D_1 \ ; \ A_1 d_1 \ ; \ C_1 \bar{d}_1 b_1 a_1 \ ; \ D_2 \bar{a}_1 c_1 \ ; \ D_1 \ ; \ C_2 d_1 d_2 \ ; \ B_2 \bar{a}_1 c_2$$

Les deux premiers messages correspondent à l'envoi des deux premiers messages de B et D respectivement. Quand A en envoie un à son tour, il acquitte alors le message D_1 qu'il a reçu en joignant un acquittement positif d_1. Le site C qui a bien reçu $A_1 d_1$, se rend alors compte qu'il a perdu le message D_1. C'est pourquoi il joint un acquittement négatif $\bar{d}_1$ au message C_1. On peut voir les effets de la transitivité des acquittements lors de l'envoi du message D_2 : l'acquittement du message C_1 valide aussi par transitivité les messages A_1 et B_1. On notera aussi qu'un message émis depuis un site S constitue un acquittement de tous les messages émis précédemment par le site S. L'ensemble des acquittements peut être vu sous la forme d'un graphe dont les nœuds sont des messages (voir figure suivante Fig. 1.10).

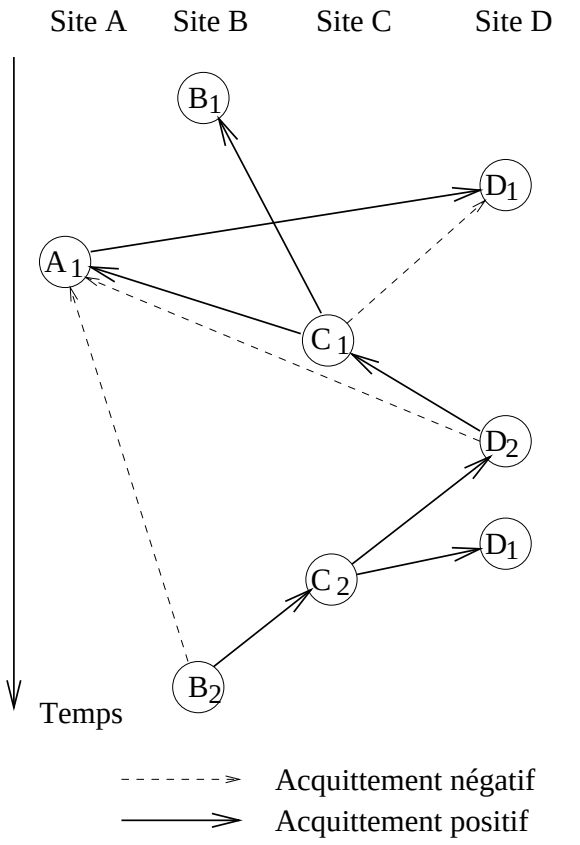


Fig. 1.10 : graphe des acquittements

Au fur et à mesure de la réception des messages chacune des stations va pouvoir construire ce graphe, car tous les messages sont émis en diffusion physique (en cas de perte de message, la vision complète du graphe ne sera possible qu'après la retransmission des

messages manquants). Les deux protocoles Trans et Total utilisent ce graphe pour déterminer quand la délivrance ou la destruction sur le site émetteur d'un message peut avoir lieu.

I.2.3.1 Trans

Ce protocole ne donne aucune garantie d'ordonnancement. En effet les messages sont délivrés dès leur réception. Il résout toutefois le problème du stockage des messages sur un site émetteur de manière élégante. On utilise le graphe des acquittements pour déterminer quelles sont les stations qui ont reçu un message donné. Le prédicat de base sur lequel est basée cette recherche est :

OPD (P,A,C) : le processeur P est certain que le processeur qui a émis le message C a reçu et validé le message A au moment où il émettait le message C.

La méthode pour construire ces prédicats consiste à trouver dans le graphe des acquittement un chemin de C à A dont aucun noeud n'est acquitté négativement par C. On peut aisément déduire de l'ensemble des prédicats si l'ensemble des sites a délivré et validé un message donné et donc s'il peut être détruit.

La recherche des prédicats permet de plus à chaque site de construire un ordonnancement partiel défini par l'unique relation suivante :

Sur le processeur P, on a $C \rightarrow B$ au sens de cet ordre partiel si et seulement si OPD (P,B,C) et pour tout message A, OPD (P,A,B) $\Rightarrow$ OPD (P,A,C).

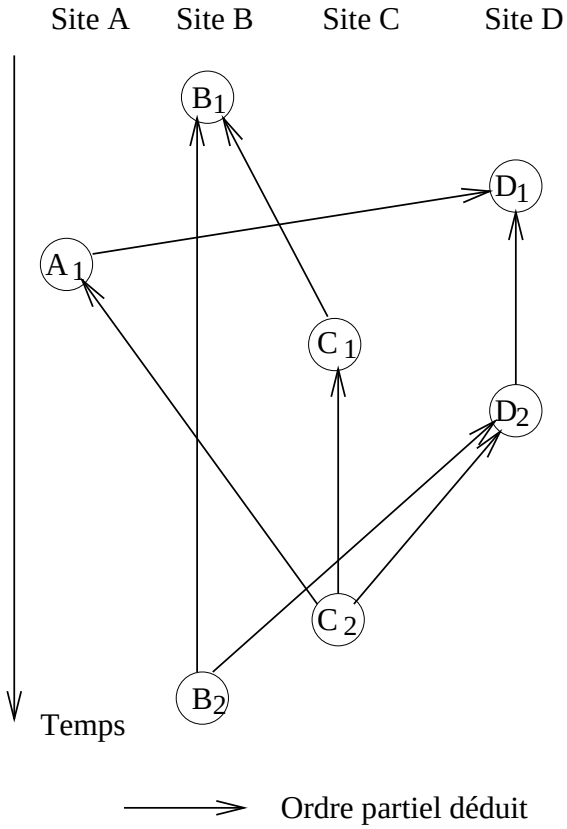


Fig. 1.11 : ordre partiel déduit du graphe des acquittements

La figure précédente représente l'ordre partiel que l'on obtient après analyse du graphe des acquittements de la figure Fig. 1.10. L'intérêt de cet ordre partiel vient de la propriété suivante : au moment de l'envoi d'un message C par un processeur P, P a délivré et envoyé un acquittement de tous les messages précédant C dans l'ordre partiel. Tous les sites, au fur et à mesure de la réception des messages, vont donc construire cet ordre partiel (sauf sur les sites en panne ou en cas de partition).

I.2.3.2 Total

Ce protocole, basé sur les mécanismes de Trans, extrait un ordre total de l'ordre partiel précédent et les messages ne sont délivrés sur une station que lorsque leur place dans l'ordre total est déterminée. Sous la condition que tous les sites construisent le même ordre total à partir de l'ordre partiel, ce protocole assure l'atomicité de la diffusion des messages sans utiliser de messages de contrôle supplémentaires et sans recourir à un site assurant un estampillage centralisé.

L'algorithme permettant d'extraire un ordre total depuis l'ordre partiel est trop compliqué pour avoir sa place dans ce rapport. Son principe s'appuie sur les propriétés topologiques de la forêt (de graphes) décrivant l'ordre partiel. Comme chaque processeur construit le même ordre partiel, à condition bien sûr que le réseau se comporte normalement, on peut prouver que chaque processeur extrait le même ordre total. Malheureusement, la décision de placer

un message dans l'ordre total nécessite l'acquisition d'un grand nombre d'informations et donc la réception de beaucoup de messages avant de pouvoir en délivrer un (par exemple, dans le cas d'un réseau de 10 stations, il faut en moyenne recevoir 7 messages avant de pouvoir décider de l'ordre du premier). De plus, rien n'empêche théoriquement que la place d'un message dans l'ordre total soit indécidable, bien que la probabilité d'un tel blocage diminue très fortement avec la réception des messages suivants. Enfin, le bon fonctionnement du protocole nécessite une émission régulière de tous les processeurs en présence ; il s'accommode mal par exemple d'une situation où un seul processeur émet en continu, les autres restant passifs.

En conclusion, ces deux protocoles proposent des solutions originales pour résoudre les problèmes classiques liés aux algorithmes répartis. Le protocole "Total" offre en particulier une diffusion atomique sans avoir recours à un site central et sans nécessiter de messages de contrôle. Malheureusement, son mécanisme est passif et le temps entre l'émission et la délivrance du message semble trop important pour qu'il puisse être utilisé efficacement pour un mécanisme d'exécution répartie.

I.2.4 Delta−4

Le projet Delta−4, tout comme ISIS a pour but de fournir les mécanismes de communication et d'exécution permettant de construire des applications industrielles fonctionnant sur des architectures réparties. Mais le projet Delta−4 est plus spécifiquement conçu pour le support d'applications tolérantes aux pannes et les systèmes temps−réel. Sa mise en œuvre s'appuie sur des infrastructures de réseaux locaux standard, telles que Token−Bus, Token−Ring ou FDDI. Delta−4 nécessite par contre que les sites communicants soient connectés au même réseau local. En effet, l'ensemble des protocoles fournis s'appuient sur un protocole de diffusion atomique AMp (Atomic Multicast protocol) qui utilise les possibilités de diffusion physique offertes par les spécifications ISO de ces différents réseaux. Le protocole s'appuie fortement sur la notion de délais de transmission bornés, et l'utilisation d'adaptateurs spéciaux assurant la propriété de silence en cas de panne (fail−silent) des sites participant au protocole.

L'ensemble du projet se décline sous deux formes différentes, Delta−4 OSA qui suit le modèle de communication de l'ISO, fournit la notion de destinataires multiples et une couche de type session qui gère les mécanismes de groupes en suivant le standard. La version Delta−4 XPA s'affranchit de la compatibilité vis−à−vis de l'ISO en réduisant les couches de protocoles. Il est aussi possible d'utiliser des interfaces matérielles spécifiques intégrant une grande partie des protocoles, ce qui améliore d'autant les performances.

Pour pouvoir s'adapter aux spécificités des applications, diverses qualités de service sont proposées basées sur les critères suivants :

- des garanties de bonne réception sur une certaines parties des sites destinataires, par exemple au moins N ou tous les membres du groupe.
- le maintien ou non de la sémantique de l'envoi en cas de la panne du site émetteur. C'est ce qui distingue les deux qualités de service "BestEffort" et "atLeast".

- des garanties d'ordonnancement des messages, tels qu'ordre FIFO, causal ou atomique.

Le tableau suivant résume les différentes qualités de service disponibles et les propriétés qui y sont associées :

Propriétés / QdS	Délivrance		Perte de l'émetteur	Ordre garanti
	> N	Tous		
BestEffort	X			FIFO
AtLeast	X		X	FIFO
Reliable		X	X	FIFO
Causal		X	X	Causal
Atomique		X	X	Atomique

Fig. 1.12 : Différentes qualités de service disponibles dans Delta−4

Le protocole de diffusion utilisé est basé sur un algorithme à deux phases assurant qu'un nombre suffisant de destinataires ont bien reçu le message :

- dans la première phase de dissémination, l'émetteur envoie en diffusion physique les données à transmettre. Tous les sites destinataires envoient un acquittement au site émetteur lors de la réception du message. Si l'émetteur ne reçoit pas les réponses dans le temps imparti, la phase de dissémination est redémarrée.
- lorsqu'un nombre suffisant d'acquittements ont été reçus, ce nombre dépendant de la qualité de service requise, l'émetteur diffuse un message de délivrance (accept). Si les acquittements sont négatifs, le site émetteur peut être amené à annuler son message à l'aide d'un message spécifique (reject).

La mise en œuvre des différentes qualités de service s'appuie sur ce protocole de base en faisant varier le nombre d'acquittement nécessaires. Les garanties concernant l'ordonnancement sont fournies par des couches de protocole supplémentaires.

I.2.5 x−Kernel

Le projet x−Kernel [22] mené à l'Université d'Arizona étudie les problèmes liés à la mise en œuvre de protocoles de communications. Dans un premier temps un noyau de système d'exploitation spécifique a été construit (d'où le nom du projet) fournissant les mécanismes nécessaires à une implantation aisée et efficace de protocoles de communications. Dans ce cadre, un protocole de diffusion Psync [24] respectant l'ordre causal a été réalisé ainsi

qu'une réécriture de nombreux protocoles normalisés tels que TCP/IP mais réécris dans le cadre de x−Kernel [23].

La deuxième phase du projet est basée sur le portage de l'environnement du noyau vers des architectures plus classiques. L'ensemble du logiciel est désormais disponible sur des plates−formes standard telles que des stations de travail Sun ou sur les systèmes basés sur le micro−noyau Mach. Les directions actuelles de recherches sont orientées vers les architectures des protocoles et leur intégration dans les systèmes modernes, le support des réseaux à haut débit et les protocoles nécessaires aux nouvelles applications qui en tirent parti. En particulier les directions suivantes sont explorées :

- utilisation des réseaux FDDI et ATM [26] dans le cadre du x−Kernel.
- découplage du protocole sous la forme d'entités logiques distinctes en particulier dans le cadre du micro−noyau Mach.
- analyse et minimisation des mouvements de données lors de la mise en œuvre de protocoles.
- proposition de nouveaux algorithmes dans le domaine de la sécurité, des mécanismes de diffusion fiable, et du transport d'images.
- utilisation de ces protocoles pour le calcul réparti et la construction de systèmes tolérant les pannes [25].

La sous−section suivante détaille le protocole Psync qui est le mécanisme de base sur lequel s'appuient la plupart des protocoles multi−points développés dans le cadre du x−Kernel.

I.2.5.1 Le protocole Psync

Le principe est similaire à celui de l'algorithme de Melliar−Smith, Moser et Agrawala présenté dans la sous−section I.2.3. Sa publication est antérieure, mais il n'utilise que des informations d'acquittement positif. Il s'appuie sur la construction d'un arbre de dépendance local à partir d'informations de causalité transportées avec chaque message. Chaque participant au groupe construit une vue locale en fonction des messages qu'il a envoyés et reçus. La figure suivante illustre la relation d'ordre induite par les dépendances entre messages notée G et la fermeture transitive de celle−ci $G_<$ lors de l'envoi de 5 messages, m1 puis m2 et m3 après réception de m1, puis m4 après réception de m3 et enfin m5 après réception de m2 et m4.

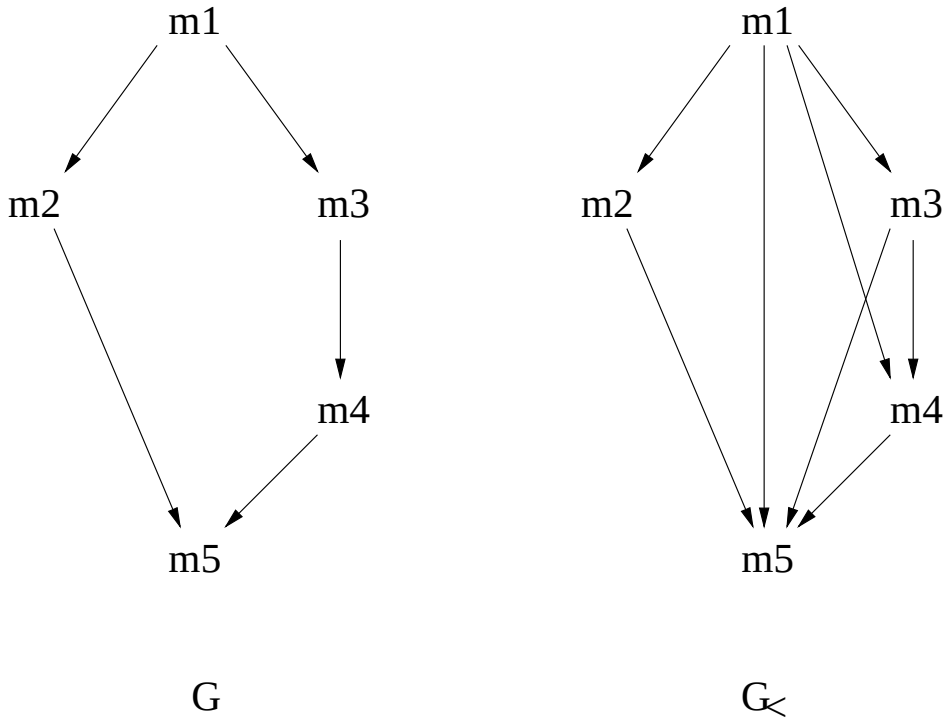


Fig. 1.13 : exemple d'un graphe de dépendance et sa fermeture transitive

Chaque site maintient en permanence une copie locale du graphe de dépendances, qui est une vue restreinte du graphe global, limitée aux messages émis et reçus sur ce site. Tout message émis transporte les étiquettes associées aux messages dont il dépend dans le graphe local. Lors de la réception, on s'assure que tous les messages correspondant aux étiquettes ont bien été délivrés sur ce site, faute de quoi le message est stocké jusqu'à ce que cette condition devienne vraie.

Ce protocole permet d'assurer simplement une délivrance respectant l'ordre causal, et peut facilement être étendu pour résister aux pertes de messages. Il est conçu comme une brique de base sur laquelle peuvent s'appuyer d'autres protocoles offrant par exemple une diffusion atomique, la résistance aux pannes de sites, ou la gestion de données dupliquées.

I.2.6 Horus

Horus est le projet de recherche mené à l'Université de Cornell qui succède à ISIS décrit à la sous−section I.2.2. Tout comme son prédécesseur, il fournit des outils et interfaces de haut niveau servant à la mise en œuvre de systèmes et d'applications réparties tolérantes aux pannes. Le nouveau projet qui s'inspire aussi du x−Kernel (I.2.5) étudie les techniques de composition de protocoles par empilement et imbrication d'objets apportant chacun une caractéristique au protocole final [30]. Les auteurs comparent la réalisation d'un protocole à un jeu de Légo ©, et fournissent un grand nombre de briques de base, dont les protocoles fournis dans ISIS. La figure suivante donne l'exemple d'une telle composition :

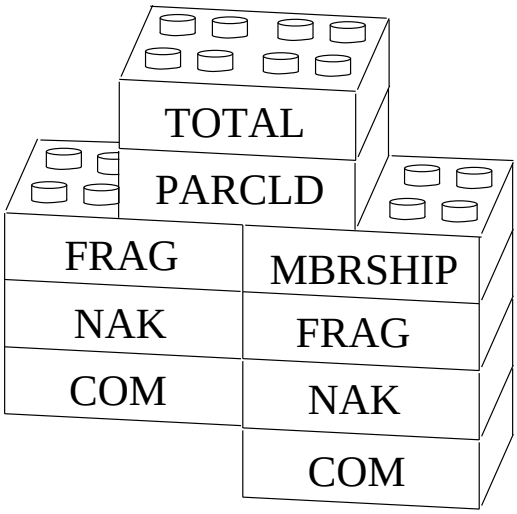


Fig. 1.14 : Composition de protocoles dans Horus

Chaque module implantant un sous−protocole possède des interfaces standardisées, tant vers les protocoles sous−jacents (downcalls) que vers les protocoles qui l'utilisent (upcalls). Le nombre de ces interfaces est important afin de couvrir un maximum de types de protocoles et d'interactions entre modules. Il y a en tout 16 appels descendants couvrant les opérations de construction, de destruction, et la gestion de groupes, de vues et de messages. Les 14 appels montants sont essentiellement liés à la transmission des résultats des opérations asynchrones, ou à la détection d'erreurs.

L'exemple illustré par la figure Fig. 1.14 montre bien les possibilités de combinaisons de protocoles élémentaires pour obtenir un protocole élaboré. Le protocole obtenu permet la diffusion de messages au sein de groupe, supportant un grand nombre de sites, et offrant une diffusion atomique. Les différentes briques utilisées sont les suivantes :

- COM est une interface vers les couches réseaux utilisées. Une grande variété de supports sont disponibles tels que ATM, Udp, Ip ou les messages Mach.
- NAK est un protocole garantissant une délivrance FIFO des messages, même en présence de pertes de messages.
- FRAG fournit la couche de fragmentation/réassemblage nécessaire pour le transport de messages d'une taille supérieure à celle supportée par les couches réseaux.
- MBRSHIP fournit le protocole d'appartenance au groupe et l'atomicité de la délivrance des messages. Il fournit la notion de groupes de processus virtuellement synchrones introduite dans le projet ISIS.
- PARCLD permet d'étendre le mécanisme de groupe pour les très grand réseaux. Cette brique s'appuie sur deux piles de protocoles, l'une destinée au maintien de la cohérence des vues entre tous les membres, et nécessite la brique MBRSHIP, l'autre sert à la transmission des messages au sein du groupe et nécessite juste l'ordre FIFO.
- TOTAL est un des protocoles offrant l'atomicité disponible.

Le projet Horus comporte aussi l'étude de nouveaux protocoles concernant le transport de données multimédia et de protocoles et algorithmes assurant la confidentialité des transferts par le biais de méthodes d'authentification et d'encryptage [29] fonctionnant dans le cadre de communication par groupes.

I.3 Principes de fonctionnement

Les propriétés offertes par les protocoles multi−points peuvent être séparées en trois groupes distincts, à savoir le maintien du service en cas de perte de message, de panne de site et finalement les garanties d'ordonnancement. On peut noter qu'à l'exception de la gestion des pertes, les problèmes posés par le maintien de la qualité de service pour des protocoles de diffusion est sensiblement plus compliqué que pour les protocoles point−à−point. Les mécanismes nécessaires sont aussi beaucoup plus complexes et donc coûteux.

I.3.1 Résistance aux pertes

Le principe général pour pallier les pertes de messages consiste à faire persister toute information transmise tant que la transmission n'a pas été acquittée par les destinataires. Le véritable problème consiste à détecter les pertes. Il est par exemple possible d'estampiller les messages et d'effectuer des vérifications lors de la réception pour détecter d'éventuels messages manquants. Mais cela n'est pas toujours applicable en particulier en l'absence d'une action concertée des divers sites ; on a alors recours à un mécanisme de chien de garde mis en place sur le site émetteur qui attend un signe de la bonne transmission du message en provenance des sites récepteurs. Cette approche suppose toutefois de pouvoir borner raisonnablement le temps de transmission maximal afin d'ajuster le délai, qui, s'il est trop court, risque d'entraîner des retransmissions inutiles, alors qu'un délai trop long pénalise fortement les performances si le taux de perte augmente.

I.3.2 Atomicité et résistance aux pannes de sites

Pour offrir la propriété de fiabilité, deux techniques sont utilisées :

− soit garantir une délivrance simultanée des messages sur tous les sites en vérifiant par exemple que tous les sites ont reçu un message avant d'envoyer le suivant. Ce mécanisme nécessite un protocole à trois phases (envoi, confirmation,validation) très coûteux mais respecte exactement la définition de la fiabilité.

− soit garantir la délivrance du message dans un futur proche même en cas de perte de message. Cela nécessite donc un mécanisme de détection des pertes et de stockage d'une copie du message pour pouvoir le délivrer par la suite si nécessaire. Dans ce cas, l'atomicité peut être garantie sous réserve de la disponibilité des messages si une retransmission est nécessaire, et est donc liée à la résistance du protocole aux pannes de sites.

Ces propriétés sont mises en œuvre par l'estampillage des messages (on rejoint alors les problèmes liés à l'ordonnancement) et le stockage des messages sur plusieurs sites.

I.3.3 Ordonnancement

Parmi les nombreuses contraintes d'ordonnancement possibles, l'atomicité est de loin la plus utilisée. La méthode "classique" pour obtenir un ordre total est l'utilisation d'un site privilégié qui délivre des estampilles associées aux messages à délivrer. On peut lui reprocher son caractère centralisé qui s'oppose aux principes des systèmes répartis ; en particulier l'arrêt de fonctionnement de ce site nécessite tout un mécanisme de récupération. L'approche novatrice et élégante de Melliar−Smith, Moser et Agrawala offre un algorithme complètement réparti mais difficilement utilisable pour un mécanisme d'exécution à cause du délai de délivrance nécessaire.

L'ordre causal, moins coûteux, peut être garanti immédiatement et de manière répartie en utilisant des estampilles vectorielles comme le propose ISIS ou des informations de dépendance comme dans l'algorithme de Melliar−Smith, Moser et Agrawala. Les propriétés sémantiques offertes sont généralement suffisantes pour éviter les incohérences introduites par l'usage de divers sites, mais il est généralement nécessaire d'utiliser d'autres algorithmes et protocoles fournis au dessus de l'ordre causal pour mettre en œuvre des mécanismes de gestion de données répliquées. Par contre l'ordre causal est suffisant pour la plupart des mécanismes liés à l'exécution et au transfert de contrôle entre sites.

Enfin l'ordre FIFO peut être réalisé pour un processus en fournissant une primitive d'envoi synchrone. Le respect de l'ordre FIFO dans le cas où plusieurs processus émettent de manière concurrente sur un même site n'a de sens que si ces processus se synchronisent par le biais d'un autre mécanisme. Dans ce cas, il semble plus efficace que cette coordination se fasse au niveau de l'utilisateur et aucun des protocoles ne propose explicitement cette option.

I.4 Architecture logicielles

L'intégration des communications dans les systèmes d'exploitation tels qu'Unix, qui n'étaient pas au départ conçus pour le partage d'informations entre sites, pose de réels problèmes. Les mécanismes usuels désormais disponibles consistent en un jeu limité de protocoles intégrés au noyau du système d'exploitation et accessibles par le biais d'une couche d'interface standardisée (tels que les "sockets" héritées des systèmes BSD) qui effectue le démultiplexage et l'interface avec les langages de programmation.

I.4.1 Intégration dans le système

Toutefois un grand nombre de reproches à cette solution hâtive mais simple subsistent ; on notera en particulier les points suivants.

- Seuls les protocoles déjà présents dans le noyau sont disponibles via l'interface des "sockets". Si l'on veut utiliser des protocoles différents, il est nécessaire de les implanter au dessus de protocoles existants, par exemple en utilisant le protocole non fiable UDP.
- Ce type de construction peut s'avérer hautement inefficace car le protocole est mis en œuvre indépendamment des capacités réelles du réseau utilisé. De plus cela force

une implantation en mode utilisateur, complètement déconnectée des protocoles existant dans le noyau, d'où des risques de mauvaises synchronisation entre les différentes couches.

C'est pourquoi la plupart des implantations de protocoles de diffusion l'ont été au sein du noyau du système d'exploitation, comme c'est le cas pour Amœba et les premières versions du x−Kernel.

De plus pour pouvoir supporter efficacement les nouveaux réseaux à haut−débit, il peut s'avérer nécessaire d'implanter la quasi−totalité des protocoles dans du matériel spécifique et dans le noyau du système d'exploitation. Par exemple, les couches AAL (ATM Adaptation Layer) qui offrent différentes qualités de service pour les réseaux de type ATM doivent être presque complètement traitées par le matériel, pour pouvoir satisfaire les contraintes temporelles liées au haut débit de telles lignes. Le vendeur de cartes ATM FORE a par exemple fourni une bibliothèque de fonctions semblables à celles des sockets pour pouvoir accéder aux protocoles spécifiques ; cette solution offre donc une interface normalisée, mais ne permet pas un accès direct de l'utilisateur aux interfaces basses du réseau et des cartes.

En ce qui concerne les protocoles multi−points, il est difficile d'offrir les interfaces standard existantes, car le type d'opérations qui peuvent s'appliquer à un groupe est beaucoup plus complexe que celles possibles pour des communications point−à−point. Plusieurs projets de recherche se sont donc attachés à fournir de nouvelles méthodes d'intégration de protocoles, soit au sein d'un système monolithique comme c'était le cas pour la première version du projet x−Kernel, soit dans le cadre, plus complexe, des systèmes architecturés sur des micro−noyaux. Les problèmes de la construction de protocoles au sein de processus externes au noyau, et donc en mode utilisateur, ont fait l'objet d'études récentes [13] [21] où il a été prouvé qu'en limitant les recopies de données, il était possible d'obtenir des performances semblables à celles de protocoles intégrés au noyau. Néanmoins les mécanismes proposés ne sont pas encore présents dans la majorité des systèmes.

I.4.2 Généricité et modularité

Comme nous l'avons vu précédemment, il existe de nombreux protocoles de diffusion offrant des fonctionnalités et des sémantiques différentes. Une première étape consiste à isoler les interfaces "canoniques" nécessaires pour la réalisation de tels services, et celles qu'elle offrent aux couches utilisateur. Les projets tels que le x−Kernel et Horus tendent à prouver que de telles interfaces génériques peuvent être définies au sein même des diverses couches constituant un protocole complexe. Il devient alors possible de construire des environnements dans lesquels les protocoles spécifiques sont obtenus en assemblant des modules fournissant l'ensemble des caractéristiques requises. De plus, en intégrant les interfaces des éléments de base aux interfaces du système d'exploitation sous−jacent, il devient aisé de concevoir des protocoles dont certaines parties "critiques" sont implantées au sein du noyau alors que les couches d'adaptations spécifiques sont chargées dans l'espace des applications qui les utilisent.

I.4.3 Normalisation

Si les protocoles de diffusion sont l'objet d'une étude poussée depuis plus d'une dizaine d'années, il n'existe toujours pas de protocole normalisé et universellement reconnu et utilisé. Certe, certaines réalisation sont aujourd'hui commercialisées avec succès, par exemple la boite à outils Isis, et des efforts de normalisation ont été entrepris (travaux de l'ANSA et protocole IP−multicast), mais aucun ne peut être considéré comme un standard établi.

Pourtant les exemples d'utilisation de protocoles de diffusion ne manquent pas, ce point sera détaillé à la section suivante, mais les besoins sont très divers. En effet, il est possible de construire des protocoles dont les sémantiques sont très différentes, un même protocole pouvant de surcroît accepter un grand nombre de paramètres (nombre de sites, degré de résistance aux pannes, délais, etc...), cette diversité ne simplifie pas la tâche des organismes de normalisation.

I.5 Utilisation

Nous présentons dans cette section les principales utilisations possibles pour les protocoles de diffusion. Trois catégories principales sont distinguées : le maintien de la cohérence de copies d'un même objet, la diffusion d'information, et des mécanismes d'exécution répartie.

I.5.1 Maintien de la cohérence

L'utilisation de protocole de diffusion pour la gestion de données répliquées est une évolution naturelle des théories développées dans le cadre de la gestion de mémoire (en particulier les caches) pour les multiprocesseurs. La gestion en cohérence stricte des copies stockées sur différents sites passe par l'adoption soit d'un algorithme d'invalidation lors des mises à jour, soit une mise à jour simultanée des replicas. Dans les deux cas un algorithme de diffusion fiable, et atomique est nécessaire pour maintenir la cohérence des copies, soit pour transmettre le message d'invalidation, soit pour transférer le nouvel état. Une variante du principe de mise à jour consiste, dans le cas d'objets actifs, à exécuter en parallèle la modification sur les objets. Cela peut procurer un gain appréciable du volume des informations transférées (paramètres de l'appel contre état de l'objet), et améliorer les performances à la condition que les objets aient un comportement déterministe, par exemple l'absence d'entrées/sorties.

I.5.2 Bus de message

Le principe d'un bus de message est de diffuser de l'information à un ensemble d'entités actives réparties sur divers sites. L'exemple des News d'Internet est un bon exemple du type de service que cela représente. Contrairement aux protocoles utilisés pour le maintien de la cohérence de copies, les clients d'un bus de message sont nettement moins exigeants sur la qualité de service en terme d'ordonnancement ou de fiabilité. La figure suivante donne un exemple d'environnement de développement basé sur un bus de message :

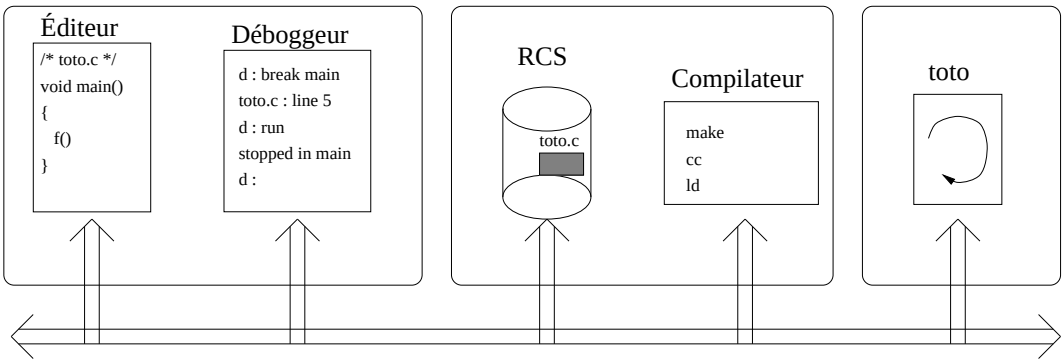


Fig. 1.15 : Environnement de développement utilisant un bus de message

Dans ce cadre, les différents outils communiquent des informations d'intérêt général à l'ensemble des outils. Par exemple lors une sauvegarde d'un fichier via l'éditeur, cet événement est notifié aux autres outils par l'envoi d'un messages spécifique sur le bus ; les applications peuvent alors réagir par exemple en sauvegardant le nouvel état dans la base et en relançant le processus de compilation. Les comportements des applications sont de type événement / réaction ; il est donc important de fournir des délais de réaction et donc de transmission raisonnables. De plus un grand nombre d'événements doivent transiter sur le bus, le débit est donc un facteur important. La description d'un environnement de ce type peut être trouvée dans [32].

I.5.3 Exécution fiable

Enfin les protocoles de diffusion peuvent trouver une application dans l'extension du modèle d'exécution "client−serveur" lorsque le service du serveur est dupliqué sur plusieurs machines. La figure suivante illustre ce mécanisme :

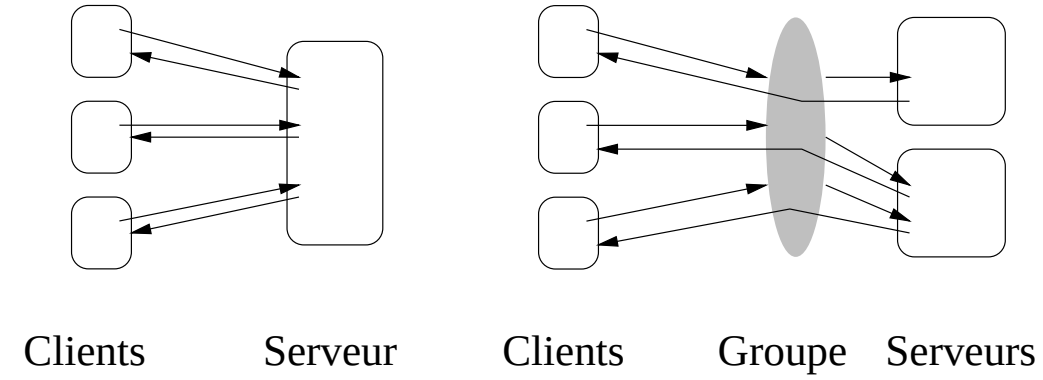


Fig. 1.16 : Extension du modèle client−serveur supportant plusieurs serveurs

L'utilisation d'un mécanisme de groupe permet de remplacer le mécanisme d'appel du serveur qui est explicitement désigné pour servir la requête, par une diffusion vers un

ensemble de serveurs répliqués et désignés de manière uniforme par le groupe. Cette approche permet plusieurs améliorations :

- fournir un service fiable : si un des serveurs tombe en panne, les autres continuent à servir les clients.
- améliorer les performances : en répartissant la charge entre les serveurs valides, plusieurs machines sont plus à même de répondre à une charge croissante. Le problème est de fournir des critères de répartition des requêtes entre les serveurs.
- augmenter l'adaptabilité et faciliter la reconfiguration du service. L'adjonction de nouvelles machines répondant aux requêtes est facilitée par l'indirection que constitue le groupe. Cela permet de faire évoluer dynamiquement le service.

Ce type d'utilisation est particulièrement sensible aux propriétés de résistance aux pannes de sites du protocole de diffusion utilisé et aux débits maximum autorisés.

On peut donc conclure que les protocoles de diffusion sont un des éléments de base permettant de construire des systèmes répartis. Ils ont suscité beaucoup d'analyses théoriques, qui ont dégagé les principales caractéristiques en termes d'ordonnancement et de fiabilité. Il en résulte une grande diversité de protocoles et de sémantiques différentes adaptées à diverses utilisations. Les environnements permettant de construire des protocoles répondant aux besoins spécifiques des applications restent un domaine de recherche actif.

La suite de ce document présente la conception, l'implantation et les résultats d'un protocole de diffusion pour réseaux locaux. La réalisation de ce protocole a été motivée par l'absence de mécanisme de communication par groupe dans l'environnement du micro−noyau Mach 3.0 qui etait la cible du projet Guide2 dans lequel s'est effectué ce travail. Cela permettait aussi d'explorer en pratique le principe des groupes opaques que nous détaillerons au chapitre suivant. Les premières expériences de communication utilisant les primitives de Mach 3.0 s'étant révélées décevantes en termes de performance, nous avons étudié l'impact de l'implantation du protocole en mode utilisateur et expérimenté une implantation du protocole au sein du noyau Mach. L'arrêt des développements sur Mach et le portage sur divers systèmes Unix a permis de vérifier la portabilité du protocole et d'analyser son comportment dans un environnement standard.

Chapitre II

Conception d'un protocole de diffusion

Ce chapitre présente les principaux choix de conception retenus pour l'implantation du protocole de diffusion. Le protocole choisi est d'abord présenté, puis critiqué. Le principe des groupes opaques retenu est exposé, en particulier en ce qui concerne les implications théoriques et pratiques qu'il entraîne. Enfin diverses optimisations sont proposées et les gains théoriques qu'elles apportent sont évalués.

II.1 Choix du protocole

Cette section motive le choix du protocole de diffusion retenu pour l'implantation. Les critères requis sont exposés, le protocole choisi est alors détaillé, puis critiqué pour motiver les modifications qui lui seront apportées.

II.1.1 Qualités requises

Le choix du protocole est naturellement dépendant de l'usage que l'on compte en faire, à savoir servir de mécanisme de base pour réaliser des systèmes distribués ou des applications coopératives. L'utilisation d'un ensemble de stations connectées à un réseau local pour les expérimentations est aussi un facteur déterminant. Le protocole choisi devra se conformer aux besoins énoncés ci−dessous :

- fournir les propriétés sémantiques requises pour la gestion d'objets répliqués. Le protocole doit offrir un mécanisme de diffusion fiable, garantissant un ordre total.
- pouvoir résister aux pannes de sites.
- fournir la notion de groupe.
- être efficace, en particulier le temps de latence induit par les communications est un critère très important lors de la mise en œuvre d'un système réparti.

Le dernier point nécessite de nous intéresser aux divers critères de performances que l'on peut distinguer pour un protocole de diffusion. La liste ci−dessous indique les principaux critères :

- le taux de transfert atteint distingue la capacité à transporter de grands volumes d'information. Ce type de critère s'applique essentiellement aux protocoles de transport de données tels que FTP, et peut être nuancé par des critères de qualité de service, par exemple dans le cadre de transport de canaux audio ou vidéo.

- le nombre de messages transmis par unité de temps. Ce critère qui permet d'analyser la résistance à la charge du protocole dans le cadre de communications impliquant un grand nombre d'échanges de données, celles−ci étant alors de petites tailles. Cela correspond par exemple aux échanges d'informations liés au transfert d'exécution dans les systèmes répartis.
- le temps de latence, c'est à dire le temps absolu séparant l'envoi des données sur le site émetteur de leur réception sur les sites récepteurs. Ce critère est très important en pratique car il est le facteur limitant les performances lorsque le réseau n'est pas saturé et que seule importe la transmission de l'information ou l'enchaînement de l'exécution entre les divers sites composant un système réparti.
- la charge processeur nécessaire au fonctionnement du protocole, en particulier lorsque la charge augmente. Cela dépend fortement du type d'interface offerte par l'adaptateur, la qualité des pilotes de périphériques, et de l'organisation logicielle du protocole.

La nécessité de fournir un protocole garantissant de faibles temps de latence nous a conduit à éliminer les protocoles garantissant le bon ordonnancement par comparaison d'estampilles entre messages successifs, tels que Total présenté au I.2.3. En effet la délivrance d'un message est alors soumise à la réception d'un ensemble de messages en provenance de divers sites avant de pouvoir être délivré localement à l'utilisateur. Ce type de protocole permet d'obtenir un bon débit car le nombre de messages échangés est minimal, mais au détriment d'un temps de latence qui peut être élevé.

Le confinement au sein d'un même réseau local des sites mis en jeu offre la possibilité d'utiliser le mécanisme de diffusion physique d'Ethernet. Notre sélection s'est donc orientée vers les protocoles pouvant faire usage de cette possibilité pour réduire la bande passante utilisée et limiter le nombre de messages échangés.

II.1.2 Protocole choisi

Parmi le grand nombre de protocoles de diffusion existants, seuls les protocoles de Chang et Maxemchuck et son successeur, le protocole d'Amœba répondaient aux critères retenus. Notre sélection s'est portée vers le second, celui−ci étant une version améliorée du premier, avec toutefois le léger désavantage d'être centralisé. Son principal avantage est d'être d'un principe très simple :

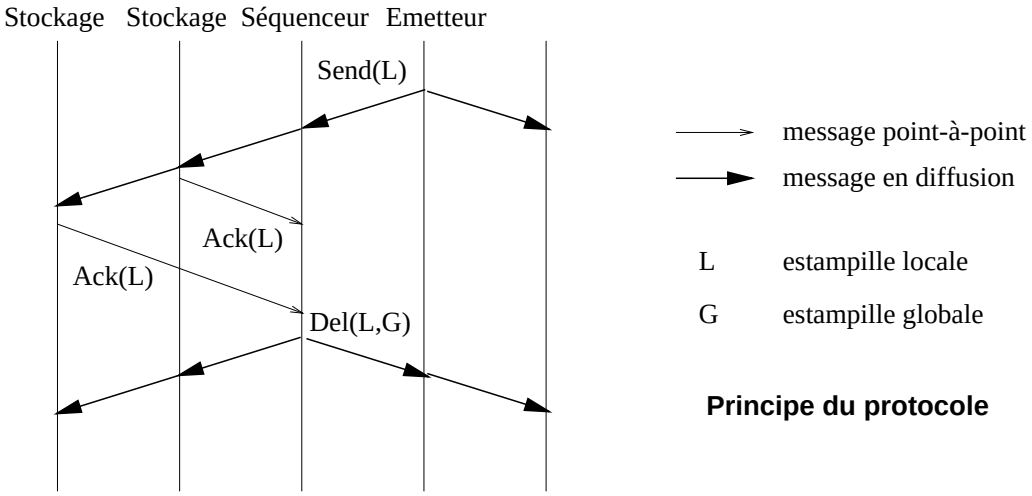


Fig. 2.1 : Le principe du protocole d'Amœba

La résistance du protocole aux pertes de messages, même dans les cas de pannes de site, est assurée par la présence de K sites de stockage dont le rôle est de conserver une copie de tous les messages distribués au sein du groupe. L'ordre total est fourni par la présence d'un site séquenceur qui est le seul habilité à décider de la délivrance d'un message, et qui fournit l'estampille globale associée à chaque message diffusé. Le séquenceur est aussi un site de stockage. Lorsqu'un site émet un message à destination du groupe, il en diffuse le contenu au sein d'un message de type SEND, et lui associe une estampille locale unique L. La valeur de L peut être formée par un couple fait d'un compteur incrémenté à chaque émission et d'un identificateur du site émetteur. Lors de la réception du message, chaque site de stockage le notifie au site séquenceur en lui envoyant un message d'acquittement ACK en point−à−point, en précisant l'estampille locale du message reçu. Enfin, lorsque le séquenceur a reçu tous les acquittements du message en provenance des sites de stockage, il émet un message de délivrance formé d'un couple <L,G> où G est l'estampille globale unique associée au message L, obtenue en incrémentant un compteur disponible uniquement sur le site séquenceur. De cette manière la succession des estampilles globales délivrées forme un ordre total, ce qui est le but recherché, mais de plus ces estampilles balayent la suite des entiers en s'incrémentant de 1 en 1, ce qui permet de détecter la perte de messages sur les sites clients. En effet, un saut dans les valeurs obtenues indique avec certitude que des messages ont été perdus. La gestion des pannes de sites est expliquée au II.3 et celle des pertes de messages est détaillée à la section II.4.

II.1.3 Critique du protocole existant

Un des principaux arguments à l'encontre de ce protocole est l'utilisation d'un séquenceur statique. En effet, on peut craindre les problèmes suivants :

- la charge imposée au séquenceur risque d'être un facteur limitant des performances du protocole. On peut décomposer son impact en deux composantes principales, à

savoir la charge processeur nécessaire pour gérer le protocole et le supplément d'entrées/sorties réseau dû à la réception des acquittements.

- l'utilisation d'un site central pour un protocole réparti est toujours un problème car il faut parer à l'éventualité d'une panne de celui−ci.
- la rotation du séquenceur est nécessaire dans l'algorithme de Chang et Maxemchuk pour éviter le problème du "buffer infini". Dans ce protocole le séquenceur est statique ce qui nécessite une gestion appropriée des tampons. Celle retenue est présentée au II.2.4.

Le protocole de resynchronisation nécessaire lors d'une panne du séquenceur est détaillé au II.3.3. Sa mise en œuvre est certes complexe et bloque toute communication au sein du groupe, mais cela doit être mis en balance avec la rareté des pannes et le gain de temps apporté par l'utilisation d'un site unique pour l'estampillage.

La surcharge imposée au séquenceur est principalement liée au nombre de messages supplémentaires que celui−ci doit traiter : les acquittements en provenance des sites de stockage. On peut remarquer que leur taille est très faible car ils ne contiennent que des informations de contrôle ne nécessitant que quelques octets. La surcharge imposée est donc essentiellement liée au temps de traitement d'un paquet. Or on constate que sur le type de machines que nous considérons, telles que des stations de travail ou de bureautique, la puissance des processeurs évolue bien plus rapidement que les performances en entrées/sorties, la surcharge relative a donc tendance à diminuer. Le protocole tente en outre de placer le séquenceur sur une machine rapide lors des éventuels changements de configuration. L'analyse des temps de traitement, et l'impact de divers paramètres, dont le choix du séquenceur sont étudiés en détail au chapitre V.

II.2 Groupes opaques

Un des problèmes théoriques majeurs liés aux protocoles de diffusion est le suivant :

Pour faire de la diffusion il est généralement nécessaire d'avoir une liste de sites auxquels doivent parvenir les messages. Pour maintenir cette liste de manière cohérente il faut que les changements soient répercutés de manière atomique vis−à−vis du protocole

En effet, des propriétés comme la causalité ou l'atomicité nécessitent de savoir de manière absolue quelle est la liste des sites participant à un groupe. L'exemple le plus frappant est l'introduction du protocole GBCAST (I.2.2.3) dans les premières versions d'ISIS [3]. Son but est de fournir une primitive atomique par rapport aux protocoles ABCAST et CBCAST. Son emploi est nécessaire pour maintenir une vue cohérente des membres d'un groupe entre les divers participants. Ce problème est à la base d'un gros travail théorique concernant les mécanismes et algorithmes permettant de détecter la mort des sites [7]. De plus, comme l'implantation du protocole utilise ce type d'informations pour son fonctionnement, en particulier le respect de la sémantique associée à la délivrance de messages, il doit être mis en place indépendamment des couches supérieures du protocole. Les réalisations

récentes des bibliothèques de protocoles l'intègrent souvent comme un module de base mis en œuvre indépendamment des mécanismes de délivrance de messages.

II.2.1 Présentation

Les groupes opaques sont définis par le fait que le protocole n'utilise pas pour sa bonne marche une liste comportant l'ensemble des membres d'un groupe, même fractionnée sur un ensemble de sites. Ce type de groupe nécessite donc forcément l'emploi d'un mécanisme de diffusion indépendant du protocole, et fourni directement par les couches d'accès au réseau. La diffusion physique peut par exemple être employée sur tous les réseaux locaux contenant des sites faisant partie du groupe. Les principaux avantages de la réalisation du protocole sur un groupe opaque, quand c'est possible, sont de deux ordres:

- une simplification forcée du protocole qui ne doit, ni ne peut manipuler la liste complète des sites. Cela force la séparation entre ce qui relève du mécanisme de diffusion réalisé par les couches réseau et le principe de fonctionnement du protocole. Cela rejoint la tendance consistant à isoler le mécanisme de détection des pannes de sites du protocole lui−même.
- une amélioration des performances et particulièrement des temps de latence lorsque le nombre de sites augmente ; en effet, le mécanisme de diffusion fourni par les couches inférieures a de bonnes chances d'être plus efficace que la réalisation du même service depuis le protocole. De plus, isoler les mécanismes de connexion et de déconnexion du reste du protocole peut limiter l'impact de ces opérations en confinant leur effet à un petit nombre de machines.

Le principe théorique des groupes opaques a des conséquences pratiques importantes comme nous allons le voir par la suite.

II.2.2 Propriétés

Le concept de groupe opaque se justifie bien dans le cadre du protocole retenu. En effet, l'essentiel des opérations requises par un membre qui n'est pas séquenceur ou site de stockage consiste à émettre ou à recevoir des paquets en diffusion. Le fonctionnement normal du protocole sur ces sites ne requiert pas d'échange de messages point−à−point. Vis−à−vis de la réception des messages adressés à un groupe, seule l'appartenance au groupe du site local intervient. Tous les sites reçoivent tous les messages si ceux−ci sont émis en diffusion physique sur le réseau. C'est donc au site local de savoir, lorsqu'il reçoit un message adressé à un groupe, s'il doit le garder ou non. Cela peut être réalisé simplement à l'aide d'une variable booléenne pour chaque groupe, indiquant l'appartenance ou non au groupe.

Nous désignons par la suite un tel site par l'adjectif "anonyme", contrairement aux sites de stockage, dont le séquenceur, qui forment le "quorum" chargé d'assurer la sémantique des opérations liées au groupe. La figure ci−après illustre cette répartition :

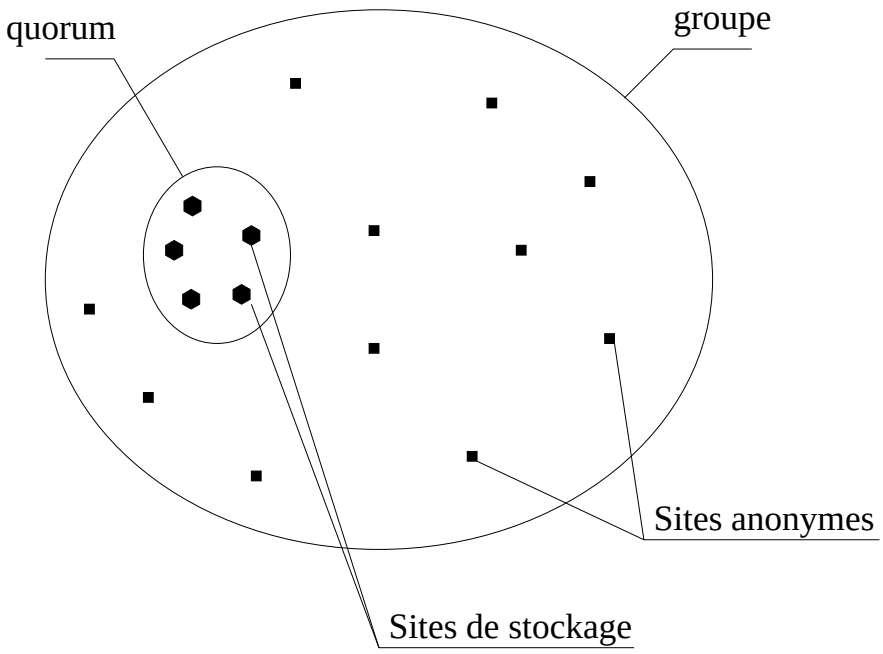


Fig. 2.2 : Le groupe formé du quorum et d'un ensemble de sites anonymes

Le coût associé aux opérations d'abonnement et de résiliation pour les sites anonymes est très faible :

- La procédure de connexion requiert la connaissance de l'association entre le schéma de désignation externe du groupe, typiquement une chaîne de caractères, et l'identificateur interne utilisé par exemple pour le démultiplexage des messages entrants. La connexion d'un site anonyme au groupe nécessite donc un échange de messages entre le nouveau client et un des sites du quorum.
- La déconnexion d'un site anonyme, est par contre une opération locale atomique qui consiste à retirer le groupe de la liste des groupes locaux, et à détruire les données associées.

Ces opérations ne perturbent en rien le fonctionnement normal du groupe, en particulier le trafic courant n'est pas bloqué. C'est le résultat de l'absence d'état global associé à la composition du groupe.

L'admission d'un nouveau site au sein du quorum est elle aussi non bloquante, elle se produit lorsqu'un site demande à se connecter au groupe, et que le quorum ne comporte pas un nombre suffisant de sites assurant la duplication. Dans ce cas le séquenceur diffuse un message comportant la nouvelle composition du quorum, après y avoir ajouté le nouveau site, celui−ci prend alors part au mécanisme de stockage et d'acquittement des messages à partir de la réception de ce message. La perte du message émis par le séquenceur provoque éventuellement une phase de resynchronisation du groupe, seul cas où l'opération devient bloquante. Par contre le départ, annoncé ou détecté après un délai de garde, d'un des sites du quorum entraîne une phase de resynchronisation qui est décrite en détail au II.3.2.

Le principe des groupes opaques, quand il est applicable, permet de réduire significativement la complexité des protocoles, ce qui se traduit en pratique dans notre cas par une

réduction des coûts de connexion et de déconnexion, ce qui est important si on utilise un groupe comme un bus de messages.

II.2.3 Inconvénients

On peut noter deux inconvénients majeurs liés à l'utilisation de groupes opaques:

- il faut modifier le protocole pour éliminer complètement toute recherche d'un état global à tous les sites du groupe. Dans notre cas nous avons dû modifier l'algorithme d'élimination des messages conservés sur les sites de stockage (ce point est détaillé à la sous−section II.2.4 suivant)e.
- les applications utilisant les mécanismes de groupe peuvent avoir besoin d'une liste des membres que le protocole n'est pas capable de leur fournir. Pour pallier cette lacune, il est possible de rajouter une couche de protocole supplémentaire gérant la liste des membres. On peut profiter pour l'implanter, des propriétés du protocole de groupe ; dans notre cas, l'ordre total et l'atomicité fournissent directement la cohérence de vue. Seule la détection de la perte d'un site lors de pannes est difficile car il faut recourir au mécanisme habituel de chien de garde.

Les mesures effectuées et détaillées aux chapitre V nous ont révélé un autre problème pratique lié aux groupes opaques, et concernant l'impossibilité de fournir un contrôle de flux pour un protocole géré en groupe opaque.

II.2.4 Modifications nécessaires

Pour prévenir le problème du "buffer infini" sur les sites de stockage, le protocole de Chang et Maxemchuk s'appuie sur la rotation du séquenceur sur l'anneau virtuel des sites (mécanisme détaillé au I.2.1). La modification de cet algorithme lors de l'élaboration du protocole d'Amœba a eu pour conséquence de bloquer le séquenceur sur un site. Le mécanisme retenu par Kaashoek et Tanenbaum pour éliminer les messages dupliqués sur les sites de stockage consiste à collecter régulièrement les estampilles globales de tous les sites du groupe, et à éliminer tous les messages antérieurs à la valeur minimale trouvée.

Malheureusement cette technique est, par définition, inapplicable pour un groupe opaque. Il faut donc définir une nouvelle politique de gestion des historiques sur les sites de stockage. La solution retenue dans notre implantation est de garder un nombre de messages borné, avec une élimination successive des anciens messages au fur et à mesure de l'attribution des nouvelles estampilles globales. Ce principe a l'avantage d'être simple, car c'est une opération purement locale, et qui plus est l'élimination d'un message périmé coïncide avec le stockage d'un nouveau lors de la délivrance d'une estampille globale. En pratique, cela signifie que si un site a été déconnecté du réseau pour une courte période, par exemple lors de l'engorgement du système, il pourra recharger tous les messages manquants depuis les sites de stockage. Mais si le nombre de messages perdus est supérieur à la taille de l'historique, le site devra quitter le groupe. La taille de l'historique est donc un paramètre important de la configuration du groupe. Par exemple si celui−ci est utilisé pour maintenir un faible nombre d'objets dupliqués, il peut être avantageux de limiter la taille de l'historique, car recharger l'ensemble de la base est moins coûteux que de reconstruire l'état courant depuis une très

ancienne version. Par contre si le groupe sert à maintenir des données vitales, par exemple des informations de configuration d'un système réparti, il est important d'avoir un historique de grande taille pour éviter de redémarrer le système lors d'une déconnexion temporaire.

D'autres politiques de gestion des historiques peuvent être suggérées, sachant qu'aucune ne doit reposer sur la collecte d'informations sur l'ensemble des sites :

- baser l'élimination sur un critère temporel, tout message étant éliminé de l'historique après expiration d'un délai de garde.
- limiter le volume de l'historique, particulièrement intéressant si les messages échangés sont de petite taille, par exemple des informations de contrôle sur un bus de message.
- utiliser un algorithme adaptatif basé sur les critères précédents, où la taille de l'historique peut par exemple fluctuer en fonction du débit au sein du groupe.

Il semble important pour des raisons de simplicité et d'efficacité que la politique adoptée ne nécessite pas d'échange de message entre les membres du quorum, mais soit basée sur une évaluation locale de l'état du protocole.

II.3 Gestion des pannes de sites

La résistance du protocole aux pannes de sites est un des points importants d'un point de vue tant théorique, que pratique. Nous définissons dans un premier temps ce qui caractérise une panne de site, puis le mécanisme de reprise après panne est détaillé, enfin des mécanismes d'optimisation sont présentés.

II.3.1 Définition

La technologie des réseaux actuels ne permet de donner une définition de la panne d'un site que par la détection de sa non−activité, telle qu'observée depuis les autres sites. Même si des techniques de détection systématique de l'arrêt ou de la déconnexion d'un site existent, elles nécessitent l'utilisation de matériels non standards, et sont donc destinées à un petit nombre d'applications critiques. Les installations usuelles, telles qu'Ethernet, ne permettent pas de distinguer l'arrêt physique d'un site (suite à une erreur matérielle, ou à une coupure de courant), d'une déconnexion entre l'adaptateur et le médium ou à une absence de traitement suite à un dysfonctionnement des couches réseaux mises en œuvre.

Pour les sites anonymes, comme on l'a vu précédemment, un site se considère comme "en panne" s'il lui manque trop de messages pour pouvoir les recharger depuis un des sites de stockage. C'est l'équivalent de la notion d'hypothèse de panne consécutive à une absence de réponse prolongée de la part d'un site, méthode utilisée dans le cadre des groupes explicites (non opaques). Dans ce cas le site se déconnecte du groupe en avertissant les client locaux de l'arrêt des communications, charge à ceux−ci de demander une reconnexion si nécessaire.

II.3.2 Détection de perte et de pannes

Par contre les membres du quorum effectuent une surveillance mutuelle basée sur des échanges de messages à intervalles réguliers. On se retrouve dans le cadre classique d'une détection de panne "active". Le mécanisme est basé sur une valeur délimitant le temps maximum qui peut être nécessaire entre la transmission et la réception d'un message. Cette valeur conditionne toutes les échelles de temps prises en compte dans le protocole pour détecter des pertes de messages. Pour des stations locales connectées à un même segment Ethernet, on peut raisonnablement supposer que ce délai n'est pas supérieur en pratique à une dizaine de millisecondes. Mais cela suppose une prise en compte immédiate de chaque paquet sur le site destinataire, hypothèse qui ne peut être justifiée en pratique que si le système d'exploitation fournit des garanties concernant les divers temps de traitement, et donne la main au protocole immédiatement. Ce type de contrainte forte ne peut être justifiée que dans le cadre de systèmes temps−réel, en déclarant le protocole comme une tâche prioritaire. Les machines cibles de notre protocole étant des stations de travail standard, ce type de services n'est pas offert par le système, aussi nous avons choisi des valeurs limites très supérieures. La valeur de base qui est un paramètre Δ fixé à la compilation du protocole a été fixée à 1 seconde lors des tests. Nous avons choisi volontairement une valeur de deux ordres de grandeur supérieure à la valeur minimale possible pour s'assurer que le bon fonctionnement du protocole n'est pas lié à l'hypothèse du modèle synchrone (même si la détection des erreurs l'utilise). Les délais de garde suivant sont utilisés dans le protocole :

- PROTOCOL_MAX_DELIVER_DELAY, fixé à $3 * \Delta$ correspond au délai à partir duquel un site considère qu'un message envoyé au groupe a été perdu et doit être envoyé à nouveau. La valeur choisie correspond au délai maximum pour l'émission (SEND), puis les acquittements qui s'effectuent simultanément, et enfin la délivrance (DELIVER).
- PROTOCOL_GROUP_SYNCHRO_DELAY, vaut lui aussi $3 * \Delta$ et est l'unité de temps utilisée lors des phases de resynchronisation, qui sont décrites à la sous−section suivante II.3.3.
- PROTOCOL_GROUP_FAILURE_DELAY, fixé à $8 * \Delta$ est l'intervalle de temps maximum admis par les sites de stockage lorsqu'une estampille globale est attendue. Ce cas se produit si un des sites de stockages ou le séquenceur est en panne (matérielle, logicielle ou surcharge), ce qui bloque la délivrance des messages au sein du groupe. Si un des sites du quorum détecte ce problème il déclenche une phase de synchronisation.
- PROTOCOL_ANON_GROUP_ABORT_DELAY, vaut $50 * \Delta$ constitue le délai maximum qu'un site anonyme accepte pour la délivrance d'un de ses messages au sein du groupe. Ce type d'erreur est exceptionnelle et est destinée à la détection de la perte complète du groupe, par exemple si les transmissions sont bloquées, ou si tout les sites de stockages sont morts ce qui entraine de fait la fin du groupe.

Nous avons choisi de laisser aux seuls membres du quorum la possibilité d'initier une phase de resynchronisation. En effet ce processus est coûteux et interrompt les communications au sein du groupe pendant son déroulement : il ne doit être utilisé que si les conditions

l'imposent absolument. On veut donc éviter la situation où un site anonyme surchargé attribue à tort les délais et les pertes qu'il détecte à un mauvais fonctionnement du réseau ou du groupe, et risque de déclencher une phase de resynchronisation abusive, qui n'améliorerait pas la situation.

II.3.3 La resynchronisation

Le mécanisme de resynchronisation est lancé dès que la perte d'un site du quorum semble avoir été détectée. Elle a pour but de reformer une vue stable de celui−ci, de mettre à jour les sites de stockage retardataires (c'est−à−dire ayant une estampille globale inférieure à l'estampille maximum trouvée au sein du quorum). C'est un protocole nécessitant un consensus, on peut donc affirmer qu'il est complexe. La figure suivante illustre l'algorithme dans le cas où l'un des sites du quorum tombe en panne :

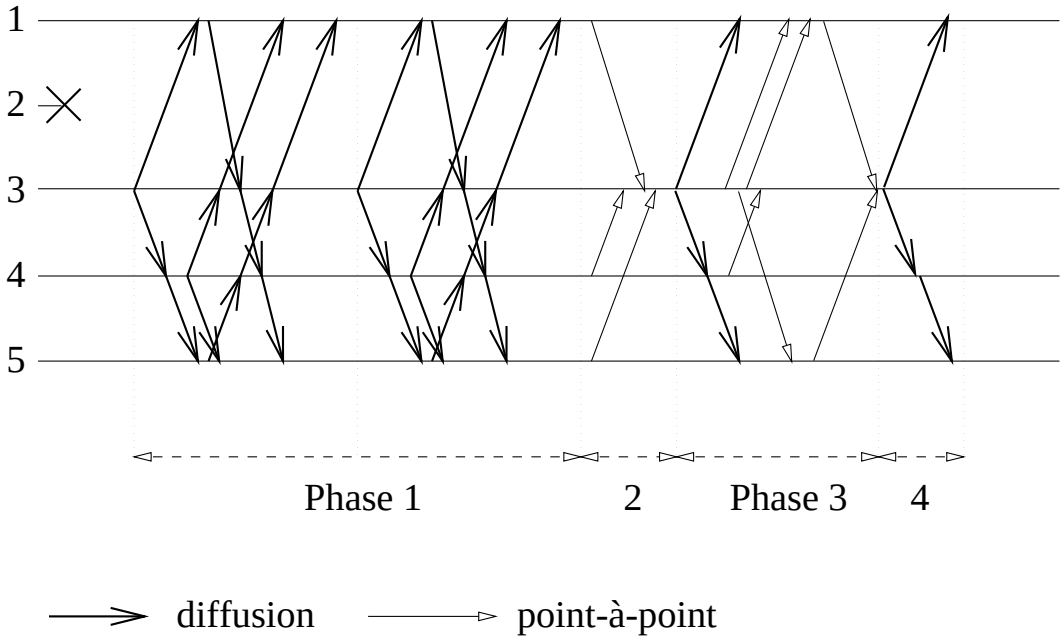


Fig. 2.3 : Protocole de resynchronisation, un exemple

Le protocole de resynchronisation commence dès que l'un des sites du quorum détecte le problème. Il envoie alors un message en diffusion de type START_SYNCHRO, qui déclenche le processus. Dans une première phase (Phase 1), chaque site envoie en diffusion des messages SYNCHRO_1 et collecte ceux émis par les survivants du quorum précédent. Ce message comporte la puissance nominale de la machine, corrigée par la charge à l'instant donné, et le niveau d'estampille globale courante sur le site émetteur. Cette phase se termine sur un site soit après réception d'un message de tous les autres sites du quorum précédent, soit après plusieurs délais de garde chacun suivi d'une phase d'échange de messages. La première méthode permet d'effectuer cette phase rapidement dans le cas où aucun site n'a disparu.

Une fois la première phase terminée, chaque site choisit parmi sa liste de survivants le candidat au poste de séquenceur en se basant sur les informations collectées. Le site le plus rapide parmi les sites de plus haut niveau d'estampille globale est retenu. Dans l'exemple fourni, tous les sites votent pour le site 3 (Phase 2). On remarquera qu'il est possible, quoique improbable, qu'un des sites anonymes possède une estampille globale de niveau supérieur à celle de tous les sites survivants du quorum. Ce cas difficile peut se produire si le séquenceur tombe en panne juste après l'émission d'une estampille globale, et qu'aucun des autres sites du quorum ne reçoive ce message. Les sites anonymes suivent les informations diffusées lors de la première phase et doivent dans ce cas réémettre un message START_SYNCHRO, comportant les dernière associations entre estampilles locales et globale. Ce mécanisme est similaire dans son principe à ceux utilisés pour minimiser l'impact des pertes de messages décrit à la sous−section suivante II.4.2.1. Cela a pour effet d'éviter la perte de l'estampille globale en remettant à jour les sites du quorum et de redémarrer la phase de resynchronisation.

Si un consensus des sites survivants est obtenu, le site ayant obtenu tous les votes se déclare séquenceur en envoyant un message de type SYNCHRO_3. Cette étape (Phase 3) est aussi utilisée pour mettre les sites de stockage à jour, le séquenceur envoie les messages et estampilles manquants aux sites qui en ont besoin. Dans notre exemple le site 5 est en retard de 1 message, le site 1 nécessite 2 mises à jour, par contre le site 4 est à jour.

La dernière phase du protocole (Phase 4) consiste en l'envoi message de type SYNCHRO_4 de chacun des sites de stockage vers le séquenceur dès qu'il sont à jour, dans notre exemple le site 4 envoie ce message dès la réception du message SYNCHRO_3 alors que les sites 1 et 5 ne le font qu'après réception des mises à jour. Une fois tous les messages en provenance des sites de stockage collectés, le séquenceur diffuse un message de type SYNCHRO_5 qui finit la phase de resynchronisation.

La figure page suivante donne une description formelle de cette procédure sous la forme d'un automate de Pétri. Pour des raisons de lisibilité, certaines transitions ont été omises, en particulier le redémarrage de la procédure après l'expiration du délai de garde dans les phases 3 et 4. De plus pendant toute la durée de la resynchronisation, les sites de stockage continuent à répondre aux demandes de mises à jour de la part des sites anonymes, cette transition décrite dans l'état 1 n'a pas été répliquée partout. De même, la réception d'un message START_SYNCHRO initie une transition vers l'état 1, et ce depuis tout état du graphe.

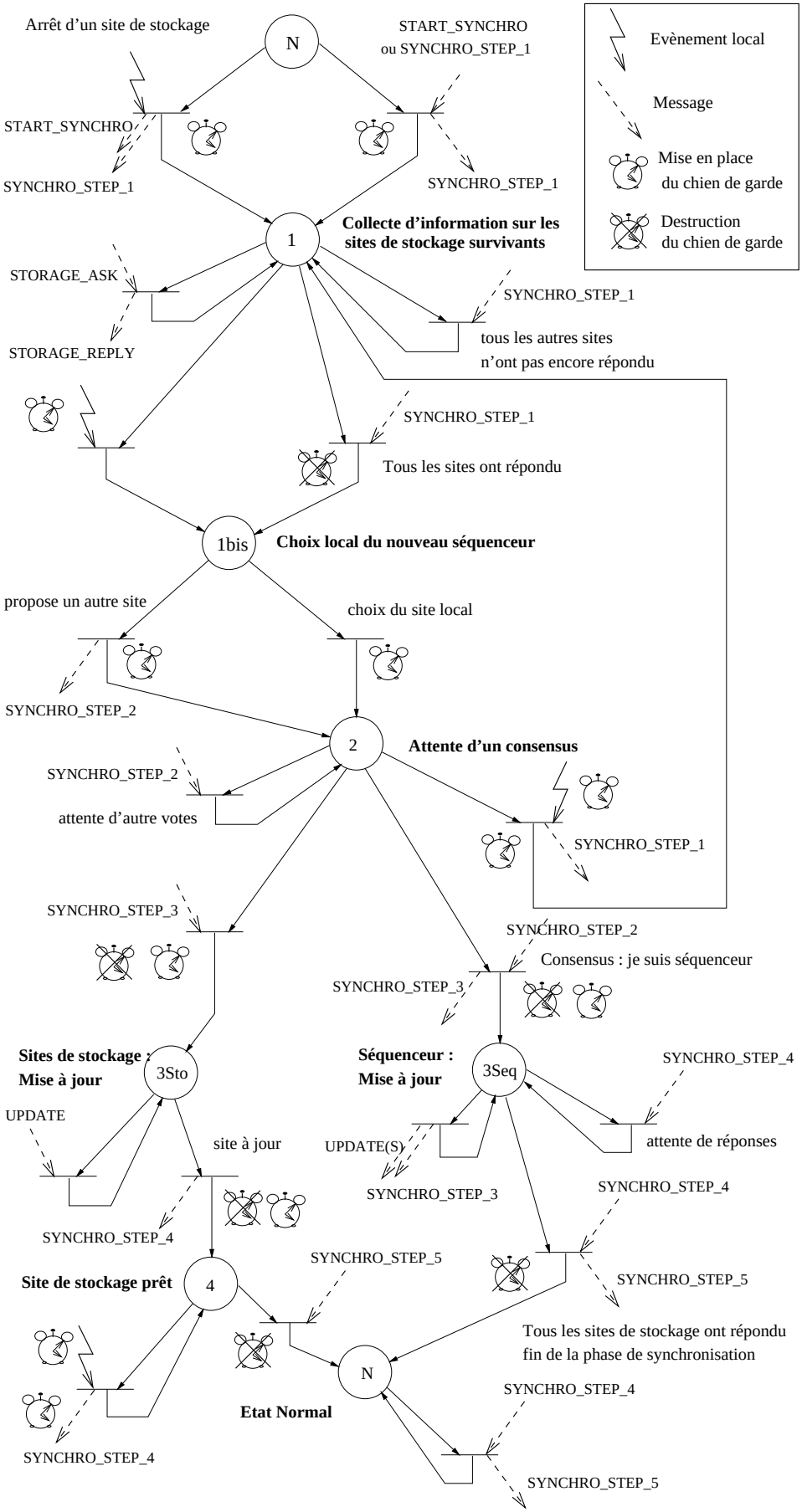


Fig. 2.4 : Automate a jeton associé à la phase de resynchronisation pour le quorum

Les sites anonymes ne prennent pas part à la la resynchronisation sauf dans le cas bien précis où une estampille globale a été perdue par les sites du quorum. Toutefois ils entrent dans un mode de fonctionnement restreint pendant cette période, qui débute dès la réception d'un message de type START_SYNCHRO ou SYNCHRO_x où x est inférieur a 5. Ils quittent cette phase à la réception d'un message de type SYNCHRO_5 ou DELIVER indiquant la reprise de la marche normale du groupe.

Durant la phase de resynchronisation, les sites anonymes n'émettent pas, toutes les requêtes en provenance des clients sont bloquées et ne seront effectivement transmises qu'après le redémarrage du groupe. Pour éviter la saturation au moment de la reprise, un mécanisme d'étalement dans le temps des transmissions de messages accumulés est nécessaire. Les premières versions du protocole n'en étaient pas pourvues et la charge subite résultant de la transmission simultanée de tous les messages en stock provoquait souvent une surcharge du quorum, et donc une nouvelle phase de resynchronisation.

Pour résumer, le mécanisme de resynchronisation est une tâche lourde, mettant en œuvre un protocole à quatre phases basé sur un consensus, mais est heureusement limité aux seuls membres du quorum. Durant cette période, qui peut atteindre plusieurs secondes, toute activité normale de délivrance des messages au sein du groupe est stoppée, il est donc impératif de ne la déclencher qu'à bon escient.

II.3.4 Optimisation

Une des optimisations utilisées pour limiter le coût du processus de resynchronisation est de s'assurer de la réactivité du protocole dans le cas où aucun des sites du quorum n'a disparu. On peut vérifier que dans ce cas le déroulement de l'ensemble des opérations nécessaires ne s'appuie jamais sur l'expiration d'un délai de garde, ce qui minimise la durée totale de la phase de resynchronisation, et donc son impact global sur les communications au sein du groupe.

De manière plus générale, l'ensemble des mécanismes de détection et de mise à jour des groupes bénéficierait d'une gestion de détection indépendante des protocoles eux−mêmes. C'est ce principe qui est mis en œuvre dans les mécanismes de détecteurs de pannes ("failure suspector") qu'utilisent les protocoles récents tels qu'Horus. Cela permet de factoriser cette fonction pour tous les protocoles et tous les groupes présents sur une même machine. Mais le coût lié au maintien de la cohérence des vues demeure, en particulier lorsque le nombre de sites croit et si les protocoles nécessitent une cohérence de vue absolue. Le principe des groupes opaques permet de limiter cette cohérence à un faible nombre de sites, mais maintenir celle−ci en cas de perte d'un site reste coûteux.

II.4 Optimisations en cas de pertes

Deux types d'erreurs sont susceptibles de se produire lors de l'exécution de protocoles de communications : la panne de site ou du médium de communication qui, comme nous l'avons vu, demandent un traitement coûteux mais heureusement exceptionnel, et les pertes ou la déformation d'informations qui sont relativement fréquentes. Tous les adaptateurs de

réseaux effectuent une vérification sur les informations reçues basée sur de la redondance de données, et sont donc à même d'éliminer les informations déformées. Cela augmente d'autant plus la probabilité de perte de blocs de données perceptible au niveau des couches de communications. Cette section explique les techniques mises en œuvre pour limiter l'impact des pertes de paquets ; nous détaillons d'abord les différentes pertes de messages possibles et leur impact, puis les différentes optimisations sont présentées ainsi que les résultats théoriques qu'elles entraînent.

II.4.1 Impact des pertes de messages

L'impact de la perte d'un message, et donc les mécanismes mis en œuvre pour rétablir une situation normale dépend énormément du type de message. La figure suivante tente de regrouper les messages par catégories, fonctions de l'importance de la perte et des actions à effectuer :

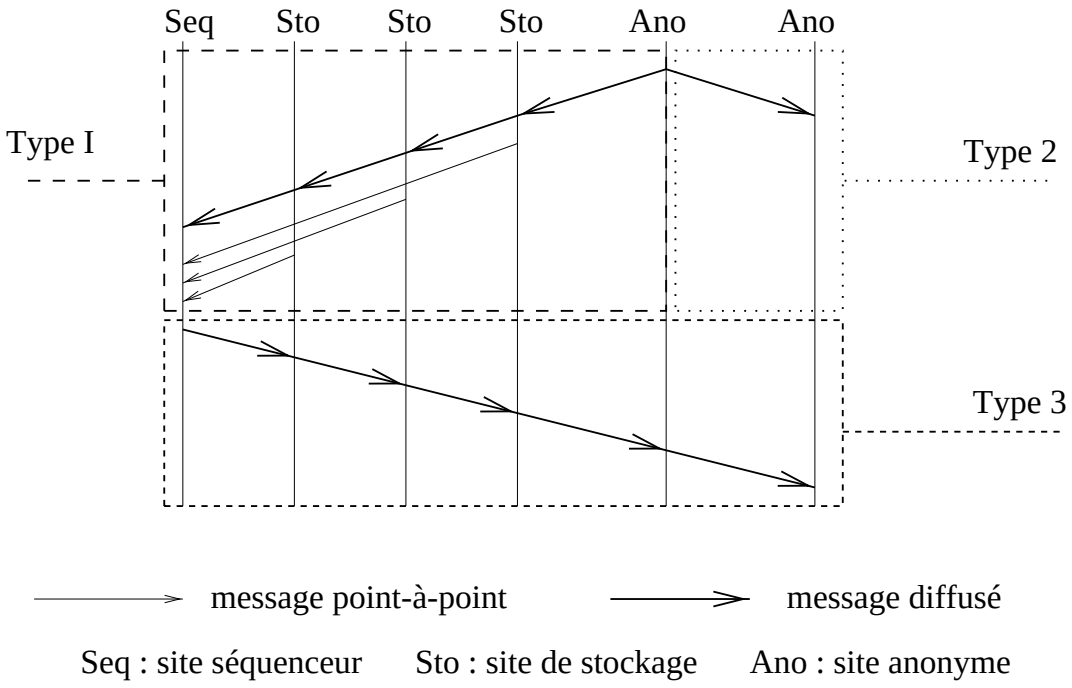


Fig. 2.5 : Différentes catégories de perte de message

Comme on le constate, trois types de pertes de messages peuvent être recensées :

- la perte d'un message de type 1, qui recouvre le message initial envoyé en diffusion et reçu par tous les sites de stockage ainsi que les acquittements des sites de stockage au séquenceur, entraîne la non−délivrance de l'estampille globale associée au message. Ce type d'erreur freine beaucoup le déroulement du protocole car la réémission par le site émetteur n'aura lieu qu'après l'expiration d'un délai de garde assez long.
- le deuxième type de perte a lieu lorsqu'un site anonyme ne reçoit pas le message initial contenant les données transmises et l'estampille locale associée. La perte est détectée lors de la réception du message contenant l'estampille globale associée et

nécessite de recharger le contenu du message depuis un site de stockage. Cette erreur nécessite d'envoyer à nouveau tout le contenu du message, mais ne bloque pas le déroulement du protocole.

- enfin le dernier type de perte concerne le message en provenance du séquenceur et attribuant l'estampille globale du message. La perte est détectée à la réception du message suivant de ce type et nécessite juste de redemander l'association entre estampille globale et locale à un site de stockage.

La plupart des pertes de type I et III peuvent être masquées en utilisant des techniques de duplication présentées ci−dessous.

II.4.2 Techniques de duplication

La méthode utilisée consiste à transporter des informations de contrôle − donc n'occupant que peu de place − en même temps que des messages d'un autre type, plus particulièrement ceux comportant des données. Par exemple, il est possible de transporter des associations <estampille globale,estampille locale> dans le même paquet que le message initial comportant les données émises par un site. Ce type de technique est appelé *piggy−backing* et sera référencé sous ce terme par la suite. Cette méthode été utilisée de deux manières.

II.4.2.1 Piggy−backing des estampilles globales

Il est possible de s'affranchir aisément des conséquences liées aux pertes de type 3 en utilisant de manière systématique la technique de piggy−backing pour les estampilles globales. Chaque site du groupe, même anonyme, conserve une petite liste comportant les dernières associations entre estampille globale et estampille locale des messages délivrés sur le site courant. Tous les en−têtes des messages émis en direction du groupe comportent une place réservée où est copiée cette liste avant l'émission d'un message. A chaque réception d'un message, cette liste est extraite de l'en−tête et toutes les associations manquantes sont analysées, entraînant la délivrance des messages associés. Le traitement de la liste est effectué avant même celui de l'en−tête, ce qui permet d'éviter des demandes de messages inutiles dans le cas où le message courant est justement une estampille globale en provenance du séquenceur.

L'expérience montre que ce mécanisme est suffisamment efficace pour éliminer complètement les demandes d'estampilles aux sites de stockage, du moins quand le message en question n'a pas été perdu lors de la première phase de diffusion. Ce mécanisme permet aussi d'éliminer une source de conflit rare mais impossible à traiter autrement lorsque le site séquenceur tombe en panne après la délivrance d'une estampille globale que seuls des sites anonymes ont reçu.

II.4.2.2 Piggy−backing des acquittements

Les pertes d'acquittements en provenance des sites de stockage vers le site séquenceur sont coûteuses car elles bloquent la délivrance du message, aussi il est utile d'utiliser la même technique de piggy−backing pour ce type d'information. Chaque site de stockage conserve un historique des derniers messages acquittés, sous la forme d'une liste

d'estampilles locale, et lors de chaque émission d'un message à destination du séquenceur (généralement un acquittement) cette liste est insérée dans l'en−tête du message. Lors de la réception le séquenceur peut alors compléter les acquittements des messages en attente, ce qui provoque souvent l'émission à ce moment de l'estampille globale associée. Cette technique permet de réduire en pratique les cas de non−délivrance de l'estampille globale aux seuls messages n'ayant pas été reçu par tous les sites de stockage.

Le principe du piggy−backing des acquittements et des estampilles globales permet de diffuser à l'ensemble des sites les dernières valeurs courantes utilisées par le protocole. Ce mécanisme agit donc comme un second protocole s'exécutant simultanément avec les communications du groupes et améliorant la cohérence de vue entre les sites en cas de perte de messages. La sous−section suivante s'attache à quantifier l'impact de ces mécanismes sur les performances du protocole.

II.4.3 Analyse statistique des optimisations

L'analyse théorique de l'impact des pertes de messages nécessite dans un premier temps de modéliser les différentes erreurs possibles. Le réseau Ethernet et les systèmes en temps partagé qui hébergent le protocole sont sujets à deux types d'erreurs entraînant la perte de messages :

- le réseau Ethernet est basé sur le principe d'un médium commun à toutes les machines. L'accès peut être concurrent, c'est−à−dire qu'il n'y a pas de régulation comme dans les réseaux à jeton, l'émission simultanée de deux machines différentes donne lieu à un incident nommé collision qui, s'il est détecté, annule la transmission courante. La retransmission a alors lieu après un délai aléatoire. Dans certaines conditions, il est donc possible qu'un message ne soit jamais émis, ou ne soit correctement reçu que sur une partie des sites pour les messages émis en diffusion.
- pour des raisons logicielles liées au système d'exploitation et aux couches d'accès au réseau, des paquets peuvent être perdus alors que l'interface les a correctement capturés. Ce type d'incident peut arriver lors de forts débits si la machine est trop chargée pour pouvoir traiter dans le temps imparti les paquets entrants.

La figure suivante résume différents points de pertes possibles dans le cas le plus général, celui d'une diffusion :

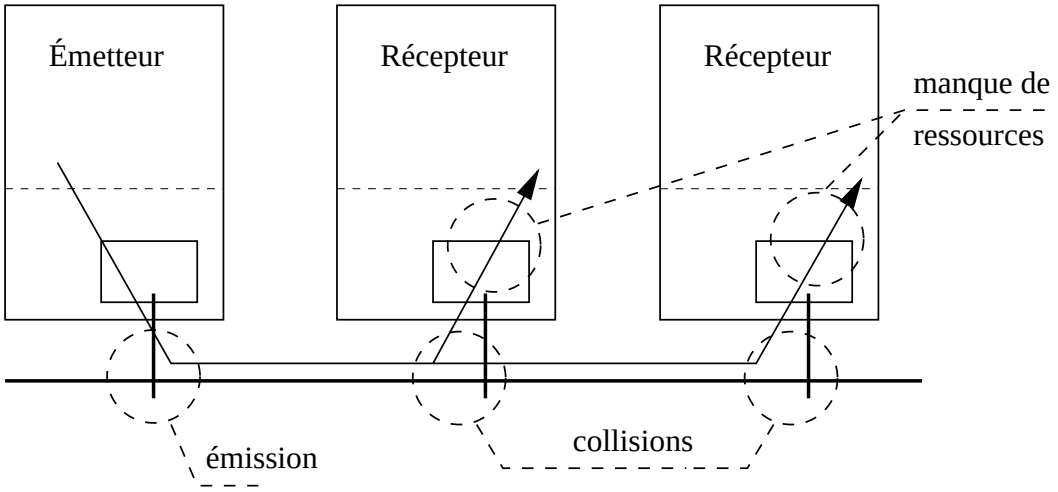


Fig. 2.6 : Analyse des points de perte possible lors d'une diffusion

On constate que la modélisation des pertes de messages nécessite la prise en compte de deux paramètres, la probabilité de non−émission d'un message modélisant les erreurs au niveau du site émetteur et la probabilité de non−réception de celui−ci sur un site destinataire. Dans le cadre de transmissions point−à−point, ces deux facteurs peuvent être réunis en un seul paramètre, mais l'utilisation de communications multi−points nous force à les séparer. La probabilité de perte d'un paquet lors de son émission est notée P_e par la suite, de même P_r désigne la probabilité de perte lors d'une réception.

La probabilité de la délivrance d'un message dépend :

- Du site considéré pour l'émission, par exemple s'il s'agit d'un site de stockage non séquenceur, un message de moins est requis, à savoir l'acquittement de ce site de stockage vers le séquenceur.
- Du site récepteur, car s'il coïncide avec le site émetteur, la probabilité de perte du message initial est nulle.
- Du nombre de sites de stockage utilisés.

Considérons la probabilité de délivrance d'un message sur un site anonyme, le message ayant été émis en direction du groupe par un autre site anonyme, le quorum étant formé de K sites de stockage. La délivrance de l'estampille globale nécessite l'émission du message initial, sa bonne réception sur tous les sites du quorum, puis l'émission et la réception des acquittements vers le séquenceur.

Les calculs faits par la suite supposent les hypothèses simplificatrices suivantes :

- les pertes de messages sont indépendantes entre elles, ce qui n'est pas forcément réaliste, par exemple lors de la saturation des tampons en réception sur un site.

- on ne tient compte que des pertes de messages, l'impact des pannes de sites ou d'une dégradation irréversible des conditions d'émission ou de réception n'est pas traitée.

Dans le cadre du protocole initial on obtient alors :

$$P_{global} = 1 - (1 - P_e) * (1 - P_r)^K * [(1 - P_e) * (1 - P_r)]^{K-1}$$

$$\boxed{P_{global} = 1 - (1 - P_e)^K * (1 - P_r)^{2K-1}}$$

Pour le protocole optimisé, la duplication des acquittements rend leur perte improbable, ce qui fournit le résultat suivant

$$\boxed{P^{opt}_{global} = 1 - (1 - P_e) * (1 - P_r)^K}$$

La délivrance du message sur le site est aussi conditionnée par sa réception lors de la diffusion initiale et l'envoi de l'estampille globale du séquenceur vers le site courant. On obtient donc dans le cas du protocole initial une probabilité de bonne réception, c'est−à−dire sans recourir à des mécanismes de gestion d'erreur, qui s'exprime par la formule suivante :

$$\boxed{P_{total} = 1 - (1 - P_e)^{K+1} * (1 - P_r)^{2K+1}}$$

Pour le protocole optimisé, le piggy−backing des estampilles globales permet d'annuler les conséquences de la perte du message final, d'où la formule :

$$\boxed{P^{opt}_{total} = 1 - (1 - P_e) * (1 - P_r)^{K+1}}$$

Le gain obtenu est très significatif, on peut remarquer que la version améliorée permet de s'affranchir de la gestion explicite de toutes les pertes de messages exceptées celles survenant sur le message initial. La dernière formule traduit le fait qu'une gestion d'erreur doit être mise en œuvre si le message initial n'est pas émis ou n'est pas correctement reçu sur les sites destinataires. La figure suivante illustre graphiquement le gain obtenu dans le cas où K vaut 5 :

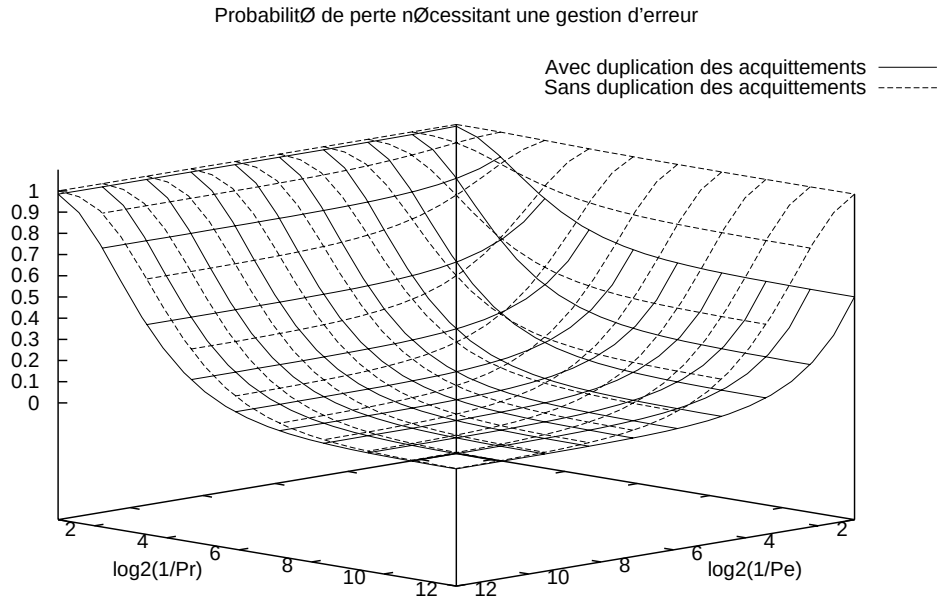


Fig. 2.7 : Graphe de la probabilité de perte pour les deux protocoles

Cette figure montre l'évolution des valeurs obtenues pour la probabilité de perte entraînant une gestion d'erreur. Cette courbe est fonction des deux paramètres et son échelle est représentée de manière logarithmique afin de mettre l'accent sur les zones des pertes élevées. Les courbes associées au protocole initial et au protocole optimisé ont été tracées pour des taux de perte compris entre 1/2 qui constitue un cas extrême inacceptable en pratique et 1/4096 qui est proche des conditions normales de fonctionnement d'un réseau local..

On observe une nette diminution des taux de perte en particulier quand les pertes en réception augmentent. L'amélioration est encore plus significative lorsque l'on observe des taux de perte en émission élevée. Si l'on recoupe avec les pannes réelles que modélisent ces deux paramètres, on peut donc espérer une nette amélioration de la résistance du protocole aux mauvaises conditions réseau, telles que les collisions, mais un gain moindre pour les pertes occasionnées par une surcharge du site récepteur.

Cette analyse théorique manque de finesse, car nous nous sommes basés sur une distribution monotone des pertes de messages dans le temps. Si les erreurs liées à des problèmes physiques telles que les collisions ou la perte de message dues à des mauvaises transmissions suivent en première approximation une telle loi de distribution, ce n'est pas le cas des pertes par manque de ressources sur une machine destinataire. En effet, si un paquet est rejeté, par exemple par manque de mémoire pour l'accepter, il est très probable que les messages suivants subiront le même sort. L'évolution dans le temps de la variable doit pouvoir être modélisée avec plus d'exactitude par une fonction créneau dont la fréquence des pics et leur durée suivent des distributions classiques.

Cette analyse, même sommaire, de l'impact du piggy−backing des acquittements et des estampilles globales, montre l'amélioration notable des performances du protocole que l'on est en droit d'attendre. En effet, ces changements permettent d'éviter de bloquer la transmission d'un message jusqu'à l'expiration d'un délai de garde sur le site émetteur, pour toutes les pertes de type III et la majorité des pertes de type I. Le gain principal espéré consiste en la diminution des latences en présence de pannes, mais aussi en une amélioration de la stabilité du protocole dans des conditions difficiles.

II.4.4 Technique de détection rapide

Comme expliqué au II.3.2 nous n'avons pas voulu faire appel à des hypothèses de délai stricts de transmission pour l'implantation du protocole. Les délais de détection de perte de message sont donc très longs proportionnellement aux temps de transmission effectifs. Les optimisations détaillées dans la section précédente permettent d'éliminer toutes ces latences dues aux pertes de messages sauf dans le cas précis où un des sites du quorum ne reçoit pas le message initial.

Pour garantir qu'un message ne sera pas associé plusieurs fois à une estampille globale, il faut être très vigilant lors des retransmissions. Nous avons choisi de ne laisser cette possibilité qu'au site émetteur ; en outre, chaque envoi d'un message se voit associé un numéro de retransmission augmentant à chaque réémission. De cette manière l'estampille locale associée reste identique mais il est possible de distinguer s'il s'agit d'une retransmission, et de l'identifier de manière unique. Cela permet d'éviter les doubles délivrances.

Reste à essayer de minimiser le délai en cas de perte du message par un site de stockage, afin d'améliorer la réactivité du protocole en cas de pertes. Deux méthodes viennent à l'esprit :

- une méthode passive, consistant à diminuer le temps à partir duquel le site émetteur, ne voyant pas d'estampille globale associée à son message , le réémet. Ce serait en fait diminuer les temps de transmission maximaux que l'on tolère et s'oppose à notre hypothèse de ne pas faire reposer le principe du protocole sur des délais de transmissions faibles.
- une méthode active, consistant en une détection de la perte au sein du quorum et sa notification à l'émetteur pour qu'il retransmette le message sans délai.

La figure suivante illustre un cas trivial de détection de la perte d'un message par l'un des membres du quorum, il s'agit de la perte du message par le séquenceur. Elle se produit lorsque le séquenceur reçoit un acquittement pour un message qu'il ne connaît pas :

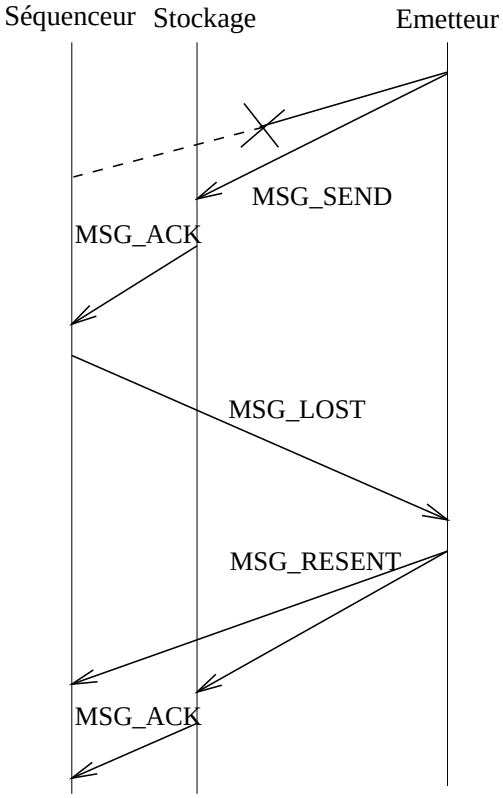


Fig. 2.8 : Détection de perte par le séquenceur et demande de réémission

Lors de la détection de ce type d'incident, le séquenceur indique à l'émetteur de retransmettre le message. L'identification du site émetteur est possible à l'aide de l'estampille locale. La mise en place effective de ce mécanisme nécessite de conserver une trace de la demande de réémission pour ne pas émettre autant de messages de type MSG_LOST qu'il y a de sites de stockage.

Cette technique permet d'accélérer facilement le traitement des pertes de messages par le séquenceur mais statistiquement il est plus probable que le phénomène inverse se produise, c'est−à−dire la perte du message par un site de stockage non séquenceur. En pratique cela se traduit par la réception de la quasi−totalité des acquittements sur le site séquenceur, mais l'estampille globale n'est pas délivrée. Le séquenceur ne peut décider a priori si les sites n'ayant pas encore répondu ont perdu le message initial, ou si l'acquittement tarde juste à être délivré. Il serait possible par exemple d'installer un délai de garde dès la réception du premier acquittement, qui lorsqu'il se déclenche demande une réémission du message. Mais le respect des hypothèses concernant les délais de transmission rend cette méthode aussi inefficace que la réémission effectuée normalement par le site émetteur.

Une autre technique, proche dans l'esprit des méthodes de piggy−backing appliquées avec succès pour les acquittements et les estampilles globales, consiste à rajouter une information sur les messages reçus mais non complètement acquittés à chaque message émis par le séquenceur et à destination des sites de stockage. Ceux−ci peuvent alors demander au séquenceur les messages manquants et lui envoyer l'acquittement manquant. Mais cette

méthode n'a pas été implantée car elle surcharge le site séquenceur qui est déjà le goulot d'étranglement du protocole sous fortes charges.

II.5 Conclusion

Le protocole mis en œuvre s'inspire de celui utilisé pour Amœba. Il a l'avantage d'être très simple et rapide, mais repose sur l'utilisation d'un site séquenceur statique. Il utilise de manière optimale la diffusion physique d'Ethernet ce qui nous permet de gérer l'ensemble des participants sous la forme d'un groupe opaque. La connexion et la déconnexion des sites n'ont alors aucun impact sur le fonctionnement du protocole, sauf lors de la suppression d'un des sites de stockage où un protocole de reformation d'un quorum cohérent est nécessaire. Des optimisations, principalement basées sur le principe de piggy−backing, permettent de diminuer fortement l'impact de perte de messages et d'assurer une bonne réactivité du protocole, critère important dans le cadre d'applications interactives. Mais les performances du protocole sont aussi très dépendantes des choix d'implantation que nous détaillons au chapitre suivant.

Chapitre III

Mise en œuvre

Ce chapitre présente l'implantation du protocole. Les principaux choix de réalisation et leurs conséquences sont présentés, puis des détails techniques sur les protocoles sont fournis. Enfin nous détaillons les diverses architectures systèmes qui ont servi de cible à l'implantation, les interfaces nécessaires et les conséquences pratiques des singularités de chaque système.

III.1 Choix de réalisation

La mise en œuvre du protocole a été fortement conditionnée par plusieurs choix de réalisation concernant l'architecture logicielle du protocole, la gestion de la mémoire et le modèle d'exécution. Les choix d'architecture et d'implantation qui ont guidé la réalisation du protocole ont été motivés par les points suivants :

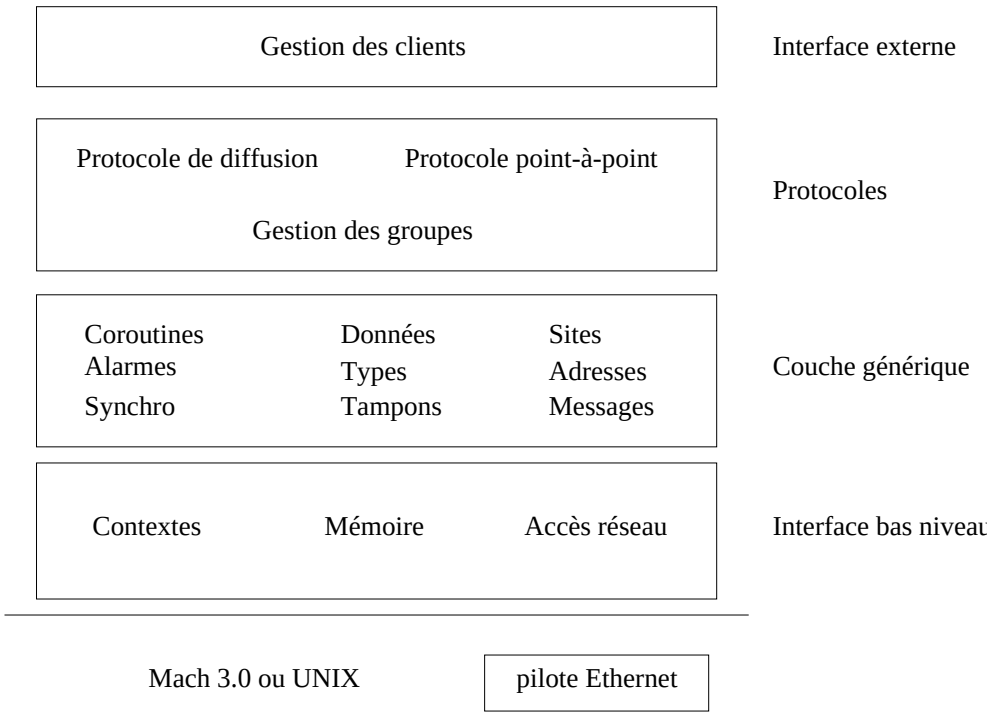
- la portabilité du protocole est un point essentiel. La mise en œuvre initiale sous la forme d'un serveur s'exécutant en mode utilisateur au dessus du noyau Mach 3.0 a permis de dégager les quelques primitives système nécessaires au portage sur de nouvelles architectures.
- plusieurs projets de recherche tels que le x−Kernel I.2.5 et Horus I.2.6, ont prouvé l'intérêt d'une architecture "ouverte" pour la réalisation de protocoles de communication au sein de systèmes distribués. Offrir un protocole figé sans possibilité de configuration limite fortement les applications susceptibles de l'utiliser. Sans forcément fournir plusieurs types de protocoles, la conception logicielle doit permettre d'étendre ou de modifier aisément l'implantation standard pour répondre à de nouveaux besoins.
- il a été démontré que le coût induit par de multiples recopies en mémoire est un des principaux facteurs de dégradation des performances des protocoles de communication. Ce phénomène est particulièrement sensible sur les machines de type IBM−PC utilisées pour les premiers développements car leur bande passante mémoire est très faible.
- l'utilisation des possibilités de diffusion physique offerte par les réseaux Ethernet est un des points clés nécessaires pour obtenir de bonne performances dans le cadre d'un réseau local.
- le protocole est prévu pour s'exécuter sur un réseau local, nous avons donc fait l'hypothèse classique consistant à supposer que les délais de transmission sont bornés. Toutefois nous avons limité son emploi à la détection d'éventuelles pannes ou

pertes de messages, le bon fonctionnement du protocole ne s'appuie pas sur cette hypothèse.

- le protocole sera structuré sous forme de threads / coroutines.
- les données échangées sont de petite taille, donc il n'est pas nécessaire de fournir un contrôle de flux.

III.1.1 Architecture logicielle

Les points énoncés précédemment nous ont conduits à choisir l'architecture logicielle décrite par la figure suivante :



Le protocole a été conçu pour être exécuté sous la forme d'une tâche Mach ou d'un processus Unix. Il s'exécute en mode utilisateur et n'a donc pas a priori d'accès direct aux mécanismes de base du noyau sous−jacent (en particulier l'accès au pilote Ethernet). Une couche d'interface basse "isole" donc le protocole des particularités du système ; elle fournit une interface standard pour toutes les ressources des bas niveau. S'appuyant sur cette interface, une couche générique fournit un ensemble de services nécessaires à la mise en œuvre de protocoles, une attention particulière étant portée aux problèmes d'allocation de mémoire et de mouvements de données. Deux protocoles ont été implantés, le protocole de diffusion fiable qui a fait l'objet du chapitre précédent, ainsi qu'un protocole point−à−point fiable bidirectionnel basé sur le principe classique des fenêtres d'acquittement.

III.1.2 Interface avec le système

L'intérêt de cette couche d'interface est apparu dès la première mise en œuvre sur le micro−noyau Mach 3.0. Pour des raisons d'efficacité, il semblait plus intéressant d'utiliser les primitives du micro−noyau que les fonctions standard Unix fournies par le serveur OSF/1. L'emploi de primitives isolant le code du protocole des spécificités du système de base nous a alors semblé naturel afin de faciliter le portage sur des architectures plus traditionnelles (et pour l'utilisation d'un émulateur décrit au chapitre IV suivant). La première étape de l'implantation a donc consisté à dégager les fonctions de base du système opératoire qui nous sont nécessaires :

- L'accès au réseau est le point le plus critique. En effet, la mise en œuvre est basée sur l'utilisation de la diffusion physique au niveau Ethernet. La figure suivante rappelle la structure d'un paquet Ethernet.

  Il nous faut émettre des trames Ethernet en spécifiant le champ destinataire de la trame (mis à 0xFFFFFFFFFFFF pour de la diffusion physique) ainsi que le champ type qui précise le protocole client. Il faut aussi pouvoir recevoir les paquets concernant le protocole en les filtrant grâce au champ type. Trois primitives sont donc nécessaires : Initialisation_Ethernet exécutée une seule fois permet l'accès au pilote Ethernet, et fournit des renseignements nécessaires tels que le numéro de l'interface utilisée, et l'adresse physique de celle−ci sur le réseau. Deux primitives Send_Ethernet et Recv_Ethernet permettent respectivement l'envoi et la réception d'un paquet sous la forme d'une zone de mémoire contiguë et d'un entier spécifiant sa taille. Si la primitive d'envoi peut être indifféremment bloquante ou non, la primitive de réception est supposée bloquante.

- Le modèle d'exécution d'un protocole un tant soit peu complexe est par nature multiflot, c'est à dire composé d'un ensemble de tâches (processus séquentiel) qui coopèrent pour offrir un service. Par exemple, un flot est requis pour la réception des paquets en provenance du réseau (réception bloquante), un autre sert les requêtes en provenance des clients, enfin la gestion des délais de garde peut être effectuée par un autre processus séquentiel. La structuration du protocole sous la forme d'un ensemble de tâches coopérantes nous a semblé un bon modèle, mais sa mise en œuvre nécessite un support système sous la forme de *threads* (processus léger partageant un même espace d'adressage) ou de *coroutines* (même principe mais le noyau ne gère pas l'ordonnancement des flots entre eux). Le micro−noyau Mach fournit par exemple des threads, mais ceux−ci ne sont pas disponible en standard sous Unix. Pour implanter une bibliothèque de coroutines trois primitives sont nécessaires : il faut pouvoir sauver et restaurer un contexte d'exécution d'un flot (Sauver_Contexte et Restaurer_Contexte) et fournir un mécanisme de Test_and_Set atomique permettant de synchroniser deux flots en concurrence.

- Enfin, le système doit offrir une primitive permettant d'allouer des blocs de mémoire Allouer_Memoire. Cela n'est en pratique pas un problème car la fonction de la librairie C *malloc* est disponible à cet effet. Toutefois, il s'est avéré utile d'encapsuler cet appel car cela permet d'effectuer un contrôle des allocations mémoire. De plus, lors du portage du protocole au sein du micro−noyau Mach 3, les fonctions d'allocation mémoire standard n'étaient pas disponibles, et seul le code de la fonction d'allocation a du être modifié.

En conclusion, les dépendances systèmes sont peu nombreuses, ce qui est dû à la nécessité de travailler à très bas niveau. Le protocole peut alors être porté sur des plateformes minimales pourvues de ces quelques primitives. Par contre l'accès à ces primitives risque a priori de poser problème sur des systèmes tels qu'Unix qui restreignent l'accès aux ressources à travers des interfaces de haut niveau. En pratique, tous offrent ce type d'accès (en particulier ceux permettant d'accéder au réseau), mais les interfaces ne sont pas standardisées.

III.1.3 Interfaces et types internes

L'étape suivante de la conception a été la recherche des services génériques nécessaires à la réalisation de protocoles. Le but recherché est d'isoler les mécanismes et les abstractions facilitant la mise en œuvre de protocoles variés. La couche générique doit simplifier l'implantation, sans toutefois faire trop d'hypothèses sur le comportement des protocoles.

Les principaux objets manipulés par des protocoles sont :

- des blocs de données, sous diverses formes telles que des paquets, des messages, ou des informations à transporter.
- des informations sur des machines, telles que leur adresse.
- des délais de garde. En effet, le seul moyen de tenter de détecter une perte de message sur les réseaux actuels est de constater l'absence d'acquittement une fois un certain laps de temps écoulé.

De plus la structuration d'un protocole peut bénéficier d'une architecture multi−flots ; la couche générique fournit donc les primitives nécessaires à la structuration du protocole sous la forme d'unités d'exécution concurrentes. Nous détaillons ci−dessous les structures et les primitives exportées par la couche générique :

III.1.3.1 Structures d'exécution et délais de garde

Les interfaces suivantes permettent d'implémenter un protocole sous la forme de flots d'exécution indépendants, qui se synchronisent lors d'accès aux variables partagées par le biais de sections critiques. On notera que ces primitives sont très classiques, la plupart d'entre−elles sont présentes sous une forme voisine dans le micro−noyau Mach et dans la norme 1004.3 relative à la mise en œuvre de threads dans un environnement POSIX :

- MkThread : fonction de création, qui prend comme argument la fonction associée au nouveau flot et son nom et retourne un descripteur de type `T_machinter_Thread` pour la nouvelle entité.

- KillThread : fonction de destruction, utilise un seul argument, le descripteur du flot à supprimer.
- SetPriority : cette fonction permet de réajuster la priorité d'un flot d'exécution sur les systèmes qui le supportent. En pratique l'expérience prouve que son utilisation peut être dangereuse car sur certains systèmes tels que Mach 3, l'affectation de priorités trop hautes peut complètement geler le système. La mise en œuvre sous UNIX n'utilise pas de priorité.
- Schedule : cette procédure appelle l'ordonnanceur qui peut choisir de stopper le flot courant pour affecter le processeur à une autre tâche.
- SuspendThread et ResumeThread sont deux fonctions duales dont le but est de suspendre ou de réactiver l'exécution d'un flot dont le descripteur est donné en argument.
- GetTime : fournit une horloge locale au site exprimant la durée en dixièmes de secondes depuis le lancement du processus.
- Sleep : force le flot courant à l'inactivité pendant un intervalle de temps minimum (exprimé en dixièmes de secondes).
- MkCritical crée une variable destinée à protéger une section critique ou l'accès à des variables partagées entre plusieurs flots, et retourne un identificateur de type `T_machinter_CriticalSection`.
- CriticalEnter et CriticalLeave sont les deux routines permettant l'utilisation de l'exclusion mutuelle, la routine CriticalEnter est susceptible de bloquer le flot courant à cet effet, jusqu'à ce que le verrou soit relâché par CriticalLeave.

Un délai de garde est défini comme une routine donc l'exécution est planifiée à l'avance pour avoir lieu après un certain laps de temps, à moins qu'elle ne soit explicitement annulée. Il n'y a donc que deux fonctions associées :

- MkTimeOut : fonction de création dont les arguments sont le délai, la routine d'exception et ses paramètres.
- KillTimeOut : fonction de destruction associée.

La mise en œuvre des délais de garde est basée sur l'utilisation d'un flot réservé à leur gestion qui vérifie périodiquement l'expiration des délais et exécute dans son contexte propre les routines de traitement associées.

III.1.3.2 Gestion de données

L'efficacité de l'implantation d'un protocole est en grande partie liée [22][26][27] à la minimisation des mouvements de données entre les différentes couches logicielle qui le composent. Il nous a donc paru important de fournir un ensemble de primitives gérant de manière cohérente les différentes représentations qui y sont associées dans les diverses couches. En effet, si les couches hautes manipulent des blocs de mémoires typés fournis par les applications clientes, à bas niveau tout est représenté sous la forme de paquets Ethernet vus comme un tableau de caractères de taille bornée.

Dans toute la couche générique, la gestion de la mémoire en provenance des clients est basée sur un type de base `T_machinter_Data` qui représente un bloc de données contiguës, typées afin de pouvoir supporter des systèmes hétérogènes. Des primitives permettent de gérer les types qui leur sont associés, de transmettre ces fragments entre sites (cf. III.1.3.3), et de gérer le stockage de ces éléments. Afin de limiter les recopies au strict nécessaire, les stuctures composant le type `T_machinter_Data` ne référencent les données que de manière indirecte via un pointeur ; on y associe aussi leur taille, un compteur de référence, une clé et un pointeur vers le type des données. Les types associés aux informations transportées ou stockées par les protocoles sont construits par agrégation d'un ensemble de types élémentaires reconnus par la couche générique, qui permet de construire des représentants pour tous les types usuels représentés par une zone de données contiguë.

La mise en œuvre des protocoles nécessite de conserver temporairement les données transitant via le réseau par exemple tant que les destinataires n'en ont pas accusé réception. La couche générique fournit donc un service de tampons permettant le stockage et la recherche de données de manière efficace. Le type associé `T_machinter_Buffer` permet donc de stocker des éléments du type T_machinter_Data et offre deux types de recherche : soit par clé (entier sur 32 bits), soit FIFO.

L'allocation des structures internes à la couche générique est basée sur un principe de réservation statique d'un nombre important de chaque élément, une liste des blocs libre est alors maintenue permettant de fournir rapidement toute demande sans avoir à recourir à des allocations "au vol". Un gain de performance notable peut être obtenu lors d'un fonctionnement sur une machine chargée en gérant ces listes en LIFO (Last In First Out) car cela limite les accès aléatoires en mémoire virtuelle (le protocole s'exécute au sein d'un processus). Le confinement des accès à une zone limitée de l'espace d'adressage évite les accès disques dus aux défauts de page et offre en pratique de meilleurs résultats dans ces conditions.

Les primitives de manipulation des types associés aux données sont les suivantes :

- MkType retourne un identificateur de type dont le contenu est vide, équivalent logique de `void` en C.
- AddTypeField permet de construire des types complexes en concaténant un tableau d'éléments d'un type donné à la description existante d'un type passé en premier argument. Par exemple, l'appel `AddTypeField (type, MACHINTER_TYPE_BIT16, 3)` ajoute à la description de `type` un tableau de trois entiers courts codés sur 16 bits chacun.
- KillType permet de détruire un identificateur de type.

Une fois que le type associé à un bloc de données a été construit, il est possible de créer l'instance du T_machinter_Data qui va servir à référencer cette portion de mémoire :

- MkData construit un objet à partir de l'adresse des données, de leur taille, d'une clé de référence codée sous la forme d'un entier sur 32 bits, et du type associé aux données.
- KillData permet de détruire une instance d'un T_machinter_Data. Un mécanisme de compteur de références permet d'éviter la destruction si cet objet est par exemple

référencé dans un buffer. Cette routine peut avoir des effets de bord dus à la désallocation de structures associées à l'objet libéré, par exemple appeler KillType sur le type associé (un compteur des références est aussi associé aux types) ou libérer de la mémoire allouée pour les données lors de leur réception sur ce site.

- Les fonctions DataAddr, DataLen et DataKey permettent de lire respectivement l'adresse, la taille ou la clé associée d'un objet memoire donné en argument.

D'autres fonctions annexes axées autour de la gestion des blocs de données sont présentes, essentiellement pour faciliter le développement. En particulier il est possible à tout moment d'énumérer les blocs alloués, d'afficher leur contenu (les informations de type associé facilitent l'opération). En effet, en pratique l'implantation d'un protocole nécessite de s'assurer que tout bloc de mémoire alloué sera libéré, sous peine d'un "crash" à plus ou moins longue échéance. Les primitives de gestion de tampons sont précisement fournies pour pouvoir stocker des données dont on ne peut immédiatement se débarrasser :

- MkBuffer permet de créer un tampon, un argument permet d'en préciser le type : `SORTED_BUFFER` ou `FIFO_BUFFER` suivant le type de méthode d'extraction nécessaire.
- KillBuffer et DestroyBuffer permettent de détruire un tampon, la deuxième procédure forçant la destruction des données qui y sont stockées.
- BufAdd permet d'insérer un élément à un tampon, et la routine duale BufSub permet d'extraire un élément, BufClean enlève tous les éléments d'un tampon.
- BufHead renvoie le premier élément d'un tampon, dans le cas d'un tampon trié, la fonction renvoie l'élément de plus petite clé.
- BufSeek recherche un élément possédant une clé dont la valeur est passée en argument, cette fonction échoue sur un tampon FIFO.
- CheckBufferData est une fonction permettant la manipulation globale des éléments stockés dans un tampon. Sa signature C est `CheckBufferData (T_machinter_Buffer buffer, T_Bool (*TestFunc) (T_machinter_Data data), T_Void (*ExecFunc) (T_machinter_Data data))` . Tous les éléments sont passés sucessivement en argument de la première fonction, et si le résultat est vrai, la deuxième fonction est appliquée. La séparation entre test et traitement sert à minimiser le nombre d'entrées et sorties de section critiques, car le test doit s'effectuer en exclusion mutuelle.

De même, des primitives nécessaires aux phases de mise au point permettent de vérifier l'intégrité des tampons, d'en afficher leur contenu, etc.

III.1.3.3 Primitives de communication

Deux hypothèses majeures qui ont conditionné la structuration du protocole sont les suivantes :

- Le support réseau est un LAN Ethernet, ce qui entraîne en pratique un taux de perte de paquet relativement faible, sauf en cas de panne matérielle ou de saturation des couches du protocole.

- La taille des messages échangés est en moyenne petite, le protocole n'étant pas destiné à faire du transport de données (tel que FTP) mais à servir d'outil de construction pour système réparti.

Nous avons donc choisi d'offrir au niveau de la couche générique des primitives de transport non fiables, bien que supportant le mécanisme de fragmentation et réassemblage pour des messages de plus de 1500 octets (taille limite liée aux trames Ethernet). En cas de perte d'un paquet l'ensemble du message est perdu et la gestion de l'erreur est laissée aux couches supérieures. Ce choix est acceptable dans le cadre des hypothèses énoncées précédemment, le transport de grand blocs de données nécessitant alors d'implémenter un mécanisme de fragmentation à plus haut niveau avec gestion des erreurs. L'avantage principal est de pouvoir offrir une interface simplifiant l'accès au réseau sans imposer de politique de gestion des erreurs, celles−ci sont du ressort des protocoles mis en œuvre au dessus de la couche générique. La figure suivante illustre cette architecture :

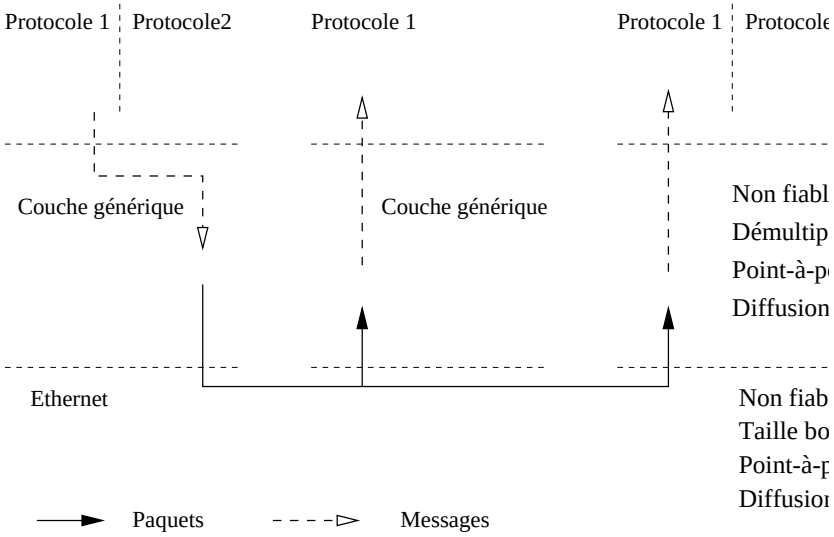


Fig. 3.1 : Architecture des couches basses des protocoles

Un message est constitué d'un bloc d'en−tête et d'une suite de blocs de données typées (du type `T_machinter_Data` décrit au III.1.3.2, désigné par la suite par Data). Ce support du typage est conservé au sein de la couche générique, les informations de type et les données elles−mêmes sont émises simultanément et le type associé aux données est reconstitué sur les sites destinataires. Cela permet de simplifier l'implantation de protocoles en environnement hétérogène car la conversion entre les représentations physiques peut alors être réalisée de manière transparente au sein de la couche générique, au détriment d'une légère augmentation du volume total des données transférées.

L'utilisation de la diffusion physique a pour inconvénient que tous les sites connectés au réseau local reçoivent les paquets. Afin de limiter le traitement associé, lors de la réception du premier paquet relatif à un message, le header est transmis au protocole avant même la phase de reconstitution des Data associés. En retour le protocole indique si le message l'intéresse ou non, et la suite du traitement est alors mise en œuvre seulement si nécessaire.

Cela permet de diminuer la charge dans le cas du traitement de messages de contrôle ou si le site n'appartient pas au groupe destinataire du message. Ce mécanisme est décrit par la figure suivante :

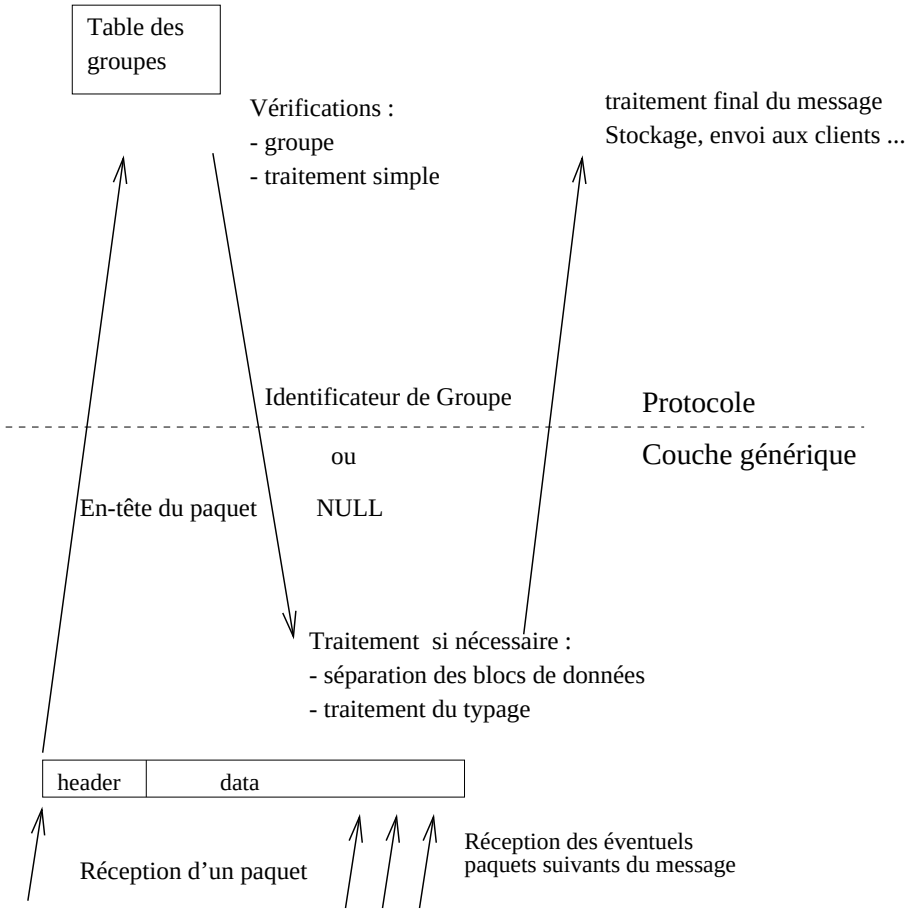


Fig. 3.2 : traitement à deux phases des messages en réception

La gestion de la mémoire dans la couche générique tente au maximum d'optimiser les point suivants :

- limiter les recopies de données
- simplifier l'implantation de protocole en les dechargeant des opérations complexes (allocation, fragmentation, typage, etc.).

La figure suivante illustre l'application de ces principes lors de la réception d'un message formé de plusieurs blocs. Le premier bloc tient complètement dans un paquet reçu et le Data associé référence directement cette zone mémoire, par contre le deuxième a nécessité l'allocation d'une zone propre, ce qui a forcé la recopie des données. Mais du point de vue du protocole ces subtilités sont cachées et l'interface d'accès au Data est identique.

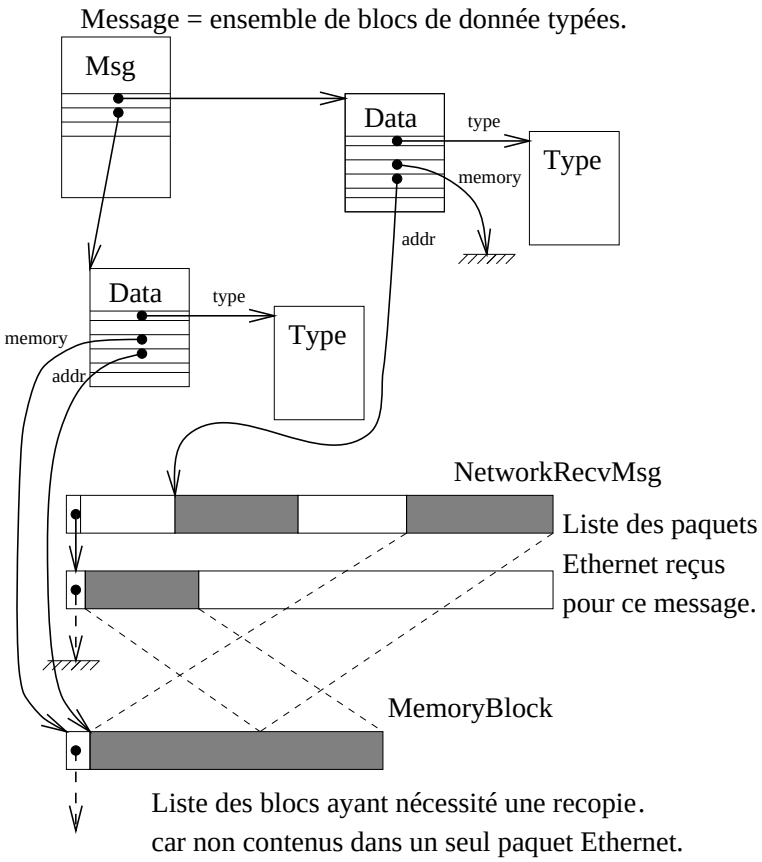


Fig. 3.3 : Gestion mémoire en réception d'un message

La partie réseau de la couche générique effectue aussi les associations entre adresses logiques utilisées dans les couches hautes du protocole et les adresses physiques associées aux adaptateurs Ethernet. Les deux types associés sont :

- `T_machinter_NodeAddress` pour les adresses réseau, ce qui correspond pour Ethernet à un tableau de 6 octets. La variable machinter_BroadcastAddress permet de désigner l'adresse physique utilisée pour la diffusion sur Ethernet.
- `T_Site` pour les adresses logiques, il est codé sur un entier non signé de 32 bits et est initialisé par défaut avec l'adresse Internet du site correspondant.

A chaque site on peut aussi associer un bloc de données (un Data) pour y stocker des informations spécifiques.

Les interfaces d'accès exportées par la couche générique sont les suivantes :

- StartNetwork : initialise la couche réseau et associe au site courant une adresse logique donnée en argument.
- AddHost : rajoute un élément à la base de sites gérée par la couche générique, ses arguments sont une adresse physique et l'adresse logique correspondante. On peut redéfinir l'adresse logique d'un site déjà existant dans la base ou donner une adresse logique −1 si celle−çi n'est pas encore connue.

- SetHostData : permet d'associer un bloc de données à un site de la base désigné par son adresse physique.
- GetHostData : fonction inverse de la précédente, qui recherche le bloc de données associé à un site.
- GetHostSite : recherche l'adresse logique d'un site dont l'adresse physique est donnée en argument
- GetHostAddress : recherche de l'adresse physique d'un site dont l'adresse logique est fournie en argument.

Les primitives suivantes traitent de la manipulation du type `T_machinter_Msg` correspondant aux messages :

- MkMsg renvoie une structure de type message à destination d'un site dont l'adresse physique est fournie en argument.
- MsgAddHead permet d'insérer un bloc de données en tête d'un message. Aucune recopie de donnée n'est effectuée, on se contente de rajouter un pointeur vers le Data fourni en argument en début de structure.
- MsgAddQueue, du même type permet d'ajouter des données en fin de message.
- MsgGetData est la fonction permettant d'extraire des données associées a un message. Chaque appel renvoie le Data suivant stocké dans le message après l'en avoir supprimé.
- MsgKillMsg permet de détruire un message dont toutes les données ont été lues.
- MsgDestroy doit elle être appelée lors de la destruction d'un message incomplètement lu, les blocs de données encore pointés par le message sont alors désalloués.

Enfin les trois fonctions suivantes permettent la réception et l'envoi de messages :

- SetCheckFunc permet aux couches utilisatrices de définir la fonction qui sera appelée lors de la réception du premier paquet d'un message. Le type associé est, suivant la syntaxe C :

```
void * (*T_machinter_ChkFunc)
      (T_machinter_NodeAddress from, T_Void *info_header,
                        T_Bool is_broadcast).
```

  Les paramètres sont l'adresse physique du site émetteur, un pointeur sur l'en−tête du message, et un booléen indiquant s'il s'agit d'un message point−à−point ou émis en diffusion. Si le protocole souhaite que le traitement des éventuels paquets suivants et que l'analyse des types et la décomposition en Data soient effectués, il renvoie un pointeur qui permettra d'aiguiller le message ainsi formé vers le groupe destinataire.
- SetRecvFunc permet de spécifier la fonction a appeler après le décodage complet d'un message en provenance de réseau. La signature associée à cette fonction est :

```
void (*T_machinter_RecvFunc)
    (T_machinter_NodeAddress from, void *group,
```

```
void *info_header, T_machinter_Msg msg,
  T_Bool is_broadcast)
```

Les paramètres supplémentaire sont `group`, valeur retournée par la première étape du décodage ainsi que le message transmis aux couches supérieures implémentant le(s) protocole(s).

- MsgSend sert à transmettre un message sur le réseau. Ses arguments sont un pointeur vers l'en−tête du message, sa taille, le message à transmettre et l'adresse physique du destinataire.

On notera que les primitives de communication permettent un traitement rapide des messages simples tels que des acquittements, en effet le message n'est composé que de l'en−tête et seule la première étape du décodage doit être effectuée. Il n'y a pas de surcoût induit par la couche générique. Ce mécanisme de décodage en deux étapes peut surprendre car par suite de perte de messages, il est tout à fait possible qu'un message dont l'en−tête a été analysée lors de la réception du premier paquet ne soit pas transmis du fait de la perte d'un des messages suivants composant le message. Par exemple le traitement d'informations de *piggy−backing* contenues dans l'en−tête est possible mais nécessite des précautions.

La section suivante illustre l'utilisation de cette couche générique pour la réalisation de deux protocoles.

III.2 Protocoles mis en œuvre

Les deux protocoles implantés sont ceux dont le principe est décrit au chapitre précédent II, à savoir un protocole de diffusion fiable, garantissant un ordonnancement total et résistant aux pannes de sites, ainsi qu'un protocole de communication fiable point−à−point entre deux participants d'un groupe. Cette section se borne donc à exposer et justifier les choix liés à la mise en œuvre des groupes et des protocoles.

III.2.1 Implantation des groupes

Les groupes sont des éléments visibles depuis les applications utilisatrices des protocoles, et doivent donc pouvoir être désignés sous une forme compréhensible : nous avons donc retenu comme schéma de désignation externe une chaine de caractère (limitée à 100 éléments). Par contre, il est nécessaire d'utiliser un mécanisme de désignation interne plus efficace, ce qui pose le problème du choix d'un tel identificateur. En effet la méthode de résolution du nom externe en nom interne pose un problème de consensus, tous les sites connectés devant impérativement fournir une résolution de nom identique. La méthode consistant à utiliser un serveur de nom unique a été écartée car centralisée et donc non tolérante aux pannes de sites. L'approche adoptée consiste à maintenir localement une table de conversion pour tous les groupes auxquels des clients locaux sont abonnés. L'accès d'une station à un groupe auquel elle n'est pas abonnée utilise des primitives différentes n'utilisant que la désignation externe du groupe. Il ne reste plus qu'à assurer la cohérence et

l'unicité des noms internes pour chaque machine abonnée à un groupe. Ces deux propriétés resultent des choix suivants :

- les opérations de création d'un nouveau groupe et d'adhésion à un groupe existant sont clairement séparées, ce sont deux interfaces différentes au niveau d'un client. Dans les deux cas, le site envoie en diffusion des messages précisant le type d'opération désirée à intervalles réguliers durant un certain laps de temps (noté T par la suite). Ces messages impliquent une réponse immédiate d'un site les recevant : dans le cas d'une tentative de création et si ce groupe existe déja localement, un message d'erreur est renvoyé ; s'il s'agit d'une demande d'adhésion, le couple <nom externe, nom interne> est alors renvoyé à la station émettrice. Comme les groupes sont opaques le site fait alors partie du groupe sans qu'aucune autre opération soit nécessaire. Ce mécanisme garantit la cohérence des associations à la condition qu'il n'y ait pas de partition réseau entre un site tentant de créer un groupe existant et les membre de ce groupe.
- un identificateur interne de groupe est codé sur 64 bits ; 32 bits servant à désigner le nom du site créateur, le reste etant un compteur incrémenté localement à chaque création et dont la valeur de base est initialisée de manière aléatoire à l'initialisation du logiciel sur ce site. De cette manière l'unicité des identificateurs internes est assurée avec une très faible probabilité d'erreur, même en cas de redémarrage d'un site. De plus, lors d'une demande de création les sites récepteurs verifient que le nom interne proposé n'est pas déja utilisé par un groupe existant, ce qui provoque une erreur et une réinitialisation du compteur local.

Le cas défavorable est celui lié à une partition du réseau lors d'une création, le facteur critique étant la valeur de T, durée limite qui sépare conceptuellement l'absence de communication due à des pertes de messages ponctuelles de la partition où les sites sont logiquement considérés comme déconnectés. Le choix de T est un compromis entre le risque de création abusive d'un groupe déja existant et le délai nécessaire à la création d'un nouveau groupe. En pratique et dans le cadre d'un réseau local, une dizaine de seconde est un choix sans risque. Remarquons que l'opération d'adhésion à un groupe existant est quasiment instantanée. Enfin le choix d'identificateurs de groupes sur 64 bits et le mécanisme d'adhésion permettent de faire cohabiter des groupes formés au cours d'une partition même lorsque celle−ci prend fin. Seul le cas d'utilisation simultanée d'un identificateur interne pour deux groupes distincts pose alors problème ce qui est fort peu probable car cet identificateur est en partie forgé à l'aide de l'adresse du site créateur rendant les cas de recoupement très improbables (cela nécessite que le site créateur quitte le groupe, redémarre, puis soit isolé au moment où il tente de recréer un autre groupe et enfin que les étiquettes locales sur 32 bits soit par malheur identiques !).

Une fois un site connecté à un groupe, toutes les communications dans le cadre de ce groupe utilisent l'identificateur interne. A la réception d'un message, les couches basses de la couche générique transmettent l'en−tête du message pour un premier traitement (optimisation décrite au III.1.3.3) et la structure de donnée associée au groupe est alors recherchée. Les messages de création ou d'adhésion à un groupe sont traités à ce niveau. Si le groupe n'est pas présent sur le site, le traitement est alors abandonné, sinon le contrôle est

transféré à la routine de réception associée au protocole spécifique utilisé par ce groupe. Ce mécanisme permet d'implanter plusieurs protocoles simultanément.

III.2.2 Protocole de diffusion

L'implantation du protocole de diffusion s'appuie sur les services de la couche générique. En particulier, toute la gestion des données transférées utilise le type Data pour référencer les blocs manipulés et le stockage utilise intensivement des objets de type Buffer. Pour chaque groupe et sur chaque site, plusieurs instances du type Buffer sont utilisées :

- sent_not_delivered est utilisé lors de l'émission de données utilisateur (`MSG_SEND`), celles−ci sont stockées afin de pouvoir les retransmettre si le message a été perdu, les informations associées à chaque Data dans ce tampon sont le nombre de retransmissions déja effectuées et l'heure approximative du premier envoi. Le tampon est trié suivant une clé qui est un compteur d'émission géré localement et incrémenté à chaque envoi de nouvelles données depuis ce site.
- received_not_delivered est présent sur chaque site du groupe et permet de conserver les messages reçus depuis le site émetteur (`MSG_SEND`), mais dont le séquenceur n'a pas délivré l'estampille globale (`MSG_DELIVER`). Les informations associées à chaque Data dans ce tampon sont le nombre de retransmissions déja effectuées et l'heure approximative du premier envoi. Le tampon est trié suivant une clé forgée à l'aide de l'estampille locale et du site émetteur. Le site séquenceur associe aussi à chaque Data les acquittements reçus pour celui−ci en provenance des sites de stockage. Une fois tous les acquittements reçus il peut délivrer une estampille globale pour ce message.
- received_and_waiting est présent sur chaque site du groupe et permet de conserver les messages dont l'estampille globale a été reçue en provenance du séquenceur mais dont l'estampille est trop grande pour qu'ils puissent être délivrés immédiatement. Cela indique que des messages ont été perdus. Le site courant doit donc les demander à un site de stockage, les données sont alors provisoirement entreposées dans ce tampon. Celui−ci est trié par l'estampille globale des Data en attente et on leur associe l'heure de stockage, l'estampille locale et le site émetteur.
- received_and_delivered n'est présent que sur les sites de stockage (séquenceur compris) et permet de conserver les données délivrées localement afin de pouvoir les retransmettre à un site qui ne les a pas reçues lors du premier envoi (`MSG_SEND`). Le tampon est trié par l'estampille globale des données auxquelles on associe l'heure de stockage, l'estampille locale et le site émetteur. La politique d'évacuation des messages peut alors être basée sur un nombre maximum de messages conservés ou sur un critère temporel.

La gestion des délais de garde qui déclenchent le traitement des erreurs dûes aux pertes de messages est implanté sur chaque site par un flot d'exécution indépendant dont le seul rôle est de parcourir régulièrement les différents tampons pour vérifier l'absence de blocage dû à la perte d'un message. En particulier, les messages entreposés dans sent_not_delivered doivent être réémis périodiquement, la période ne devant pas être trop courte pour éviter la

saturation inutile de la bande passante, une valeur trop longue faisant chuter la réactivité du protocole, point particulièrement sensible pour les programmes interactifs. Une valeur de l'ordre de la demi−seconde semble correcte pour la bonne marche du protocole sur un réseau local. La détection d'un message dont l'émission a débuté depuis un délai supérieur à T provoque le déclenchement d'une phase de resynchronisation car cela semble indiquer la panne d'un des sites de stockage, du séquenceur, voire une partition. Enfin sur les sites de stockage ce flot d'exécution verifie à intervalle régulier la bonne marche du séquenceur, et vice versa.

III.2.3 Protocole point−à−point

Un protocole point−à−point fiable a aussi été construit au dessus de la couche générique. Il utilise un algorithme classique basé sur le principe de fenêtre d'acquittement de taille fixe. Il permet de transférer de manière fiable des blocs d'information entre deux processus appartenant à un même groupe. Son but était essentiellement de tester les interfaces de la couche générique pour s'assurer de leur adéquation pour des protocoles d'un type très différent du protocole de diffusion. Cela a aussi permis de tester ces composants logiciels dans un autre contexte, ce qui permet d'éliminer des erreurs résiduelles.

III.3 Place dans le système

Pour des raisons historiques, la mise en œuvre du protocole a été initialement conçue pour s'exécuter comme une tâche serveur au dessus du micro−noyau Mach 3. Le protocole a pu être porté sur plusieurs systèmes Unix et des expérimentations visant à implanter le serveur au sein du micro−noyau Mach ont pu être effectuées. La facilité des portages prouve le bien fondé de la couche d'interface avec le système.

III.3.1 Serveur sur Mach 3.0

L'environnement d'expérimentation du micro−noyau Mach 3.0 utilisé est basé sur la version du système Unix OSF1/MK (la compatibilité a été vérifiée avec la version BSD de Carnegie Mellon University). Dans cette implantation, les fonctions d'Unix sont offertes par un émulateur. Le code du noyau Unix est implanté dans une tâche Mach (donc en mode utilisateur) et les appels systèmes Unix des autres processus sont redirigés vers des ports (canaux de communications Mach) de cette tâche. Le protocole fonctionne lui aussi comme un serveur dans une tâche Mach. La figure suivante illustre cette architecture logicielle :

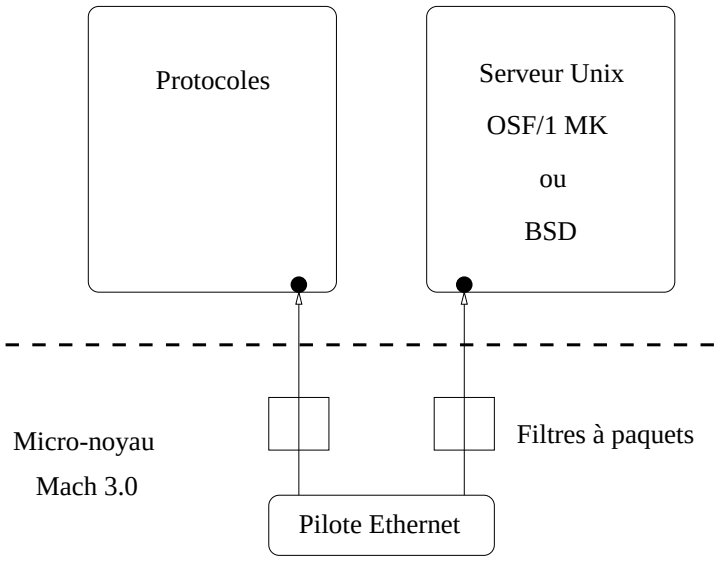


Fig. 3.4 : Architecture dans l'environnement Mach 3.0

L'implantation du protocole au dessus de Mach 3.0 utilise en priorité les interfaces du micro−noyau plutôt que les primitives offertes par l'émulateur Unix OSF/1. Toute la gestion des flots concurrents est donc basée sur les threads fournis par le micro−noyau. De même les communications sont basées sur le filtre à paquets de Mach, inspiré des versions BSD d'Unix.

III.3.1.1 Présentation du micro−noyau Mach 3.0

L'architecture du micro−noyau Mach a été initialement présentée dans [20]. Le principe d'un micro−noyau est de fournir une interface de haut niveau aux ressources matérielles disponibles au sein d'un ordinateur (Entrées/Sorties, processeurs et mémoire) sans toutefois fixer les politiques d'allocation de celles−ci, ce qui est le rôle du système d'exploitation. Le système opératoire de la machine est alors formé du micro−noyau et d'un ou plusieurs serveurs s'exécutant dans le mode non privilégié du processeur et fournissant les interfaces d'un système classique (par exemple UNIX). Les avantages théoriques associés à l'utilisation d'un micro−noyau sont nombreux :

- la nouvelle organisation logicielle du système opératoire permet d'explorer des solutions originales concernant la mise en œuvre de l'ensemble des services offerts aux processus utilisateurs. L'intégration du support des réseaux, un des points faibles des systèmes UNIX, fournit de bon exemples de réalisations originales dans le cadre des systèmes à micro−noyau, par exemple migration du code vers l'espace utilisateur [21], ou implantation de systèmes répartis Guide−2 [31] et QNX [33].
- le confinement du code machine−dépendant au seul micro−noyau, ce qui permet de "nettoyer" l'implantation du système d'exploitation et d'améliorer sa portabilité : en effet seul le portage du micro−noyau est nécessaire pour supporter une nouvelle gamme de machines.
- la minimisation du code s'exécutant dans le mode privilégié du ou des processeurs, ce qui limite d'autant les risques de pannes dûes à une erreur logicielle. L'arrêt

impromptu d'un des serveurs n'est plus fatal car celui−ci peut être redémarré "à chaud" sans avoir a rebooter la machine, exemple KeyKos [35] , VSTa [34].

- les interfaces standard du micro−noyau permettent d'implanter de nouvelles fonctions à coté du système opératoire existant sans avoir à modifier celui−ci. Les plates−formes à micro−noyau sont par là même idéales pour se livrer à des expérimentations dans le domaine des systèmes d'exploitation.

Le micro−noyau Mach fournit cinq types d'abstractions :

- une *tâche* est un ensemble de ressources système, constitué d'un espace d'adressage et d'un ensemble de capacités permettant l'accès à d'autres ressources.
- un *thread* est un flot d'exécution, unité de base d'allocation processeur. Il appartient à une tâche et a accès à toutes les ressources accessibles par les capacités qu'elle possède.
- un *port* est un canal de communication unidirectionnel. Toutes les abstractions offertes par le micro−noyau (à l'exception de l'espace d'adressage) sont référencées par des ports. L'accès à une ressource est conditionné par la possession d'une capacité en lecture ou écriture du port associé à la ressource.
- un *message* est l'élément de base de communication inter−tâches. C'est un ensemble de données, et en particulier il est possible de transmettre des capacités ce qui permet à des tâches distinctes de partager des ressources.
- un *objet mémoire* (memory object) est l'unité de base de gestion de la mémoire virtuelle. La gestion de la pagination associée à un objet mémoire est fournie par un gérant mémoire (memory manager ou *pager*), unité d'exécution traitant les échanges entre mémoire physique et espace disque.

Le micro−noyau Mach a été spécifiquement optimisé pour son implantation sur machines multi−processeur, qu'elles soient à mémoire partagée (optimisation de la gestion de la mémoire commune grâce aux pagers) ou basées sur une communication par messages. En particulier plusieurs flots d'exécution d'une même tâche peuvent s'exécuter concurremment sur des processeurs différents. La figure suivante illustre le modèle d'exécution et de communication offert par le micro−noyau :

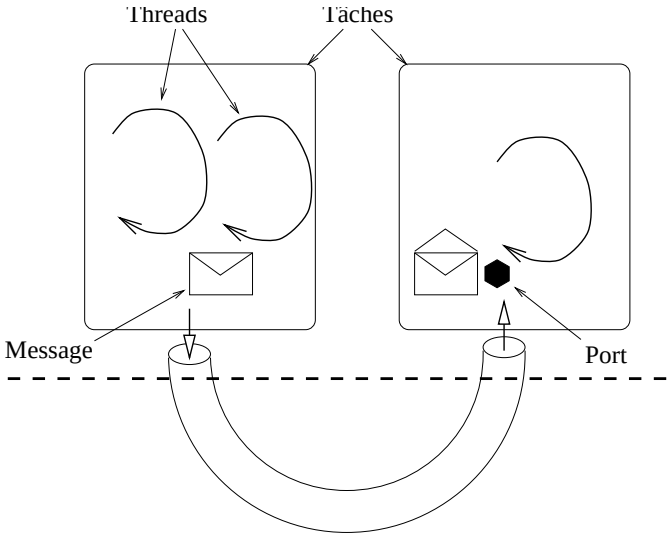


Fig. 3.5 : Tâches, threads, messages et port Mach

III.3.1.2 C−threads et threads noyau

L'environnement de programmation standard fournis avec Mach 3.0 comprend une librairie d'interface aus processus légers fournis par le noyau : les C−threads. Son principe est d'effectuer un multiplexage des flots d'exécution légers fournis au programmeur au dessus des threads du noyau. Le but est de minimiser le nombre d'appels au noyau en remplacant, quand c'est possible les appels à des primitives bloquantes ou de synchronisation et le changement de contexte qui s'ensuit par des appels non bloquants, la librairie se charge alors de redonner la main a un autre C−thread de la même tâche. Les C−threads sont bien sûr implantés à l'aide des threads du noyau Mach, mais pas nécessairement dans un rapport 1 pour 1. La figure suivante illustre ce cas de figure, l'exécution des cinq C−threads etant multiplexée sur seulement deux flots alloués dans le micro−noyau.

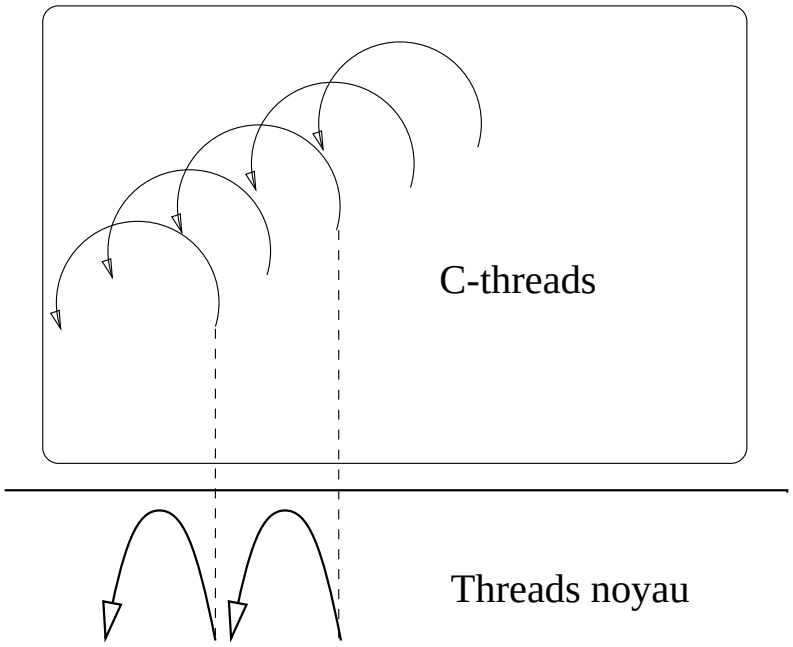


Fig. 3.6 : Multiplexage de C−threads

Nous avons choisi de ne pas utiliser les C−threads pour la mise œuvre du protocole, mais d'implanter nos primitives d'exécution concurrente directement sur les primitives du micro−noyau. En effet, la bonne exécution du protocole est directement liée aux mécanisme d'entrées / sorties associés à l'interface réseau. Peu de flots concurrents sont en pratique nécessaires, à savoir une boucle de traitement des paquets entrant, une boucle de gestion des informations en provenance des clients et le traitement asynchrone des délais de garde. Par contre, la garantie d'avoir un flot d'exécution dédié pour chacun d'entre eux assure par exemple que la boucle de réception des paquets n'est pas bloquée par le traitement asynchrone (peu urgent) dû à l'expiration de délai de garde. Le surcoût en termes de ressources se justifie par un gain de réactivité, spécialement en environnement multiprocesseurs. De plus la librairie de C−threads ne permet pas de donner explicitement la main à un autre flot d'exécution, rendant par là même difficile la réalisation de la primitive ResumeThread de la couche générique, alors que cette primitive est directement disponible souc la forme d'un appel Mach pour les threads noyau. Par contre, le micro−noyau Mach ne fournit pas explicitement de primitives de synchronisation entre flots d'une même tâche et nous avons dû recourir à des instructions de test atomique spécifiques à l'Intel i386 pour implanter les primitives de synchronisation de l'interface générique. Cette lacune se justifie par le fait que ce type de primitive est disponible au niveau de la librairie des C−thread, la mise en œuvre y est d'ailleurs elle aussi basée sur un test−and−set basé sur le jeux d'instruction de chaque processeur cible.

III.3.1.3 Le filtre à paquets

Le filtre à paquets a été conçu comme un moyen pratique d'implémenter les fonctionnalité des protocoles hors du noyau du système d'exploitation. Son principe de fonctionnement, décrit initialement dans [13], est le suivant :

- un appel système permet de déposer un fitre sur une interface réseau. Le filtre est écrit sous la forme d'un petit programme qui à partir du contenu d'un paquet et d'opérations booléennes, retourne l'acceptation ou le rejet du paquet sur ce filtre.
- à la réception de chaque paquet, le noyau interprète successivement tous les filtres associés à l'interface dans ce contexte. Si un filtre accepte le paquet, celui−ci est transmis au programme qui a posé le filtre.

L'avantage de ce mécanisme est sa très grande souplesse dans les critères de choix des paquets. Il est ainsi possible d'implanter en mode utilisateur le traitement de plusieurs piles de protocoles sous la forme de plusieurs programmes ayant chacun déposé un filtre spécifique. L'inconvénient est le temps nécessaire à l'interprétation de chaque filtre sur un paquet jusqu'à trouver le destinataire, celui−ci pouvant atteindre la milliseconde. Le coût lié à la remontée des données depuis le noyau vers l'espace utilisateur peut être minimisé en regroupant les packets dans le noyau et en les transmettant par lots. Dans la plupart des UNIX d'inspiration BSD, le démon rarpd(8) qui offre le service de résolution inverse d'adresses (adressage physique vers adressage logique) est implanté à l'aide d'un filtre à paquets. Le filtre suivant est celui mis en place par le protocole pour sélectionner uniquement ses propres paquets :

```
#define SIZE_PACKET_FILTER 3
filter_t filter[SIZE_PACKET_FILTER] = {
      NETF_PUSHHDR+6, /* Mot type du paquet Ethernet */
       NETF_PUSHLIT | NETF_EQ,    /* est−t−il egal */
         0xFABC /* Constante réservée au protocole */
};
```

Le premier élément commande à l'interpréteur d'empiler le sixième mot de l'en−tête du paquet (celui qui contient le champ type), le deuxième fait empiler la valeur littérale contenue dans l'élément suivant et applique le test d'égalité, le troisième contient un identificateur de type associé au protocole.

Le principal problème lié à la mise en œuvre sur le micro−noyau Mach est dùe à la gestion des paquets au sein même du micro−noyau. Le noyau alloue un faible nombre de pages pour la gestion des paquets arrivant depuis l'interface réseau. Les pages sont réservées en priorité aux paquets qui ne sont pas émis en diffusion, ces derniers étant considérés comme moins prioritaires. Lors de l'exécution du protocole, il n'y a pas de problème sous faible charge, mais dès que le débit dépasses une vingtaine de messages diffusés par seconde, le taux de perte des paquets émis en diffusion (MSG_SEND et MSG_DELIVER) augmente subitement jusqu'à provoquer une famine complète. A ce point seuls les messages de contrôle et de gestion des erreurs sont effectivement délivrées, et le protocole s'écroule ! Le seul remède consiste à modifier le micro−noyau pour ne pas discriminer les messages en diffusion, ce qui peut être fait en modifiant le source de la fonction ethernet_priority() du micro−noyau. Un petit programme permettant de patcher directement les binaires du fichier de boot du noyau Mach pour l'architecture i386 a aussi été mis au point afin de limiter les recompilations.

Pour conclure sur l'implantation du protocole au dessus du micro−noyau Mach 3.0, on peut remarquer que cette plate−forme offre la possibilité d'implanter des protocoles dans

l'espace utilisateur à l'aide d'un jeu de primitives bien standardisées. Contrairement aux systèmes UNIX les interfaces offertes pour le développement à bas niveau sont simples et complètement détaillées, et le portage du protocole depuis Mach 3.0 de l'OSF vers la version originelle de CMU n'a demandé aucune modification du code source, seule une recompilation a été nécessaire. Par contre, l'implantation des primitives est parfois surprenante, par exemple la gestion des messages en diffusion ou l'absence de fonction de synchronisation. La sous−section suivante traitant des essais d'implantation dans le noyau Mach nous donne l'occasion de détailler plus en avant le traitement effectif subi par les paquets en provenance de l'interface réseau.

III.3.2 Expérimentations dans le noyau Mach

L'un des principaux inconvénients de l'implantation d'un protocole hors du noyau du système d'exploitation est le temps de la latence supplémentaire induit par la traversée des données entre espace utilisateur et espace noyau. Considérons par exemple le traitement d'un message de contrôle tel que décrit par la figure suivante :

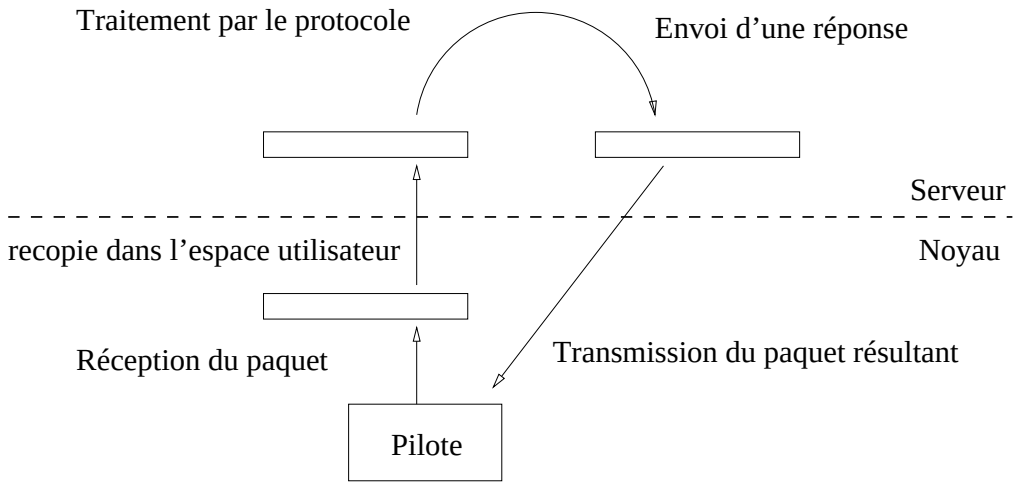


Fig. 3.7 : Mouvements de données entre espace utilisateur et noyau

Avant même le traitement du paquet par le protocole, une recopie complète de son contenu aura eu lieu afin de le mettre à disposition du processus implantant le protocole. On notera qu'il est théoriquement possible de recevoir directement le paquet dans l'espace utilisateur mais il est alors impossible que plusieurs processus distincts puissent recevoir concurremment des paquets. En effet le démultiplexage doit s'appuyer sur le contenu même du paquet qui est indisponible lors de la réception de celui−ci. Une autre méthode consiste à reserver une page de mémoire physique pour chaque paquet entrant (ce que fait Mach 3.0), ce qui permet ensuite de coupler celle−ci dans l'espace virtuel du processus récepteur en ne modifiant que les tables de pages du processus. Cela conduit toutefois à une perte importante de mémoire, vue la différence de taille entre un paquet Ethernet (56 à 1514 octets) et une page de mémoire physique (4096 octets ou plus).

Le coût induit par le transfert des données et de l'exécution vers l'espace utilisateur nous ont incité à transférer l'implantation du protocole au sein du micro−noyau Mach 3.0.

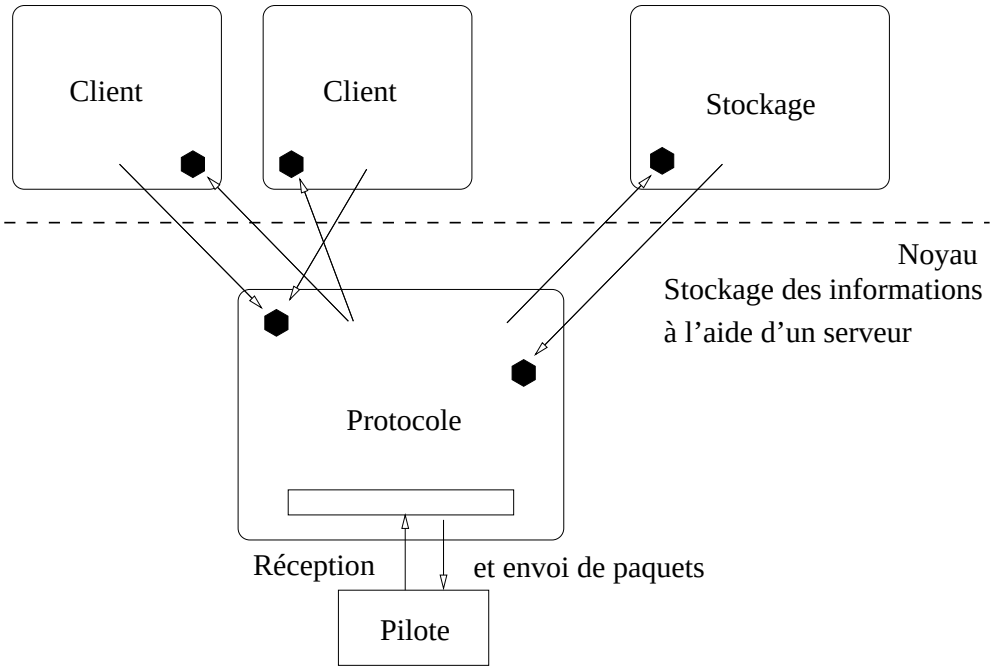


Fig. 3.8 : Implantation du protocole dans le noyau

La méthode utilisée a consisté à intégrer le code du protocole dans l'arborescence des sources du micro−noyau Mach 3.0, et à installer le protocole en déroutant si nécessaire le flot d'exécution traitant un paquet en réception pour exécuter notre protocole. L'émission d'un paquet est réalisée en appelant directement la routine d'écriture du pilote de la carte Ethernet. L'allocation mémoire utilise les routines du noyau *kalloc* et *kfree*. Néanmoins ces routines ne sont pas disponible lors du traitement d'interruption, ce qui est le cas lors de la réception d'un paquet ; les complications qui s'ensuivent sont détaillées dans la sous−section III.3.2.2.

La communication entre les clients du protocole et ce dernier se fait par le biais d'envoi de messages Mach. Un client se connecte au protocole à l'aide d'un filtre a paquets spécial (il a fallu pour cela modifier l'appel système correspondant dans Mach) et fournit à cette occasion un port de réception des messages. De plus pour éviter le stockage de grande quantités de données dans le noyau, en particulier pour les sites de stockage, il peut être intéressant d'utiliser un démon se chargeant du stockage. Il s'enregistre de manière analogue à un client, et offre un service de stockage en espace paginé. Le noyau lui transfère les données une fois délivrées aux clients et peut être amené à les demander en cas de perte par un autre site. Le démon se charge de l'élimination des messages périmés, en fonction de la politique de suppression retenue. Ce mécanisme offre beaucoup de souplesse mais n'a pas été implanté.

III.3.2.1 Primitives du noyau

L'implantation du protocole dans le noyau nous a, dans un premier temps paru aisée : en effet un bon nombre des routines de la couche générique étaient déja présentes sous une forme similaire au sein du micro−noyau. Pour la gestion mémoire, les routines `kalloc` et `kfree` fournissent les primitives usuelles. La gestions des threads s'appuie sur les interfaces suivantes :

- `kernel_thread` permet de créer un nouveau thread dans le noyau.
- deux appels consécutifs à `thread_will−wait_with_timeout` et `thread_block` permettent de bloquer un thread pour une période de temps donnée.
- la routine `thread_block` réalise un appel direct à l'ordonnanceur du micro−noyau
- les primitives `simple_lock_init`, `simple_lock_try` et `simple_unlock` fournissent des primitives de base de synchronisation non bloquantes usuellement appelés `spin_lock`. En combinant l'utilisation de `simple_lock_try` avec un appel à `thread_block` il est possible de fournir les primitives de sections critiques décrites au III.1.3.1.

L'implantation du micro−noyau Mach 3.0 utilise de manière intensive le mécanisme de *continuation*. Le principe consiste à économiser des ressources mémoire en supprimant la pile noyau d'un flot d'exécution bloqué dans le noyau. Lors de l'arrêt de celui−ci, il indique une routine qui sera le point de reprise de l'exécution lors de son réveil. Dans l'intervalle la pile peut être désallouée. Ce gain technique complique malheureusement la compréhension du code, effet déstructurant déjà reproché aux branchements non locaux (`goto`) utilisés de manière anarchique dans les premiers langages de programmation, auquel s'ajoute l'effet temporel lié à l'arrêt momentané du processus.

III.3.2.2 Gestion de la mémoire et accès au réseau

Une des principales difficultés de l'implantation du protocole dans le micro−noyau Mach est liée aux restrictions faites sur l'allocation de mémoire. En effet la fonction d'allocation mémoire interne au noyau kalloc ne peut être utilisée lors d'un traitement d'interruption. Par conséquent si une allocation doit être effectuée lors de l'arrivée d'un paquet il faut reporter cette allocation, ce qui se fait généralement en effectuant un changement de contexte vers un autre flot d'exécution qui s'exécute à un niveau moins privilégié du processeur, mais bénéficiant des ressources nécessaires pour traiter l'allocation. La figure suivante décrit ce mécanisme lors de l'arrivée d'un paquet. Si aucune allocation de mémoire n'est nécessaire, la routine d'interruption peut gérer tout le traitement, y compris le test du paquet pour les différents filtres, et l'envoi au destinataire. Toutefois une allocation peut être nécessaire, par exemple s'il n'y a plus de page libre pour la réception des paquets suivants ou si plusieurs filtres acceptent le paquet. Dans ce cas le contrôle est donné à un thread noyau dont la tâche exclusive est de traiter les paquets arrivant et d'allouer la mémoire nécessaire.

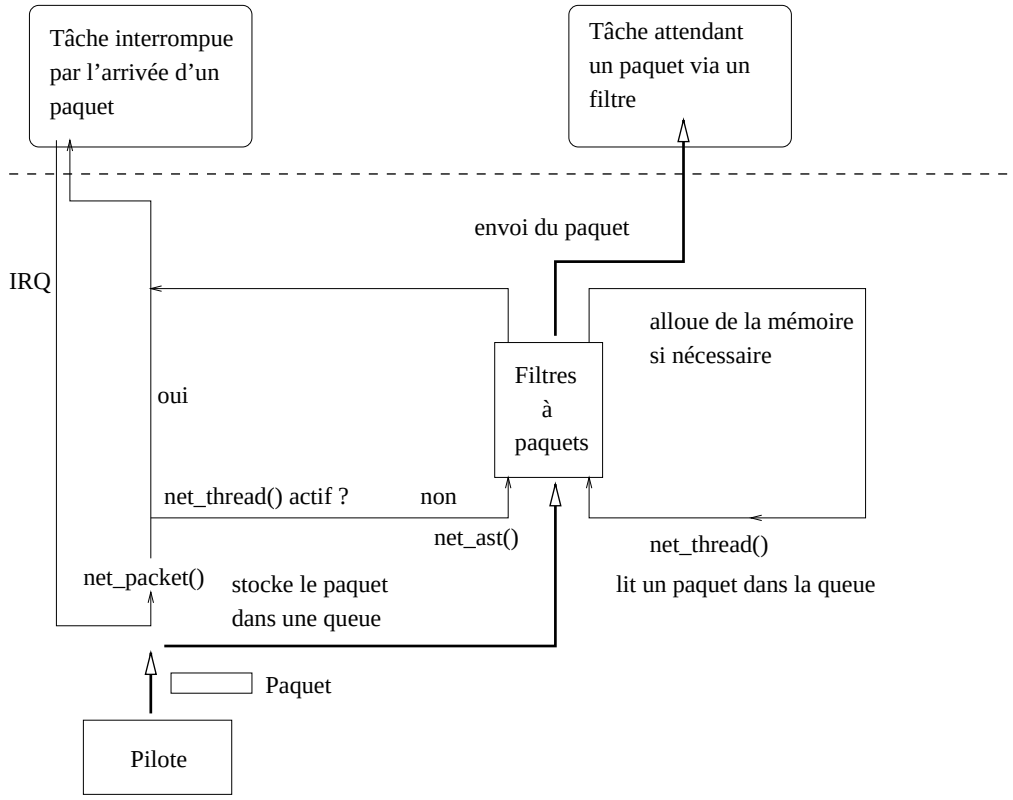


Fig. 3.9 : Traitement d'un paquet en réception dans Mach 3.0

Ce mécanisme est optimisé pour le fonctionnement sur les architectures multiprocesseur et il est alors possible que le traitement d'interruption soit pris en charge par un processeur et que net_thread soit en cours d'exécution sur un second processeur. Dans ce cas la routine d'interruption se contente de stocker le paquet dans une queue, net_thread se chargera du traitement, et il n'est pas nécessaire d'effectuer de changement de contexte.

La réception de paquets par le protocole nécessite donc un changement de contexte vers un thread noyau spécifique au protocole : en effet tout paquet arrivant est susceptible de provoquer une allocation mémoire. Le noyau offre des routines de synchronisation par condition qui sont utilisées pour réveiller le thread de réception des paquets.

Pour conclure sur l'implantation dans le micro−noyau Mach, nous retiendrons que si l'ajout de nouvelles fonctionnalités semble a priori facilité par la disponibilité de primitives de haut niveau, la compréhension des mécanismes mis en œuvre et de leurs limites n'est pas aisée. De plus si l'interface du micro−noyau vers les tâches fonctionnnant en mode utilisateur est bien établie, ce c'est pas le cas pour les primitives internes. Le prototype implanté dans le micro−noyau de l'OSF a dû subir plusieurs modifications pour l'utilisation de la version standard des sources distribués par CMU et cette version n'a jamais atteint une stabilité suffisante pour se livrer à des essais.

III.3.3 Implantation sur UNIX

La décision de porter le protocole sous UNIX a été motivée par plusieurs facteurs :

- L'absence de licence source pour les machines du type DecStation et Sun, nous privait des possibilités d'expérimentation basée sur Mach sur ces architectures. Par là même nous étions limités aux machines du type IBM−PC, moins performantes a priori en terme d'entrées / sorties, et présentes en nombre restreint au sein du laboratoire.
- L'abandon progressif de Mach pour les expérimentations dans le laboratoire, la fin du projet Guide−2 et le transfert industriel de celui−ci vers la plateforme DPX−20 sous AIX, l'UNIX d'IBM, montrait clairement les nouveaux choix et l'adoption d'un environnement standardisé.
- La disponibilité de clones UNIX, dont l'accès au code source ne nécessite pas de licence, permet désormais d'effectuer les expérimentations au cœur même du système, privilège autrefois réservé aux seules Universités (américaines !) enregistrées auprès des constructeurs. Citons les exemples de LINUX, BSD−4.4 et autres clones issus de la branche BSD, parmi les systèmes désormais disponibles en source.

Toutefois le succès de l'implantation sur un nouveau système dépend de la disponibilité des primitives de bas niveau sur lesquelles repose toute l'architecture logicielle du protocole. Si l'utilisation de threads n'est pas possible, la réalisation d'une bibliothèque de coroutines au sein même du processus UNIX permet de s'en dispenser. Par contre l'accès aux primitives d'envoi et de réception de paquets Ethernet est essentielle. Or même les systèmes récents ou les normes POSIX ne définissent pas de fonction standard pour l'accès aux couches basses des protocoles. Seul le mécanisme de `socket`, hérité de la branche BSD d'UNIX fournit un mécanisme d'accès aux protocoles implantés dans le noyau, se limitant en général aux protocoles IP. Nous détaillons dans les deux sous−sections suivantes la mise en œuvre d'un module portable de coroutines sous UNIX et les méthodes d'accès aux pilotes Ethernet offerte par les diverses versions du système.

III.3.3.1 Réalisation de coroutines

La réalisation de l'environnement de développement, basé sur un simulateur de réseau Ethernet, et détaillé au chapitre IV, a nécessité l'implantation d'une librairie de coroutines. Celle−ci a servi de base pour l'implantation du protocole sous UNIX. Les coroutines, à la différence des threads ne sont pas des entités reconnues par le noyau. Les routines d'ordonnancements sont donc gérées uniquement par le processus utilisateur. L'avantage est qu'il est possible de construire des telles primitives de manière portable à l'aide des routines `setjmp` et `longjmp` de la librairie C. La fonction `setjmp` permet de sauvegarder l'état courant du flot d'exécution du processus dans une structure de type `jmp_buf`, qui est selon la norme POSIX un tableau d'entiers où seront recopiées les valeurs des registres du processeur. Cette fonction ne réalise pas un point de sauvegarde complet du processus, seul l'état courant du processeur est enregistré. Au retour d'une sauvegarde, cette fonction renvoie zero. La fonction `longjmp` est la duale qui permet de restaurer le flot d'exécution stocké dans un `jmp_buf`. Lors de son appel l'état du processeur est restauré et le contrôle est rendu au niveau du retour du `setjmp` ayant provoqué la sauvegarde, mais avec une

valeur de retour non nulle, celle−ci étant passée comme deuxième argument à longjmp. Ce mécanisme permet d'implanter aisément un changement de contexte entre coroutines à l'aide d'un appel a `setjmp` pour sauvegarder l'état du flot courant et un appel à `longjmp` pour redémarrer une autre coroutine, tel qu'illustré par la figure suivante.

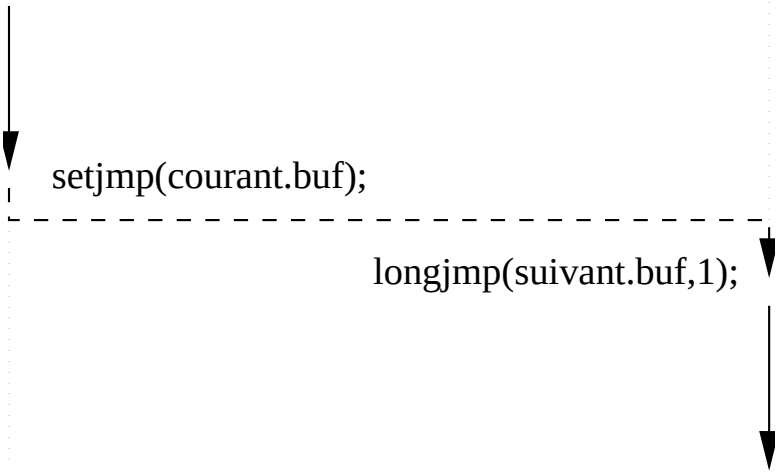


Fig. 3.10 : Changement de contexte entre coroutines

Le mécanisme de création d'un nouveau flot n'est guère plus compliqué, mais requiert un minimum d'informations sur le contenu d'une structure jmp_buf. En effet, on alloue une nouvelle pile pour la coroutine à créer, et sa mise en place nécessite de stocker le nouveau pointeur de pile dans le jmp_buf correspondant. L'indice dans le tableau correspondant est relativement aisé et sa recherche peut être automatisée.

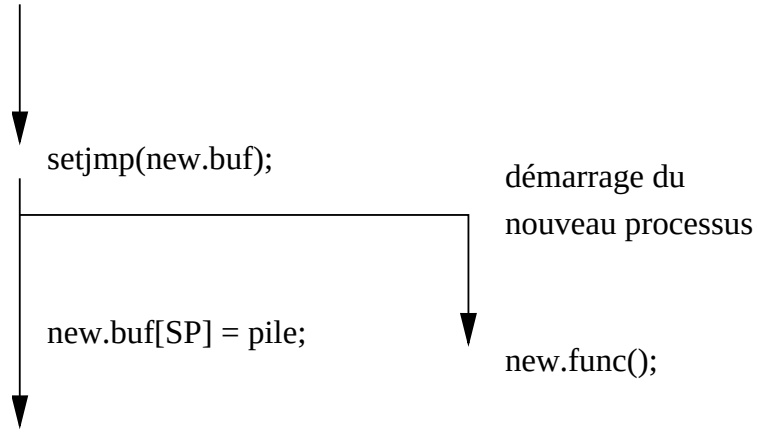


Fig. 3.11 : Création d'une nouvelle coroutine

Dans le principe ce mécanisme est indépendant du système d'exploitation, et a été utilisé avec succès sur de nombreuses plates−formes UNIX (LINUX, MIPS/BSD, SunOs, OSF/1 et BSD/Mach) et même sur MS−DOS. Par contre, certaines implantations des routines setjmp et longjmp vérifient la cohérence des piles lors de chaque appel, nécessitant alors une

réécriture en assembleur des deux routines ; ce fut le cas pour Ultrix (processeur MIPS) et AIX (processeur RS/6000). La librairie gère les appels bloquants au noyau UNIX à l'aide de l'appel système `select` qui permet de détecter la réception d'information sur un descripteur de fichier. Une coroutine effectuant une lecture qui risque de s'avérer bloquante est alors stoppée au profit des autres jusqu'à l'arrivée de données sur le descripteur associé à la lecture. Plusieurs bibliothèques de threads basées sur ce principe et respectant la norme POSIX associée (1004.3) sont disponibles, mais leur emploi pour le protocole a été abandonné. En effet, les besoins de l'implantation de la couche générique sont simples, le but est seulement de mieux structurer le logiciel et de gérer plus aisément les appels bloquants, et dans ce cas, l'emploi de threads POSIX est un peu disproportionné avec le but à atteindre. Il s'avère aussi que les manipulations nécessaires pour accéder au pilote Ethernet n'aient pas été prise en compte par les réalisateurs de la bibliothèque de threads : ce cas s'est produit sous Ultrix où l'appel système nécessaire pour lire des paquets bloquait l'ensemble du processus et non le seul thread incriminé.

III.3.3.2 Accès au pilote Ethernet

Le principal obstacle au portage du protocole sur un nouveau système est la nécessité de pouvoir recevoir et émettre des trames Ethernet depuis un processus utilisateur. Ce problème se pose de manière aigüe dans un environnement UNIX : en effet, l'accès à des ressources physiques de la machine est limité aux interfaces que fournit le système d'exploitation. Or le seul standard d'accès aux primitives du réseau passe par le mécanisme de socket, qui permet d'unifier le mode d'accès aux protocoles disponibles, ceux−ci etant mis en œuvre dans le noyau. Rien, a priori, ne permet donc de certifier que le protocole pourra être porté sur un système UNIX particulier, car les méthodes d'accès au périphérique Ethernet ne sont pas normalisées. Toutefois les quatres systèmes UNIX que nous avons retenus comme plate−forme d'essai, à savoir Ultrix sur DecStation, Linux sur IBM−PC, SunOS pour les stations Sun 4 et AIX 3 sur RS/6000 offrent tous les primitives nécessaires à la capture et à l'envoi de paquets.

Les interfaces d'accès aux pilotes Ethernet que nous avons rencontrées sont basées sur les mécanismes suivants.

- Le filtre à paquets, déjà présenté au III.3.1.3 fournit un mécanisme perfectionné de sélection des paquets. Il peut soit constituer un appel système spécifique permettant l'accès au périphérique, c'est le cas sous Ultrix, ou s'interposer comme un module STREAMS de sélection et de démultiplexage comme sous SunOs.
- Le pilote peut fournir une interface visible depuis le système de fichiers, les opérations de lecture et d'écriture correspondent alors aux routines d'émission et de réception du pilote associé. Le mécanisme de sélection des paquets entrant se fait à l'aide d'un `ioctl` et se borne alors à retenir les paquets dont le champ type correspond à la valeur fournie en argument.
- Le mécanisme de STREAMS permet de construire de manière modulaire des services aux sein d'un noyau UNIX, sous la forme de modules qui peuvent être mis en relation et même chargés dynamiquement. Cela permet de construire des protocoles ou de nouveaux mécanismes dans un noyau existant et de les rendre accessibles aux

applications. Cette interface est le mécanisme utilisé pour l'accès au pilote Ethernet sous SunOs.

- Enfin le mécanisme bien connu des sockets, hérité des versions BSD d'UNIX, peut être étendu pour offrir l'accès au périphérique Ethernet. L'accès au pilote est juste enregistré comme un protocole spécifique implanté dans le noyau.

La figure ci−dessous résume les différentes méthodes d'accès utilisées pour chaque système rencontré.

	Filtre à paquets	Fichier spécial	STREAMS	Socket
Ultrix	X			
Linux				X
SunOs	X		X	
AIX		X		

Fig. 3.12 : Méthodes d'accès spécifiques à chaque système

Pour des raisons de sécurité, l'accès au pilote Ethernet doit être restreint pour éviter les possibilité de piratage des machines situées sur le même réseau. En règle générale, seul un programme s'exécutant avec les droits du superviseur (root) a accès au périphérique. Certains systèmes offrent plus de souplesse : par exemple Ultrix permet de configurer par logiciel les limites d'accès au filtre à paquets. Par contre, l'installation standard d'AIX offre un grave défaut de sécurité à ce sujet, en effet le pilote est accessible en lecture et écriture par tout utilisateur et l'interface d'accès ne contrôle pas les trames émises sur le réseau, en particulier le champ émetteur des paquets transmis. Il est ainsi possible de déjouer tous les protocoles dont la sécurité est basée sur l'adresse émettrice des trames.

Un des reproches faits à l'implantation des protocoles en espace utilisateur est le coût lié au transfert des données depuis le noyau vers l'espace utilisateur. Pour tenter de réduire l'impact des changements de contextes nécessaires lors du traitement de chaque paquet entrant sur une machine chargée, plusieurs systèmes proposent de "bufferiser" les données reçues au sein du noyau et de les transférer en bloc lorsque le tampon est plein ou après un délai de garde. De cette manière, la charge processeur nécessaire pour traiter un fort trafic est minimisée au détriment d'une légère perte de réactivité du protocole sous faible charge.

III.4 Conclusion

Avant même l'étude des résultats obtenus au chapitre V, nous pouvons conclure que le problème de la conception de protocoles en mode utilisateur peut être résolu par l'utilisation d'un faible nombre de primitives de bas niveau. En réduisant les dépendances de l'ensemble du protocole envers le système d'exploitation sous−jacent à quelques fonctions simples et à la sémantique clairement définie, il est possible d'implanter des protocoles à bas niveau tout en garantissant la portabilité. Ces fonctions ne font pas partie de celles documentées par exemple par la norme Posix, mais toutes sont disponibles, avec toutefois des interfaces différentes sur chacun des systèmes utilisés, à savoir au sein d'une tâche Mach, dans le noyau Mach et dans diverses versions d'Unix (AIX, OSF/1, SunOs, Ultrix et Linux).

L'intérêt d'une couche d'interface fournissant toutes les primitives usuelles nécessaires à la construction de protocoles et construite au dessus de ces interfaces systèmes a pu être observé. Elle permet de faciliter la conception de divers protocoles tout en localisant les dépendances système hors de la logique du protocole, celui−ci étant décrit à l'aide de primitives de haut niveau où l'essentiel des opérations telles que la gestion de la mémoire, la fragmentation et le réassemblage, et l'empaquetage et le dépaquetage des données sont effectués de manière transparente. Le chapitre suivant présente les outils de développement mis au point dans le but de simplifier l'implantation et le test de nouveaux protocoles.

Chapitre IV

Méthode de développement

Dans ce chapitre nous abordons les problèmes liés au développement du protocole. En l'absence d'outils spécifiques un émulateur de réseau Ethernet a été développé dans le cadre de ce travail.

IV.1 Méthode de mise au point

La mise au point de toute application répartie est toujours délicate en raison de l'absence d'état global. Les protocoles de communication ne font pas exception à la règle. Nous énumérons les principaux problèmes.

- L'absence d'état global ne permet pas de superviser l'exécution du protocole lors d'une phase de tests en environnement distribué.
- Il est difficile de recréer artificiellement des conditions d'exécutions anormales telles qu'un réseau saturé ou des stations chargées.
- Enfin, lorsque l'implantation du protocole a pu être mise en défaut il est presque toujours impossible de recréer la situation ayant abouti à l'erreur. Dans ces conditions il est difficile de valider les corrections effectuées.

Dans ces conditions, il n'est pas a priori possible de suivre un cycle de développement classique Codage/Correction−Compilation−Tests/Validation. Les techniques classiques de déboguage d'applications réparties s'appuient alors généralement sur l'enregistrement des paramètres d'une exécution. Elle sera par la suite rejouée pour en analyser le comportement de manière déterministe sous le contrôle d'une application (le débogeur) centralisant les résultats collectés sur les différents sites. Notre approche consistant à construire le protocole directement sur Ethernet sans utiliser de couche intermédiaire nous prive des outils de mesure usuels, et seule la capture de trames depuis une station espion reste possible pour enregistrer le déroulement effectif d'une phase du protocole. L'analyse après coup du comportement réel du protocole est alors possible, mais il est impossible de rejouer une exécution, trop d'éléments extérieurs restant incontrôlables (charge des machines, influence des autres stations, etc.).

Très tôt, l'utilité d'un outil d'émulation du réseau et des interfaces systèmes utilisées s'est donc fait sentir. L'objectif est de contrôler complètement la séquence d'exécution du protocole en s'affranchissant des contraintes liées à la multiplication des machines mises en œuvre et des aléas liés au non−déterminisme de l'exécution répartie.

Lors de la mise en œuvre du protocole de diffusion d'Amœba, le protocole a été développé sur une machine multiprocesseurs à mémoire commune ; dans cet environnement, le réseau

Ethernet a été émulé à l'aide de mémoire partagée, le protocole s'exécutant simultanément sur chaque processeur[8].

IV.2 Un émulateur de réseau

Un tel outil a été mis au point dès le début de la phase d'implantation, une fois les spécifications de la couche d'interface système stabilisées. Le principe consiste à fournir au sein d'un processus Unix une bibliothèque d'émulation du réseau Ethernet et des interfaces d'envoi et de réception de messages disponibles.

IV.2.1 Conception

Pour simplifier encore la tâche de mise au point, nous avons opté pour une simulation sous la forme d'un seul processus, l'ensemble d'une exécution du protocole peut alors être tracée à l'aide d'un débogeur. Dans ce cas, tous les serveurs doivent partager le même espace d'adressage, même si, conceptuellement ils sont situés sur des sites différents donc dans des espaces d'adressage distincts. En pratique, il a donc été nécessaire de fournir une émulation pour les concepts suivants.

- Sites : sur chaque site s'exécute une copie du protocole. On doit donc faire cohabiter au sein de l'espace d'adressage du processus Unix toutes les copies du protocole nécessaires à l'émulation des différentes stations. Le mécanisme fonctionne par l'enregistrement des variables globales associées à l'exécution du protocole ; les processus légers de toutes les stations sont ordonnancés de manière globale et lors d'un changement de site, les variables globales du site courant sont sauvegardées et remplacées par celles de la nouvelle station. On pourrait craindre une dégradation nette des temps de simulation du fait des recopies mémoire nécessaires, mais en pratique les objets déplacés sont de petite taille (pointeurs, entiers) et le temps de changement de site n'est guère plus élevé que celui d'un changement de processus léger et certainement plus rapide qu'un changement de processus Unix.
- Réseau : le réseau est émulé en fournissant les mêmes primitives d'initialisation, d'envoi et de réception de paquets ; le principe de fonctionnement des couches basses du système et du réseau sous−jacent correspondent au stockage des paquets dans des tampons. L'émulation du réseau Ethernet simule le protocole d'accès non déterministe CSMA/CD et la bande passante de 10 Mbits/s en contrôlant la recopie des paquets entre tampons d'émission et de réception des différentes station simulées. L'utilisateur peut aussi régler le niveau de perte de paquets pour simuler l'engorgement de la fibre ou des stations, voire une installation défectueuse. La version actuelle se contente d'émuler un seul segment Ethernet mais il serait aisé d'étendre la modélisation à d'autres types de réseau et à des topologies plus complexes.
- Processus légers : ils sont fournis de la même manière que pour l'implantation standard du protocole sur Unix (c.à.d. à l'aide des primitives setjmp / longjmp) ; seul l'ordonnanceur a du être modifié pour prendre en compte le traitement particulier des variables globales lorsque l'on change de site.

- Temps : l'émulateur n'essaie pas de fournir un temps d'exécution directement corrélé avec la durée de la simulation. Le principe de fonctionnement est une boucle d'ordonnancement traitant les coroutines émulant le protocole, celle émulant le réseau et le traitement des événements prédéfinis. Cette boucle appelle aussi les routines d'interface avec l'utilisateur. Le temps interne est proportionnel au nombre de passages dans la boucle, cela permet de garantir le déterminisme de la simulation indépendamment des conditions d'exécution de celle−ci.

La figure suivante illustre la nouvelle architecture logicielle dans cette configuration.

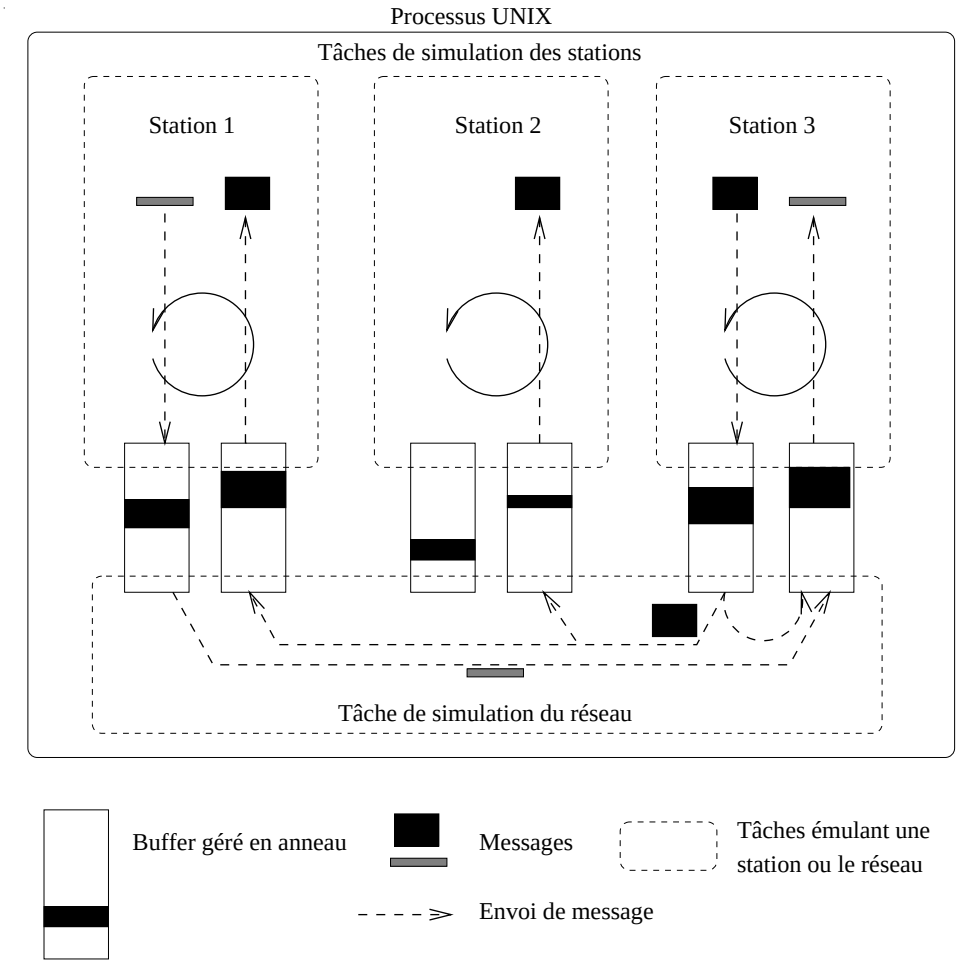


Fig. 4.1 : Architecture de l'émulateur

IV.2.2 Configuration et interface de l'émulateur

Une session de test du protocole consiste en une recompilation du protocole spécifique, au cours de laquelle les objets formant le protocole sont liés à la bibliothèque d'émulation pour obtenir un processus Unix. Lors de son lancement, il y a analyse d'un script de configuration de la session permettant de fixer les paramètres suivant de l'émulation :

- le nombre de stations maximum mis en jeu lors de cette session.
- la valeur d'initialisation du générateur de nombres aléatoires.
- l'heure en temps absolu interne du démarrage ou de l'arrêt d'une station.
- l'utilisation ou non de l'interface graphique basée sur Tcl/Tk.

Voici un exemple de fichier de configuration :

```
 FICHIER DE SIMULATION
      TEST DU PROTOCOLE :
  faible charge, arrêt d'une station de stockage et du séquen-
ceur
  Sélection d'une base du générateur aléatoire
  random 120
  stations 6
  notcl
  Départ des machines
  1 : boot 1  100 : boot 2 10000 : boot 3 4 5 6
  Arrêt du site 1
  70000 : stop 1
  Arrêt du site 2
  120000 : stop 2
  Arrêt des machines
  600000 : stop 3 4 5 6 Arrêt des stations
  600100 : end
```

Le scénario décrit une simulation avec six stations, les sites 1 et 2 qui démarrent en avance seront respectivement séquenceur et site de stockage. A l'instant 70 000 le séquenceur est stoppé, ce qui doit théoriquement provoquer une phase de synchronisation, puis le site 2 est lui aussi stoppé, alors que cette phase n'est pas encore achevée. Enfin à l'instant 600 000 toutes les machines restantes sont arrêtées puis la simulation s'achève.

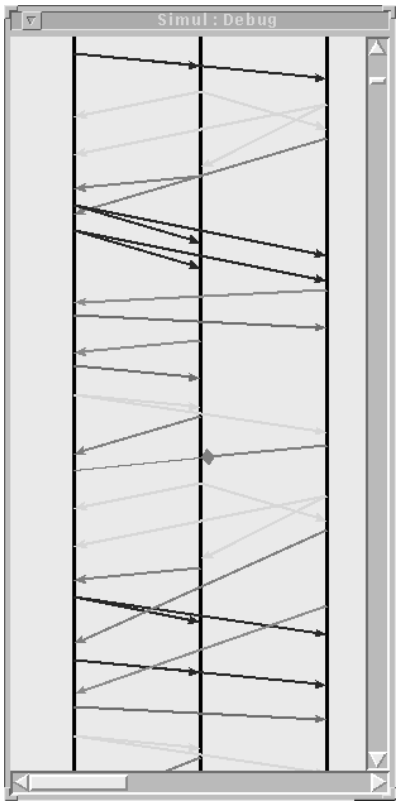
L'émulateur propose deux types d'interfaces disponibles en cours d'exécution. Une interface en mode texte permet un contrôle des fonctionnalités de bas niveau, à savoir :

- émulation du réseau : il est possible d'afficher tous les paquets transmis ou tous ceux atteignant une station de manière interactive. Il est aussi possible de consulter un historique des paquets reçus ou émis depuis une station.
- processus légers et sites : des fonctions permettent de visualiser leur état, par exemple quels sont les processus bloqués, ou le nombre de messages émis et reçus sur un site.
- mémoire : il est possible de déterminer quels sont les blocs de mémoire alloués sur chaque site, et l'emplacement dans le code où cette allocation a eu lieu.

Une interface graphique est aussi proposée. Sa mise en œuvre est basée sur un interpreteur Tcl/Tk appelé directement depuis les diverses couches du protocole. Cela permet d'observer l'évolution de plusieurs paramètres du protocole.

- Une fenêtre permet de sélectionner les vues offrant un intérêt à l'utilisateur. Il est aussi possible de geler l'émulation pour analyser l'état courant du protocole.

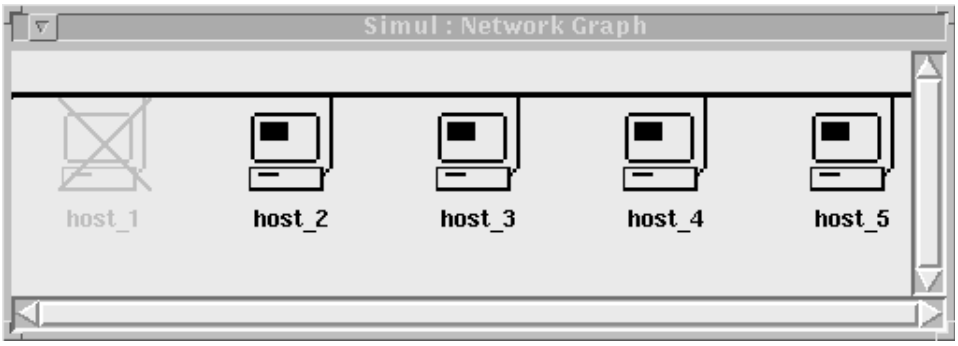
- La couche de simulation d'Ethernet exporte une vue permettant d'analyser les envois et réceptions de paquets. Le contrôle absolu de toute l'exécution offert par l'émulateur permet de donner une représentation complète et non ambiguë des échanges sur le médium. Les paquets émis mais non reçus sont aussi représentés, ce qui permet d'interpréter aisément le comportement du protocole en cas d'erreur. La représentation se fait sous la forme du schéma temporel usuel en l'algorithmique répartie.



La figure précédente illustre une phase du protocole avec trois stations, le site 1 à gauche jouant le rôle du séquenceur, les deux autres étant sites de stockage. A mi−hauteur on peut remarquer la perte d'un acquittement de 3 vers 1 (message sans flèche terminale et signalé par un losange en son milieu). A chaque type de message est associé une couleur spécifique.

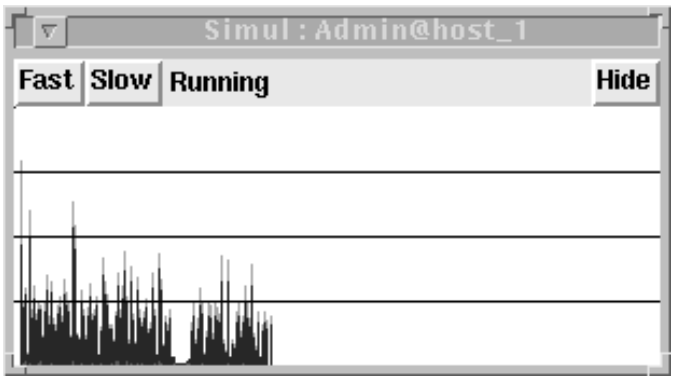
- Une fenêtre fournit une vue de l'état courant des stations. Dans l'état actuel la seule information utile concerne l'arrêt ou la bonne marche des stations. Elle se justifierait

vraiment par l'extension de l'émulateur à d'autres types de réseaux ayant une topologie complexe.



- Pour chaque groupe et pour chaque station, il est possible d'observer un graphe de charge mesurant le nombre de messages reçus au sein d'un groupe donné. Chaque colonne est formée de trois informations superposées verticalement :
  - le nombre de messages correspondant aux traitements d'erreurs tels que des demandes de retransmission ou des messages échangés lors d'une phase de resynchronisation. L'affichage de cette charge est en couleur rouge.
  - les messages de diffusion de la première phase du protocole (MSG_SEND) sont affichés en bleu foncé.
  - les messages de délivrance en provenance du séquenceur (MSG_DELIVER) sont affichés en cyan.

  L'état actuel du groupe sur ce site est aussi fourni.



  La baisse d'activité au sein du groupe que l'on peut observer correspond à une phase de resynchronisation après l'arrêt d'un des sites de stockage.
- Une vue donne des informations propres à chaque station, en particulier l'état des tampons de réception et d'émission, ainsi que la liste des groupes auquel le site a souscrit.

- Enfin une vue générale fournit la liste des stations actives et des groupes actifs à un instant donné de la simulation. Cela permet de sélectionner les vues qui offrent un intérêt, pour peu que le nombre de stations en jeu soit important, l'espace de travail de l'utilisateur serait immédiatement surchargé.

Un graphe montrant l'évolution temporelle de la charge sur le réseau achève de donner une vue d'ensemble concernant la simulation en cours.

IV.2.3 Résultats

Le principe d'émulation du réseau et des parties basses du système opératoire s'est avéré très efficace pour la mise au point du protocole, le principal avantage est la possibilité de rejouer de manière déterministe des scénarios de tests du protocole afin de valider toute modification. Une fois un problème isolé, et correctement corrigé, un script de configuration provoquant les conditions de fautes peut alors être mis au point. Il sera lancé à chaque étape de validation. Les différentes interfaces ont chacune leur utilité, l'environnement fenêtré permetant de détecter agréablement les conditions provoquant une erreur du protocole, en particulier les échanges de messages précédant ou démontrant le dysfonctionnement. On a alors recours à l'interface en ligne pour préciser l'état courant des divers composants (processus, tampons, etc...) et si nécessaire il est possible de relancer l'émulation sous le contrôle d'un déboggeur symbolique, ce qui offre l'accès à toutes les variables internes.

Toutefois certains cas d'erreurs ne peuvent être reproduits sur un émulateur, en particulier lorsque le comportement du système réel n'est pas modélisé car trop différent du modèle standard fourni par la simulation. Deux exemples ont été rencontrés lors de la mise au

point du protocole. Le premier a trait à l'implantation sur Mach 3.0 de l'OSF ; le micro−noyau considère les messages Ethernet en diffusion comme non prioritaire, ce qui, sous forte charge, entraîne la perte des seuls messages en diffusion. Il arrive aussi qu'un message en diffusion A émis avant un message point−à−point B (depuis un même site) soit reçu dans l'ordre inverse (B puis A). Il est aussi difficile de modéliser le comportement d'un site supportant une forte charge ponctuelle d'entrées/sorties, qui font chuter momentanément les débits réseau sans toutefois stopper complètement le protocole.

IV.3 Autres outils

Si l'utilisation d'un émulateur facilite la tâche du développeur de protocoles, certaines situations délicates ne peuvent être simulées. On doit alors tenter d'analyser le comportement du protocole en situation réelle. La première solution consiste à générer des traces sur chaque système mis en œuvre lors de l'exécution, et de tenter a posteriori d'analyser le comportement global par recoupement. Cette méthode est la plus puissante car elle permet d'analyser le comportement du protocole dans son ensemble depuis l'interaction avec les couches du système d'exploitation jusqu'à l'interface d'utilisation, à la condition que la production des traces n'influe pas trop sur le bon déroulement du protocole. La deuxième approche consiste à espionner le réseau depuis une station ne participant pas au protocole, cette possibilité nous est offerte par la nature même d'Ethernet (le médium est partagé par toutes les stations connectées à un segment). Il est possible de capter tous les messages émis en diffusion, et certaines interfaces peuvent aussi basculer dans un mode de collecte systématique de tous les paquets transmis sur le réseau.

Cette technique d'espionnage peut s'appuyer sur des outils existants tels que **tcpdump** disponible sur SunOs et Linux. Malheureusement les informations fournies sont de très bas niveau (simple affichage hexadécimal des premiers octets des paquets interceptés) car le protocole de diffusion n'est pas connu de tels outils. Pour simplifier la tâche de mise au point et pour permettre l'observation de l'exécution du protocole, les outils d'observation développés pour l'émulateur de réseau ont été "greffés" au dessus du module d'interface avec UNIX. Le programme résultant permet d'espionner le trafic réseau associé au protocoles de diffusion en activité sur le segment Ethernet sur lequel est connecté la machine, l'interface est très similaire à celle déjà décrite précédemment pour l'émulateur.

Chapitre V

Résultats des mesures

Ce chapitre expose tous les résultats obtenus lors de tests du protocole. Cela comprend des mesures de performances générales et l'analyse des temps d'exécution pour déterminer l'impact de l'implantation en mode utilisateur. L'étude de l'effet de divers paramètres sur les performances est aussi fournie, ainsi que l'impact des optimisations suggérées lors de l'augmentation des pertes de paquets.

V.1 Méthode de mesure

L'évaluation de la qualité d'un protocole de communication demande de prendre en compte beaucoup de facteurs, souvent contradictoires. Par exemple, on cherche à obtenir les débits de transmission les plus élevés possible, tout en maintenant la charge processeur associée la plus faible possible. Dans notre cas, celui d'un protocole de groupe donc multi−point, la notion même de débit n'est pas évidente : s'agit−t−il du débit maximum que peut émettre un site au sein du groupe, les autres sites se contentant de recevoir les messages, ou le débit maximum obtenu lorsque tous les sites participent ? De plus dans le cas de l'utilisation au sein d'un système réparti, le facteur primordial est−t−il le débit ou le nombre de messages transmis par unité de temps ?

Dans le cadre de l'utilisation du protocole pour le support d'un système réparti, il nous a semblé plus important de mesurer la "réactivité" du protocole, c'est−à−dire son aptitude à délivrer rapidement une grande quantité de messages. Typiquement la taille des messages échangés dans ce type d'application est petite, qu'il s'agisse des paramètres d'un appel distant, ou du contenu d'un objet manipulé par le système. C'est pourquoi les mesures de performances générales exposées dans la section V.2 suivante considèrent des messages de la taille d'une dizaine d'octets. L'influence de la taille des messages sur les performances sera étudiée spécifiquement au V.3.1. La puissance, l'architecture et le rôle respectif au sein du protocole des machines mises en jeu sont des facteurs très importants : aussi dans un premier temps nous nous restreindrons à un jeu de machines homogènes, puis l'influence de chacun des critères sera exposée au séparément V.3.

La méthode utilisée pour les mesures consiste à faire exécuter sur chaque site un programme identique effectuant une connexion à un groupe commun ; chaque client émet alors un message au sein du groupe, et lors de la réception d'un message sur un site, si le site courant est l'émetteur, il régénère le message en le renvoyant au sein du groupe. Le programme affiche à intervalles réguliers le taux de messages reçus par seconde, ainsi que le taux de transmission de ses propres messages. Cette méthode permet de tester de manière

réaliste le comportement du protocole dans le cadre de son utilisation pour des applications coopératives au sein d'un système réparti.

V.2 Performances générales

Cette section présente les résultats obtenus sur un groupe de machines homogènes, puis analyse les temps d'exécution relatifs a l'implantation sur le système Linux.

V.2.1 Résultats

Les résultats suivants ont été obtenus en utilisant des machines Unix basées sur des processeurs de type PowerPC 604 cadencés à 75 MHz, et tournant sous le système d'exploitation AIX d'IBM. Le réseau, de type Ethernet à 10 Mbits/s, bien que peu chargé, reflétait des conditions normales d'utilisation, à savoir une vingtaine de stations UNIX et autant de terminaux X−Windows y étaient connectés. Les stations utilisées étaient en mode multi−utilisateurs, mais sans charge particulière. Le graphe ci−dessous montre les résultats obtenus dans cette configuration :

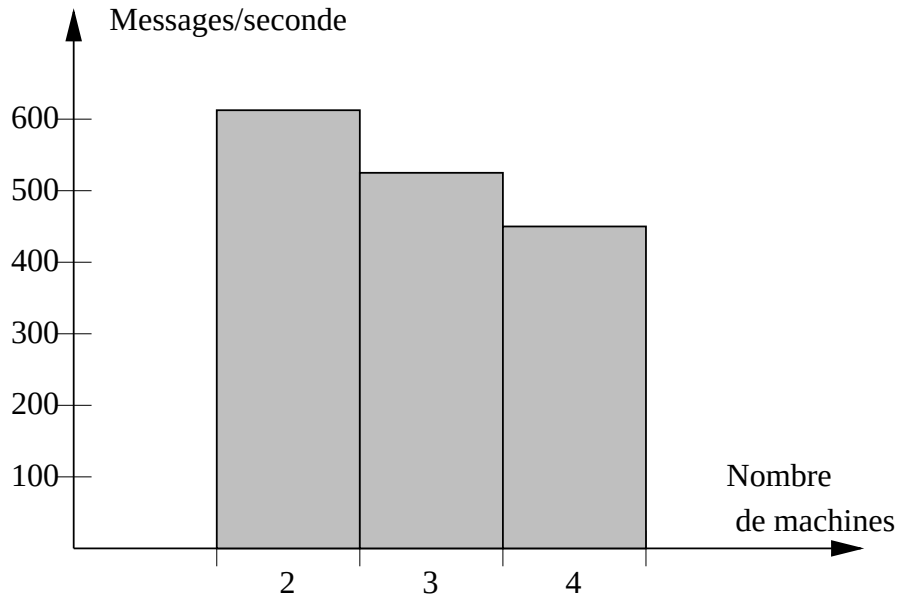


Fig. 5.1 : Performances du protocole

Cette figure indique bien les principales caractéristiques de l'implantation réalisée.

- Les performances obtenues sont bonnes, la réactivité est suffisante pour servir par exemple de base à une application répartie ayant une forte interaction avec des utilisateurs, ou à la mise en œuvre d'une gestion d'objets ou de fichiers dupliqués.

  Cela valide les principaux choix de réalisation, en particulier la possibilité d'implanter le protocole au sein d'un processus en mode utilisateur. La sous−section suivante offre une analyse détaillée du coût associé aux changements d'espaces.

- Bien que le nombre de machines dans cette expérience soit limité (seule quatre machines identique de ce type étaient alors disponibles), la courbe indique nettement le bon comportement du protocole avec l'augmentation du nombre de sites. De même cette caractéristique valide le choix qui a été fait d'utiliser le mécanisme de diffusion offert par Ethernet ; les expériences ont aussi démontré l'innocuité de l'utilisation de la diffusion pour les sites non membres de l'expérience. Les faits ont prouvé que toutes les machines atteintes par les paquets en diffusion n'ont pas montré de dégradation de performances (à l'exception de terminaux X Gipsi, désormais obsolètes) malgré le trafic élevé − plus d'un millier de diffusions par seconde − engendré par les tests.

Cette expérience préliminaire valide l'implantation et les choix principaux qui ont guidé la mise en œuvre. La suite de ce chapitre consiste en une analyse plus fine du comportement du protocole, en particulier sous l'influence de divers paramètres.

V.2.2 Analyse temporelle

Une meilleure compréhension des performances du protocole nécessite d'étudier l'impact de l'implantation de celui−ci en espace utilisateur. En particulier les coûts associés aux mouvement des données, et aux passages fréquents entre l'espace noyau et l'espace utilisateur sont importants car ils sont directement liés aux choix d'implantation. De telles mesures nécessitent la mise en place d'outils de mesure au sein même du système d'exploitation. Pour cette raison nous avons choisi d'effectuer ces mesures sur le système Linux dont les sources sont aisément disponibles et modifiables. La méthode utilisée consiste à insérer du code sauvegardant la valeur courante d'une horloge interne en divers points du chemin d'exécution emprunté lors de la réception et de l'envoi de paquets, tant dans le code du noyau que dans celui du protocole. L'horloge disponible sur les machines de type IBM−PC fournit une précision interne de l'ordre de la microseconde, accessible via l'appel `gettimeofday(2)` de Linux de manière fiable. Les mesures glanées dans le noyau et le programme utilisateur doivent ensuite être recoupées pour obtenir l'ensemble des données nécessaires au calibrage d'une exécution complète. Enfin il faut retrancher le délai nécessaire à la lecture de l'horloge et à sa sauvegarde pour obtenir le temps d'exécution entre deux points de mesure.

Les mesures ont été effectuées sur une version de Linux 1.2.8 modifiée pour collecter des mesures dans le noyau. Les cartes réseaux étaient d'un type identique, à savoir des WESTERN DIGITAL 8013, connectées à un bus ISA 16 bit. Les processeurs sont tous de la famille Intel, et disposent d'une architecture interne 32 bits. Par contre les schémas d'accès aux caches, leur taille et la largeur du bus mémoire ne sont pas identiques, de 32 Ko en mode Write Through 32 bit pour le 386, jusqu'à 256 Ko géré en mode Write Back via un bus 64 bits pour le Pentium. Le processeur 486DX/2 66 MHz est fonctionnellement identique au 486DX 33 MHz mais sa fréquence interne est doublée, ce qui accélère de manière significative les opérations qui ne nécessitent pas d'accès hors du processeur (le 486 dispose d'un cache interne), il est donc intéressant de comparer le comportement de ces deux systèmes lors d'opérations d'entrées/sorties.

La figure suivante illustre le chemin d'exécution parcouru lors de la réception d'un paquet Ethernet, depuis la routine d'interruption gérée par le pilote jusqu'au traitement du paquet par le protocole.

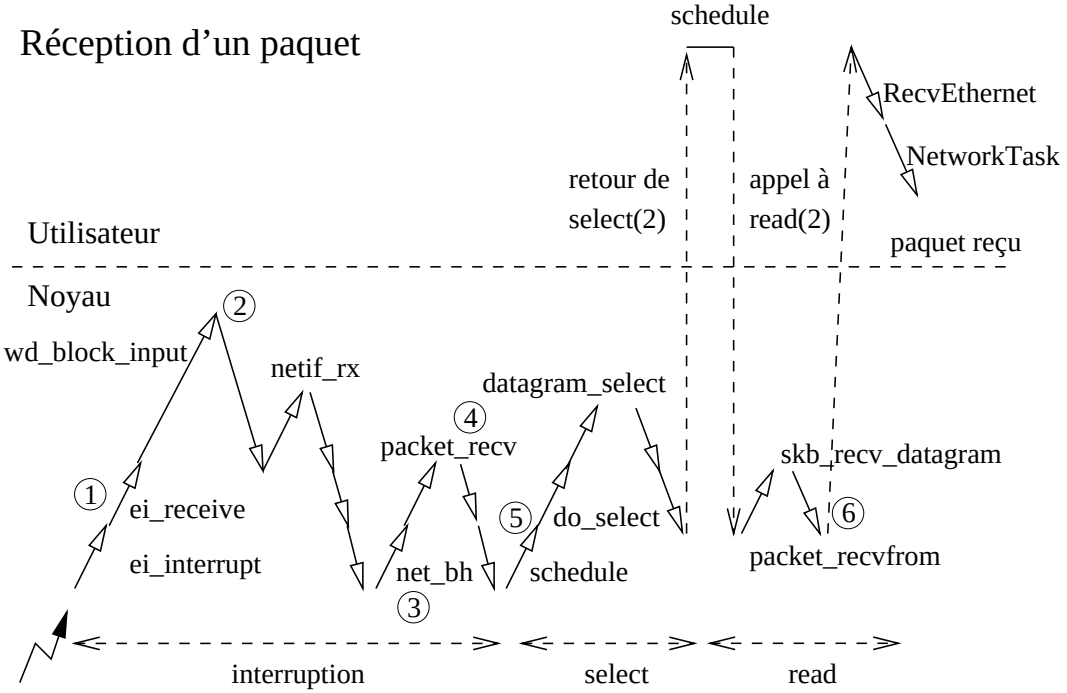


Fig. 5.2 : Séquence exécutée à la réception d'un paquet

L'ensemble des opérations effectuées à la réception d'un paquet peut être divisé en trois parties.

- Le traitement de l'interruption, qui est lui même scindé en deux parties. La première consiste en l'appel de la routine d'interruption du pilote de la carte Ethernet, cette partie s'effectue interruptions masquées. La deuxième est un épilogue traité en mode démasqué et donc interruptible. Dans la première partie, le pilote détecte qu'il s'agit de la réception d'un paquet, prend un tampon (1) parmi un lot pré−alloué, dans lequel la routine de lecture va copier le paquet (2) depuis la mémoire du coupleur. Le tampon est alors stocké dans une file de réception et l'interruption proprement dite se termine. L'épilogue est alors exécuté, en mode noyau dans le contexte de la tâche interrompue, et transfère le paquet depuis la file associée au périphérique (3) vers le protocole destinataire (4).
- L'étape suivante consiste à relancer le processus associé au protocole, et à lui rendre la main. Une fois le traitement de l'interruption terminé, le noyau appelle la routine `schedule()` qui effectue l'attribution du processeur. Le changement de contexte (5) vers le processus du protocole a alors lieu, celui−ci reprend son exécution dans la routine `do_select()` où il s'était bloqué. L'appel `datagram_select()` associé au descripteur de fichier du périphérique est alors passant, l'appel système se termine et le contrôle est rendu à l'application.

- Enfin le processus peut procéder à la lecture du paquet. Un appel système `read(2)` est lancé qui a pour effet de transférer à nouveau le contrôle dans le noyau à la routine packet_recvfrom. Celle−ci appelle skb_recv_datagram qui extrait le paquet des couches basses de la pile de protocoles, le contenu du paquet est alors recopié dans l'espace utilisateur (6). L'appel système se termine, la routine RecvEthernet du protocole est alors en possession du paquet.

Ce déroulement idéal peut toutefois être légèrement modifié si le protocole est en concurrence pour la ressource processeur : dans ce cas il se peut que le changement de contexte de l'étape (5) soit différé, voire que plusieurs paquets s'accumulent dans le noyau avant que le protocole ne reprenne le contrôle pour les lire.

Le tableau ci−dessous fournit les temps d'exécutions en microsecondes obtenus pour une exécution donnée effectués en une seule passe : en tout huit mesures distinctes ne différant que par la machine cible (processeur) et la taille des messages transportés :

Processeur	Pentium 90		486DX/2 66		486DX 33		386DX 25	
Taille	10	1000	10	1000	10	1000	10	1000
ei_interrupt	0	0	0	0	0	0	0	0
ei_receive	10	10	13	19	16	20	36	36
wd_block_input	15	14	22	22	23	26	45	43
wd_block_input end	53	393	90	360	103	418	154	555
netif_rx	3	3	4	5	2	4	9	10
netif_rx end	1	3	9	8	9	10	26	25
ei_receive end	19	19	22	22	27	27	43	42
ei_interrupt end	7	9	9	42	11	13	15	15
net_bh	6	6	10	19	12	16	35	38
packet_recv	14	15	39	48	45	53	119	126
packet_recv end	12	10	33	38	34	37	79	74
net_bh end	11	12	21	21	37	36	111	109
schedule	4	5	9	7	9	7	24	26
schedule switching	17	21	43	48	103	102	92	91
do_select	9	9	19	18	21	20	36	34
datagram_select	9	9	24	27	24	25	44	44
datagram_select end	3	1	3	5	4	4	8	9
do_select end	11	10	36	34	37	37	61	60
user : schedule	11	12	37	38	47	54	135	133
thread_read start	15	14	43	47	51	48	122	117
packet_recvfrom	15	9	28	28	41	42	295	297
skb_recv_datagram	2	4	8	8	8	8	14	15
skb_recv_datagram end	4	2	11	8	9	8	22	22
packet_recvfrom end	16	89	59	162	64	170	136	488
RecvEthernet end	5	15	9	12	23	25	107	112
net : got packet	5	4	9	11	15	15	46	49
total interruption	151	494	272	604	319	660	672	1073
total transfert	64	67	171	177	245	249	400	397
total lecture	62	137	167	276	211	316	742	1100
TOTAL	277	698	610	1057	775	1225	1814	2570

Fig. 5.3 : Décomposition du temps lors de la lecture d'un paquet

Ces résultats montrent la nette évolution des processeurs, et l'importance de l'augmentation de leur puissance de traitement, même pour des opérations d'entrées/sorties. On peut en effet constater que le temps de réception est globalement divisé par un facteur 10, alors que le processeur 386 ne date que de 5 à 6 ans. Paradoxalement c'est moins la vitesse d'exécution brute qui a été améliorée (on ne constate souvent qu'un gain d'un facteur 5), que le coût associé aux changements de contexte (fin de la routine net_bh_end) et aux appels systèmes (user : schedule, packet_recvfrom et packet_recvfrom end) où le gain atteint souvent 10 ou plus. Cette amélioration s'explique aussi par le fait que Linux utilise les instructions de gestion de tâches et de changement de contexte spécifiques à la famille de processeur Intel, et a donc pleinement tiré parti des optimisations des nouvelles versions.

Par contre, on n'observe pas un gain important entre les deux versions du processeur 486, et ce malgré une charge plus importante du processeur en version 33 MHz lors des tests − le temps pris par l'ordonnanceur (schedule) est plus long. Cela montre bien qu'une augmentation de la puissance du processeur n'est effectivement visible que si l'architecture environnant le processeur évolue de manière adéquate. Cela est aussi particulièrement visible lors de la recopie du paquet en espace utilisateur, opération qui a lieu en fin de la routine packet_recvfrom : cette opération qui force des accès hors du cache interne s'exécute sensiblement à la même vitesse sur les deux processeurs.

L'obsolescence de l'architecture interne des machines de type IBM−PC apparaît nettement lors de la lecture du paquet depuis la carte Ethernet (wd_block_input end), cette carte ne fournit pas de copie en DMA, aussi le processeur effectue la lecture directement via le bus ISA, hérité des premières machines IBM datant du début des années 80. Sur les machines actuelles le temps nécessaire à cette recopie devient le principal facteur du temps nécessaire à l'acquisition des données, et est indépendant du processeur effectuant cette opération. On peut ainsi noter que le 486DX/2 effectue la lecture d'un paquet important plus rapidement que le Pentium : en effet, pour limiter le goulot d'étranglement du bus, certaines machines ont une horloge bus supérieure à la vitesse théorique assurant la compatibilité. De plus sur le Pentium, l'accès au bus ISA passe par l'intermédiaire du contrôleur du bus PCI.

L'influence de la taille du paquet reçu sur le code exécuté est localisée en deux points précis :

- lors de la lecture du paquet de la carte vers l'espace du noyau qui a lieu dans wd_-block_input end, le temps passé est proportionnel à la taille du paquet (opération assembleur REP INSW du jeu d'instruction 80x86).
- lors de la recopie du paquet depuis l'espace noyau vers le processus implantant le protocole. Sous Linux, l'espace noyau étant couplé dans l'espace du processus, il s'agit d'une recopie simple sans manipulation de pages (opération assembleur REP MOVSL qui copie 32 bits par 32 bits) et est donc là aussi d'un coût proportionnel à la taille du paquet.

De plus ces recopies ont un effet diffus, dû à la perturbation des caches interne et externe du processeur, ce qui explique la tendance générale à une exécution plus longue sur l'ensemble du chemin dans le cas d'un paquet de 1 Koctet, même lorsque le paquet n'est pas manipulé. Ce phénomène est particulièrement visible juste après la deuxième copie (RecvEthernet end) lors du retour en mode utilisateur.

La figure suivante décrit de manière analogue le chemin d'exécution suivi lors de l'émission d'un paquet depuis le protocole implanté en mode utilisateur :

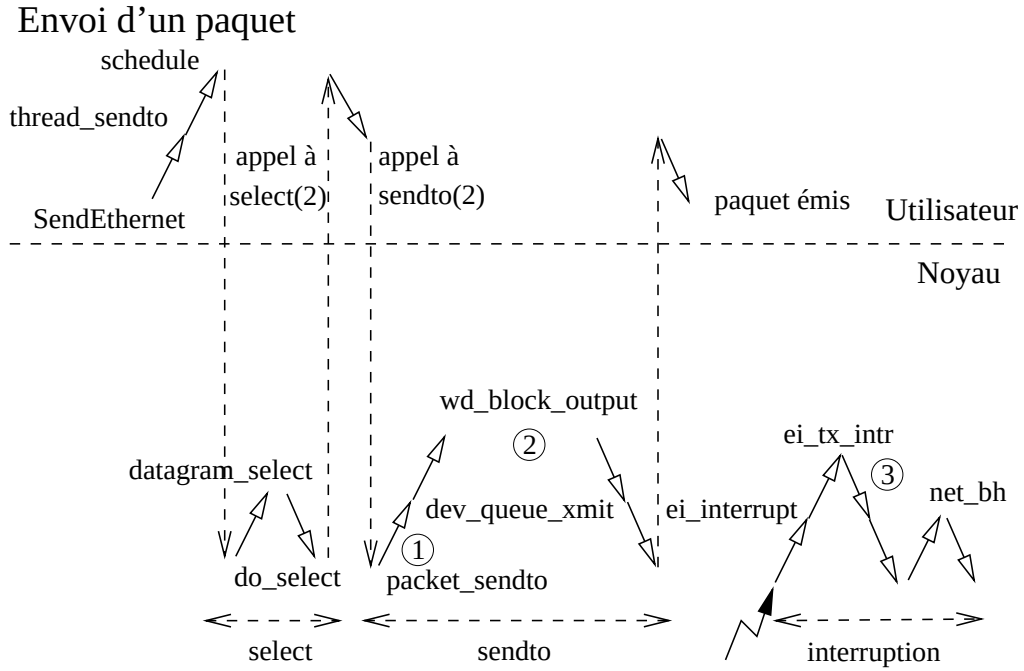


Fig. 5.4 : Séquence exécutée à l'émission d'un paquet

L'envoi d'un paquet comporte aussi trois phases :

- l'appel initial à SendEthernet invoque la routine d'écriture du noyau de processus légers. Afin de ne pas faire d'appel système bloquant, un appel système `select(2)` est effectué, ce qui transfère le contrôle dans la routine do_select du noyau puis à datagram_select. Si le pilote est prêt à recevoir des données, l'appel système retourne cette information, l'ordonnanceur peut alors décider de lancer l'appel système d'écriture ou de bloquer le flot courant jusqu'au moment où le pilote acceptera des données. En pratique, l'écriture sur le périphérique ne risque pas de bloquer le processus du protocole pour une longue période et il serait possible de se dispenser de ce test, ce qui permettrait de réduire le chemin critique, au détriment d'une gestion précise de l'ordonnancement des divers flots du protocole.
- l'écriture du paquet via le pilote Ethernet se fait via l'appel système sendto(2). La routine système packet_sendto recopie alors les données depuis l'espace utilisateur (1) vers un tampon dans le noyau. Celui−ci est installé dans une file d'émission associé au périphérique, et la routine wd_block_output du pilote charge le paquet dans la mémoire de la carte. L'envoi est alors déclenché, l'appel système se termine, le contrôle est transféré au protocole qui considère le paquet comme émis.
- une fois le paquet effectivement transmis sur le réseau, la carte Ethernet génère une interruption, la routine ei_tx_intr vérifie alors l'absence d'erreur et relance si nécessaire un nouveau transfert (3). Tout comme lors d'une interruption en réception, l'épilogue net_bh est exécuté, celui−ci traite alors d'éventuel paquet entrant et

relance les transmissions si nécessaire. Il est à noter que tout le traitement de l'interruption peut s'effectuer durant une tranche de temps affectée a un autre processus Unix que celui du protocole, si un changement de contexte a eu lieu entre l'initialisation de l'envoi et la fin de transmission du paquet.

La figure suivante fournit la décomposition temporelle de l'émission d'un paquet depuis le mode utilisateur. Ces mesures ont été produites lors de la même série de tests que celles présentées à la figure Fig. 5.3, et donc dans les mêmes conditions.

Processeur	Pentium 90		486DX/2 66		486DX 33		386DX 25	
Taille	10	1000	10	1000	10	1000	10	1000
net : sending	0	0	0	0	0	0	0	0
SendEthernet	8	62	22	111	25	120	65	344
thread_sendto	8	8	16	18	17	22	53	64
user : schedule	26	24	57	58	72	71	303	297
do_select	12	12	53	49	58	65	336	317
datagram_select	14	14	56	56	62	61	104	92
datagram_select end	2	4	9	8	9	9	17	17
do_select end	10	12	36	30	36	36	63	73
user : schedule back	7	9	35	34	36	36	122	113
thread_sento go	9	9	35	35	39	39	103	101
packet_sendto	17	17	? 104	54	55	56	136	135
dev_queue_xmit	15	86	75	179	67	186	146	535
wd_block_output	15	15	45	47	43	45	108	128
wd_block_output end	24	320	27	281	22	231	41	429
dev_queue_xmit end	31	36	70	74	72	74	* 58	168
packet_sendto end	1	2	4	3	2	3	14	12
SendEthernet done	9	10	* 36	43	* 48	42	110	114
net : Msg sent	5	7	10	9	10	13	27	29
ei_interrupt	(101)	(923)	(133)	(953)	(137)	(955)	(163)	(1009)
ei_tx_intr	11	11	18	22	18	22	34	39
ei_tx_intr end	6	7	12	13	13	14	29	28
ei_interrupt end	9	9	11	11	12	12	23	26
net_bh	5	6	15	17	17	17	36	36
net_bh end	14	14	32	34	40	45	122	127
total transfert	87	145	284	364	315	420	1063	1317
total écriture	126	502	406	725	358	689	743	1651
total interruption	45	47	88	97	100	110	244	256
TOTAL	213	694	778	1186	773	1219	2050	3224

Fig. 5.5 : Décomposition du temps lors de l'envoi d'un paquet

Un préliminaire à l'analyse de ces données est de constater que la routine d'interruption traitant la fin de l'émission d'un paquet par la carte est asynchrone vis−à−vis de l'exécution du protocole, une fois la transmission démarrée. Celle−ci survient au bout d'un laps de temps dépendant fortement de la taille du paquet, et son traitement interrompt l'exécution courante, ce qui peut rendre malaisée la collecte des mesures. C'est le cas lors de l'émission de messages de petite taille sur les processeurs les moins rapides ; une astérisque signale les mesures faussées par le traitement de la routine d'interruption qui se déclenche alors avant même le retour de l'appel système en écriture. L'intervalle de temps écoulé entre la fin de la routine wd_block_output et l'interruption est indiqué entre parenthèses.

Les remarques concernant l'évolution de la puissance des processeurs faites précédemment sont confirmées. Les opérations qui ralentissent l'ensemble du traitement restent du même type, mais en des points différents :

- les recopies en mémoire : la première a lieu dans la toute première phase en mode utilisateur lors de la mise en forme des données utilisateurs dans un paquet (entre net : sending et l'appel à la routine SendEthernet). La deuxième copie a lieu dans la routine packet_sendto lorsque le noyau transfère le contenu du paquet dans son propre espace.
- les appels systèmes : avant même l'appel à select puis à sendto, la routine schedule apelle gettimeofday, donc trois transferts de contrôle vers le noyau sont réalisés avant l'envoi du paquet.
- l'accès à la carte Ethernet via le bus ISA, est réalisé dans wd_block_output pour recopier le contenu du paquet émis dans le tampon de transmission de la carte.

L'opération d'envoi est globalement un peu plus simple que la réception, car il n'y a pas de démultiplexage et de changement de contexte à effectuer, mais elle nécessite plus d'opérations coûteuses. On constate donc que l'envoi d'un paquet nécessite un traitement plus long que la réception sauf pour la machine Pentium où cette tendance s'inverse du fait de l'amélioration des taux de transfert en mémoire et du temps de traitement nécessaire aux appels systèmes.

V.2.3 Analyse et conclusion

Le coût associé à l'implantation du protocole en espace utilisateur décroît nettement avec l'augmentation de puissance des machines. C'est un des arguments les plus couramment utilisés pour relativiser la surcharge imposée par les nouvelles architectures de système, en particulier celles basées sur un micro−noyau. En effet les chiffres obtenus mettent bien en évidence la difficulté d'une implantation efficace de ce type d'organisation sur la génération précédente de processeurs.

Si l'augmentation de la puissance des processeurs rend acceptable la gestion d'un protocole en mode utilisateur basé sur un réseau de type Ethernet, il semble difficile d'extrapoler ces résultats pour les nouveaux réseaux à haut débit tels qu'ATM et Ethernet 100 Mbits/s. L'augmentation de la bande passante du réseau semble plus rapide que celle des bus mémoire et des caches liés aux processeurs. De ce fait, l'organisation matérielle nécessaire au support de ces nouveaux réseaux doit évoluer radicalement :

- utilisation de périphériques intelligents, où l'essentiel de la logique nécessaire à l'implantation des protocoles standards − les couches AAL (ATM Adaptation Layer) pour ATM par exemple − est mise en œuvre directement sur la carte d'interface.
- gestion des transferts entre périphérique et mémoire vive réalisée en DMA (Direct Memory Access) donc sans nécessiter l'intervention du processeur. Toutefois ce type d'accès nécessite de réinitialiser fréquemment les caches mémoire pour éviter des incohérences, ce qui est très pénalisant pour les nouvelles générations de processeurs.

De même l'organisation logicielle des systèmes s'oriente vers la reconfiguration dynamique du noyau à la demande. Le principe consiste à charger dynamiquement les pilotes et les protocoles spécifique dans l'espace du noyau, et donne lieu à plusieurs type de supports.

- Les STREAMs, présents déjà depuis plusieurs années dans les versions d'Unix System V, permettent de construire des protocoles spécifiques en imbriquant l'action de divers modules chargés à la demande dans le noyau et agissant comme des filtres successifs sur les données en provenance d'un pilote de périphérique.
- Le chargement et l'édition de lien dynamique de nouveau objets au noyau existant. Cela offre une grande souplesse et permet d'intégrer aussi bien de nouveaux pilotes que des protocoles ou des appels systèmes. Ces fonctionnalités sont présentes par exemple sous Solaris, AIX et Linux.

Le grand avantage de ce type d'organisations logicielles est qu'il permet l'implantation de protocoles spécifiques sans avoir la dégradation des performances due aux transferts de contrôle entre le noyau et le protocole. Il réduit aussi le nombre de recopies mémoire nécessaires au traitement des messages, en particulier les informations de contrôle qui ne remontent pas jusqu'à l'utilisateur. Le coût associé est celui d'une perte de portabilité, car si l'interface des STREAMS est normalisée, toutes les références aux pilotes et aux points d'entrée fournis par le noyau sont bien sûr spécifiques au système d'exploitation considéré ainsi qu'au type des pilotes et d'interfaces utilisés.

V.3 Effets de divers paramètres

Dans cette section, nous étudions l'impact de divers paramètres sur les performances du protocole.

V.3.1 Taille des messages

Nous avons déjà pu apprécier l'impact de la taille des messages véhiculés sur les diverses opérations systèmes liées à l'implantation du protocole. Cette série de mesure offre une vue plus globale de l'évolution des performances, et teste en particulier le coût induit par la fragmentation et le réassemblage lorsque les données utilisateurs ne peuvent être stockées dans un unique paquet Ethernet (taille limitée à 1500 octets). Les mesures ont été réalisées sur des machines à base de PowerPC, dans les même conditions que celles déjà présentées au V.2.1. Le graphe ci−dessous montre les performances obtenues avec des tailles de messages de 10, 1000 et 2000 octets, et un nombre de stations, toutes sites de stockage, variant entre deux et quatre.

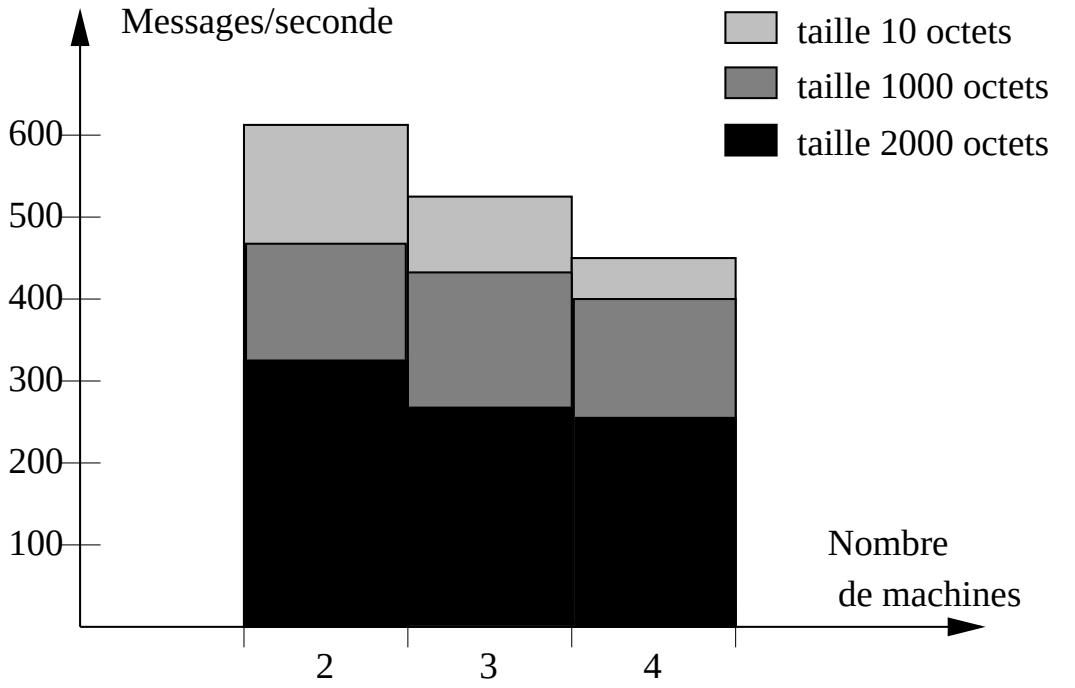


Fig. 5.6 : Évolution du débit en fonction de la taille des messages

On observe une faible dégradation des performances avec l'augmentation de la taille des messages, et surtout avec le nombre de machines. Plusieurs facteurs permettent d'expliquer ce bon comportement:

- le surplus de traitement dû à l'augmentation de la taille du paquet peut être aisément absorbé par les machines récentes. Il est finalement assez faible en comparaison des multiples communications nécessaires avant la délivrance d'un message.
- l'utilisation de la diffusion Ethernet permet de soulager la bande passante lors de l'envoi des informations à un groupe de machines. Dans le cas extrême, les messages véhiculés sont de 2 Koctets et le débit est de 260 messages transmis par seconde pour un groupe de quatre sites. La réalisation d'un tel transport basé sur des communications point−à−point nécessiterait une bande passante de 1.56 Mo/s (sans compter les messages de contrôle) ce qu'Ethernet ne peut pas fournir.

On constate aussi que doubler la taille des messages ne divise pas par deux les performances du protocole. Cela est du à la diminution relative des messages de contrôle par rapport à l'ensemble du trafic et à la simultanéité de traitement entre l'émission des fragments et leur réception. Ce phénomène, qui est à la base des protocoles à fenêtre glissante, s'observe aussi pour les protocoles multi−points.

Enfin, même si le protocole n'a pas été conçu pour faire du transport de gros volumes de données, on constate qu'il utilise convenablement la bande passante (jusqu'à 650 Ko/s lors du test avec deux sites et des messages de 2 Ko). Toutefois, il ne parvient pas à saturer le réseau du fait du caractère synchrone des acquittements, et sans doute au surcoût dû à l'implantation en espace utilisateur.

V.3.2 Puissance des machines

L'influence de la puissance du processeur sur les performances du protocole a clairement été montrée lors de l'analyse temporelle au V.2.2. La figure suivante illustre graphiquement l'évolution des performances sur des machines de type IBM PC 486 à 33 et 66 MHz − d'une puissance évaluée a 17 et 33 Mips respectivement − et pour des messages de petite taille (10 octets utilisateur). Les performances n'atteignent pas celles obtenues avec les machines de type PowerPC, mais restent du même ordre de grandeur :

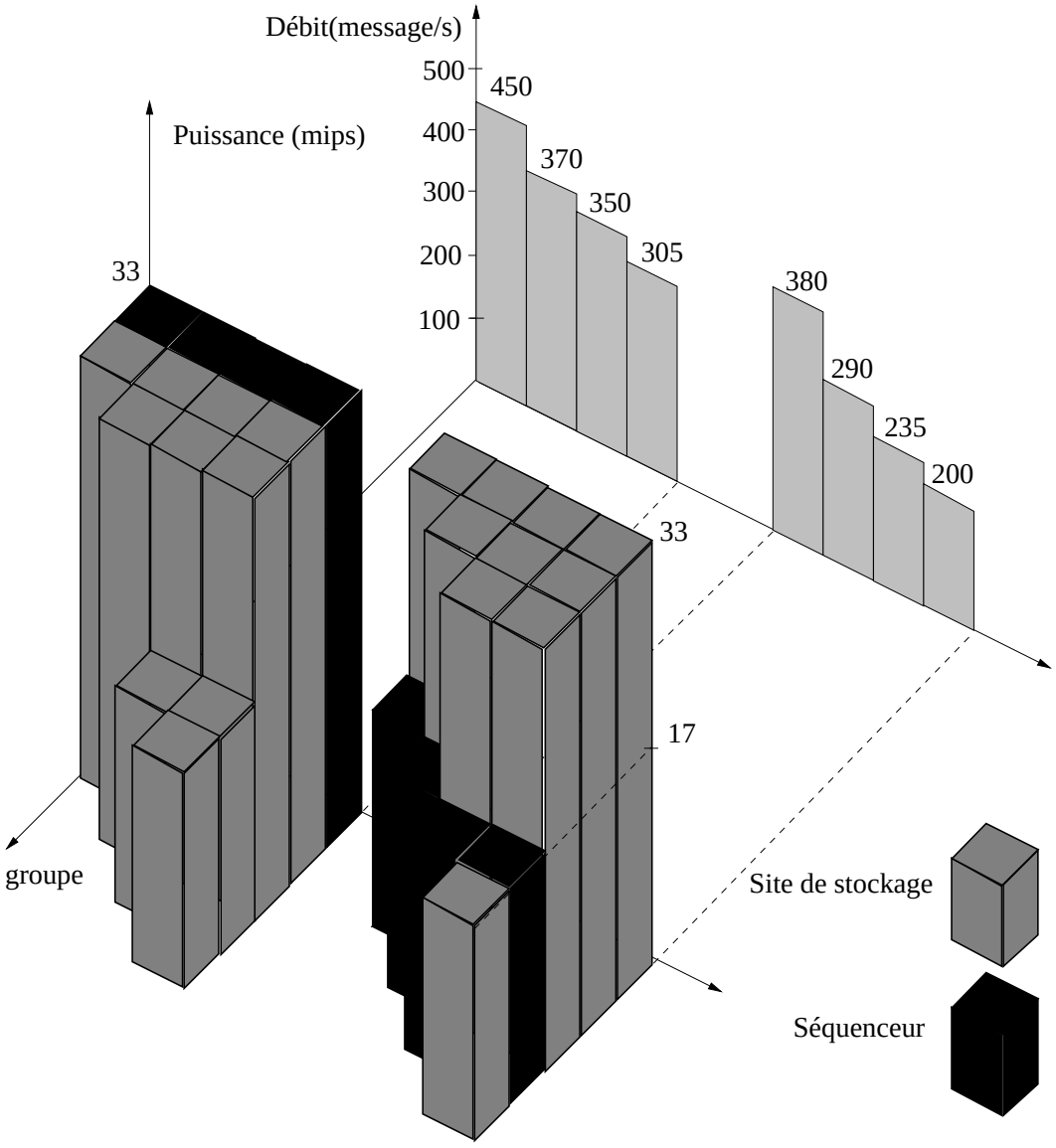


Fig. 5.7 : Résultats sur des machines standards

Le graphe montre deux séries de quatre expériences, chacune avec une composition du groupe différente, le séquenceur étant visualisé en noir, les sites de stockage en gris. Chaque site est représenté par un parallélépipède, dont la couleur indique le rôle et la hauteur la puissance. Le débit supporté par cette configuration du groupe est montré en projection sur

l'arrière plan. Dans la première série de mesures, les sites les plus rapides rejoignent progressivement le groupe, puis les deux machines plus lentes. On observe une dégradation faible du débit avec le nombre de sites. Dans la deuxième série, le séquenceur est initialement placé sur un des sites les moins rapides, puis les autres sites rejoignent le groupe. Les performances diminuent alors assez rapidement avec l'arrivée de nouveaux sites.

Le résultat de ces mesures, en particulier la comparaison avec les machines de type PowerPC, montre que la puissance des machines assurant le fonctionnement du groupe influent assez fortement sur les performances obtenue. En effet on note une dégradation d'environ 25 % lors du passage des PowerPC aux Intel 486. De plus, la comparaison entre les deux séries de mesures suggère que le placement du site séquenceur parmi les différentes machines a une influence encore plus forte. Dans ce cas, le placement du séquenceur sur un des sites les moins rapides engendre une perte qui atteint jusqu'à 33 % lorsque le groupe comporte 5 sites.

V.3.3 Placement du séquenceur

Les résultats du jeu de tests précédent nous ont incités à refaire une série de mesure traitant plus spécifiquement de l'influence de la puissance du séquenceur sur les performances du protocole. Pour ce faire nous avons repris les tests précédents mais en introduisant une machine utilisant un processeur de type 386DX à 25 MHz. La puissance estimée de celui−ci avoisine les 5 Mips (en comparaison des 17 et 33 Mips pour les 486). Deux séries de mesures ont été réalisées avec à chaque fois une composition du groupe identique. La différence ne portait que sur le placement du séquenceur, ce rôle était assuré dans la première série par le 386 et dans la deuxième par un 486 DX/2 66, le graphe suivant montre les résultats obtenus :

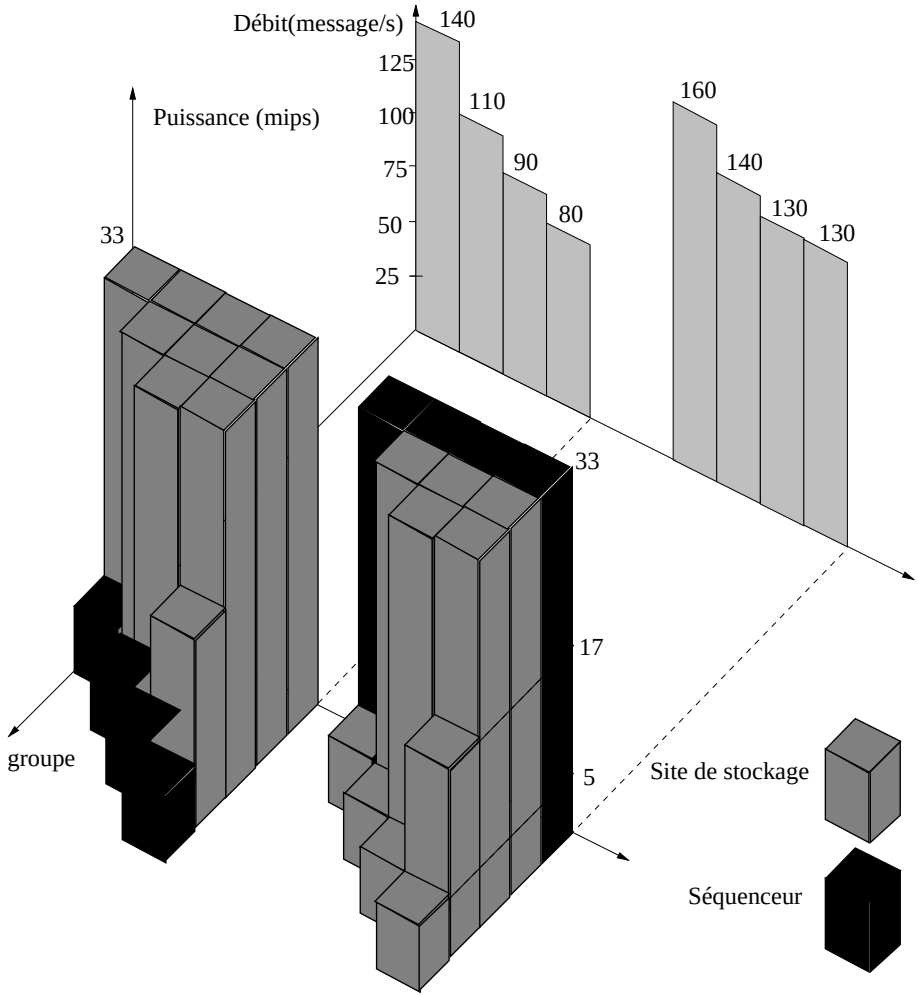


Fig. 5.8 : Influence du choix du séquenceur

La dégradation des performances due à l'utilisation du 386 est immédiatement visible, même avec deux sites, le débit chute de 450 messages par seconde à 140 ou 160 suivant le placement du séquenceur. Mais il est surtout intéressant de noter que les performances se dégradent très rapidement avec l'arrivée de nouveaux sites dans le cas où le 386 assure le séquencement, alors que la décroissance reste limitée si c'est une machine rapide qui est séquenceur. Donc il est important en pratique de s'assurer qu'une machine rapide assure la fonction de séquenceur. Cela valide l'option qui a été choisie de sélectionner comme nouveau séquenceur la machine la plus rapide parmi les sites de stockage survivants, lorsqu'une phase de resynchronisation a lieu.

Les résultats pratiques s'expliquent très bien, car lorsque le groupe comporte N sites de stockage, le séquenceur doit traiter la réception de N messages (en moyenne) avant de pouvoir émettre le numéro de séquence. Parmi ces messages, N − 1 sont des acquittements, donc de petite taille, mais la charge supplémentaire reste importante. Dans la deuxième série de mesures, le 386 reste le facteur limitant le débit au sein du groupe, mais son influence reste identique avec l'augmentation du nombre de membres car sa charge de travail est

indépendante de ce facteur. Cela explique que le débit reste quasiment constant dans le deuxième test.

V.3.4 Nombre de membres

Toutes les expériences précédentes se contentaient d'un nombre de membres limité, le groupe étant restreint aux seuls sites de stockage. Les mesures suivantes ont pour but d'évaluer l'évolution du débit avec l'adjonction de nouveaux sites. Le nombre de sites de stockage est volontairement limité à 5, séquenceur inclus, tous les autres sites sont donc anonymes − en fonction du principe des groupes opaques. De plus pour accentuer l'impact de l'augmentation du nombre de sites, nous avons volontairement placé le séquenceur sur une machine lente (le 386DX25), puis ajouté les sites un−à−un par ordre de puissance croissante. La méthode de test utilisée, telle que décrite dans l'annexe B, assure que tous les sites prennent une part active à la génération du trafic circulant au sein du groupe.

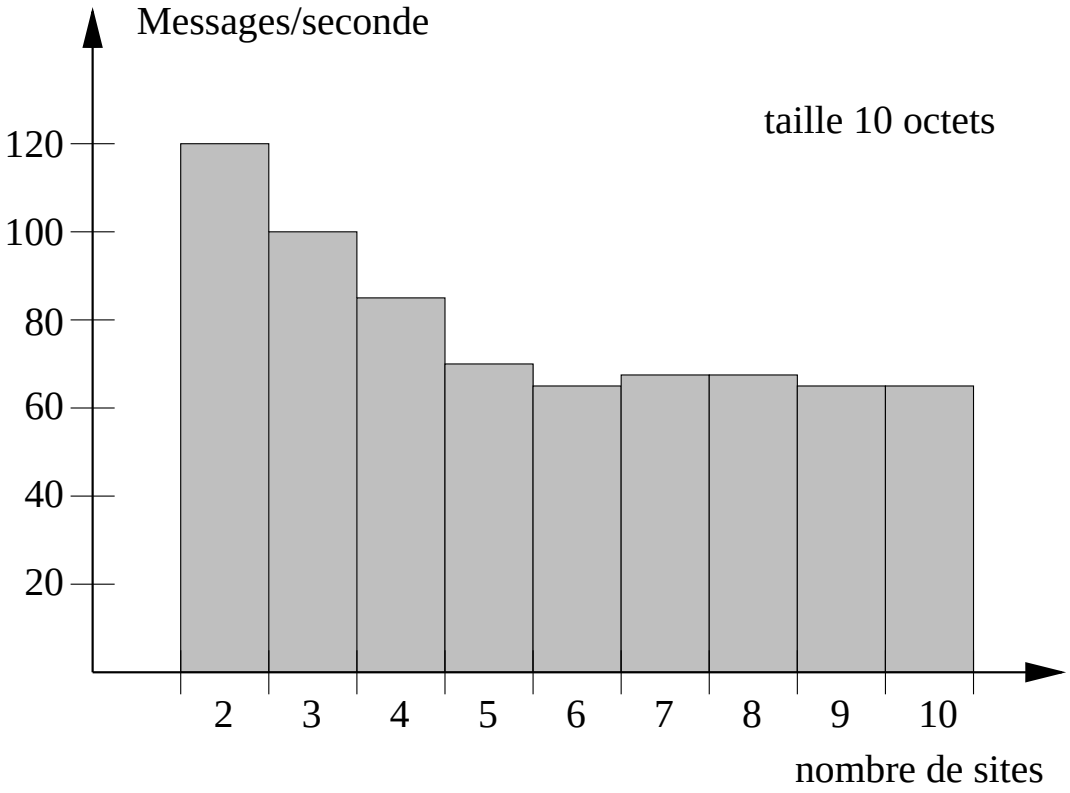


Fig. 5.9 : Évolution des performances avec le nombre de machines

Le point remarquable de ces résultats est l'absence de surcharge lorsque de nouveaux sites rejoignent le groupe. Le débit décroît dans un premier temps jusqu'à la complétude du quorum chargé d'assurer la séquencement et le stockage, c'est−à−dire pour les cinq premiers sites. Les sites suivants sont anonymes, et excepté pour l'émission de message, leur rôle est purement passif. Tous les messages qui leur sont destinés sont transmis en diffusion, et le coût de fonctionnement du groupe n'augmente pas. La seule surcharge éventuelle

est liée aux retransmissions dans les cas de perte. Statistiquement, la charge liée à la réémission de messages perdus augmente linéairement avec le nombre de sites connectés au groupe, mais dans des conditions normales de fonctionnement cela ne doit pas dégrader les performances générales du protocoles.

V.3.5 Problème du contrôle de flux

L'utilisation de groupes opaques et leur implantation à l'aide de la diffusion physique offerte par le médium de transmission se révèle très efficace comme le montrent les mesures présentées dans cette section. Toutefois, le principe de base des groupe opaques, à savoir le fait de ne pas gérer une liste des sites au sein du protocole, empêche toute rétroaction des sites (du moins des sites anonymes) sur le fonctionnement global du groupe. En particulier cela empêche d'effectuer un contrôle de flux global sur l'ensemble des machines. Cela se traduit en pratique par des cas où la connexion d'un nouveau site anonyme à un groupe existant se révèle impossible. Si la charge imposée par l'ensemble des membres du groupe est supérieure aux capacités de traitement du nouveau participant, celui−ci ne pourra pas absorber le débit et se déconnectera automatiquement après quelques secondes.

La méthode théorique pour éviter ce type de congestions consiste a maintenir un paramètre de fonctionnement qui est la valeur minimale des débits maximaux supportés par chaque clients du groupe, et à limiter − par exemple au niveau du site séquenceur − le débit du groupe en−deçà de cette valeur limite. L'implantation de ce type de contrôle est non trivial pour des protocoles point−à−point, et devient ardu pour des protocoles de groupes, car le débit maximal acceptable pour un site est fonction de la charge sur celui−ci et varie donc au cours du temps. Il faut donc collecter ces informations à intervalles réguliers, ainsi qu'à l'arrivée et au départ de chaque membre du groupe.

Ce type d'algorithme n'est pas réalisable dans le cas de groupes opaques, et nous préconisons plutôt une approche pragmatique consistant à toujours maintenir parmi les sites de stockage un site représentant le type de machine le moins puissant susceptible de se connecter au groupe. En effet, un contrôle de flux s'exerce naturellement au sein du quorum, basé sur la délivrance des acquittements en direction du séquenceur. Y maintenir une machine du type le moins puissant garantit la fiabilité des connexions même sous fortes charges, au détriment toutefois d'un débit de crête inférieur, même si tous les autres sites connectés ont la capacité de supporter un plus fort flux. D'après les résultats exposés aux V.3.3 il est préférable que le site modérateur ne supporte pas la charge du séquenceur.

V.3.6 Conclusion sur les expérimentations

Les résultats et les analyses présentées dans les deux sections précédentes mettent en évidence les bons résultats pratiques du protocole. Cela conforte le protocole déjà implanté dans le système Amœba comme étant un des plus rapides, mais cela valide surtout l'adjonction théorique du fonctionnement en groupe opaque et les choix d'implantation du protocole depuis le mode utilisateur et l'utilisation de la diffusion physique offerte par Ethernet. On peut d'ailleurs noter que les performances d'une implantation d'un protocole de groupe en mode utilisateur chuteraient très rapidement sans l'utilisation du mécanisme de diffusion.

En effet, dans ce cadre chaque émission et réception de paquet reste relativement coûteuse et adresser chaque participant l'un après l'autre entraînerait une surcharge, proportionnelle au nombre de sites, très handicapante.

V.4 Validation des optimisations

Les optimisations suggérées aux chapitres II et III ont été introduites au fur et à mesure du développement du protocole pour limiter des situations de blocage momentanées lors de la perte de messages. Le but était alors plus de faciliter le développement sur le simulateur que d'améliorer le service dans les cas de réseaux ou de stations surchargées. A l'expérience, ces modifications se sont révélées efficaces lors des tests sur notre réseau local, la fluidité du débit augmentant nettement lors de la mise en place de chaque optimisation. Dans cette section nous présentons la méthode adoptée pour quantifier l'effet de ces modifications apportées au protocole initial, et présentons les résultats obtenus.

V.4.1 Méthode de mesure

Tester un protocole dans des conditions de fonctionnement anormale est difficile : s'il est aisé de simuler une charge sur les différents sites mis en œuvre, il est très compliqué de recréer artificiellement des problèmes liés au réseau. Nous avons donc décidé de mesurer l'impact d'un fonctionnement dégradé du réseau sur le protocole à l'aide de l'émulateur présenté au chapitre IV, en effet il permet de simuler des taux de perte en émission et en réception arbitraires. La méthode adoptée pour les mesures présentées dans cette section a consisté en une double exécution du programme de simulation, le premier compilé avec la version initiale du protocole, alors que le second comportait toutes les optimisations ajoutées par la suite. Le comportement des sites clients est le même que celui décrit au V.1. Les tests ont consisté à faire varier indépendamment les taux de perte en émission et en réception pour des valeurs comprises entre 1/2 et 1/8196, ce qui couvre l'ensemble probable des situations depuis un réseau franchement inutilisable jusqu'à des conditions normales. Les taux de perte trop importants peuvent faire échouer la simulation, ce qui se manifeste par des phases de resynchronisation du groupe se succédant sans arrêt, la délivrance de messages n'a alors lieu qu'épisodiquement. Dans le cas contraire la simulation se termine sur chaque site après la délivrance de 2500 messages au client, et diverses mesures locales sont collectées :

- le nombre de message émis normalement par le site (MSG_SEND).
- le nombre de messages délivrés localement.
- le nombre de messages demandés, c'est−à−dire lorsque l'on reçoit une estampille globale sans que le message correspondant ne soit trouvé sur le site, ce qui nécessite de demander celui−ci à un des sites de stockage.
- le nombre de messages perdus, c'est le cas uniquement sur le séquenceur lorsqu'il reçoit un acquittement d'un message qui a été perdu.
- le nombre de retransmissions (MSG_RESEND), qui ont lieu lorsqu'un message a été déjà été émis mais que sa délivrance a échoué.

- le nombre de phases de resynchronisation vues par le site depuis sa connexion au groupe.

L'ensemble des valeurs des taux de pertes examinées sont 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8 1/10, 1/16, 1/32, 1/64, 1/128, 1/256, 1/512, 1/1024, 1/2048, 1/4096 et 1/8192, ce qui a conduit à 648 simulations au total. Chaque simulation a été réalisée avec 10 sites, répartis en 5 sites anonymes et 5 sites de stockage.

V.4.2 Influence sur la stabilité du protocole

La figure suivante montre les zones de stabilité du protocole sans optimisation, l'abscisse et l'ordonnée du graphe indiquent avec une échelle logarithmique l'inverse de la probabilité de perte en émission et en réception respectivement. Chaque croix correspond au succès de la simulation pour ces paramètres.

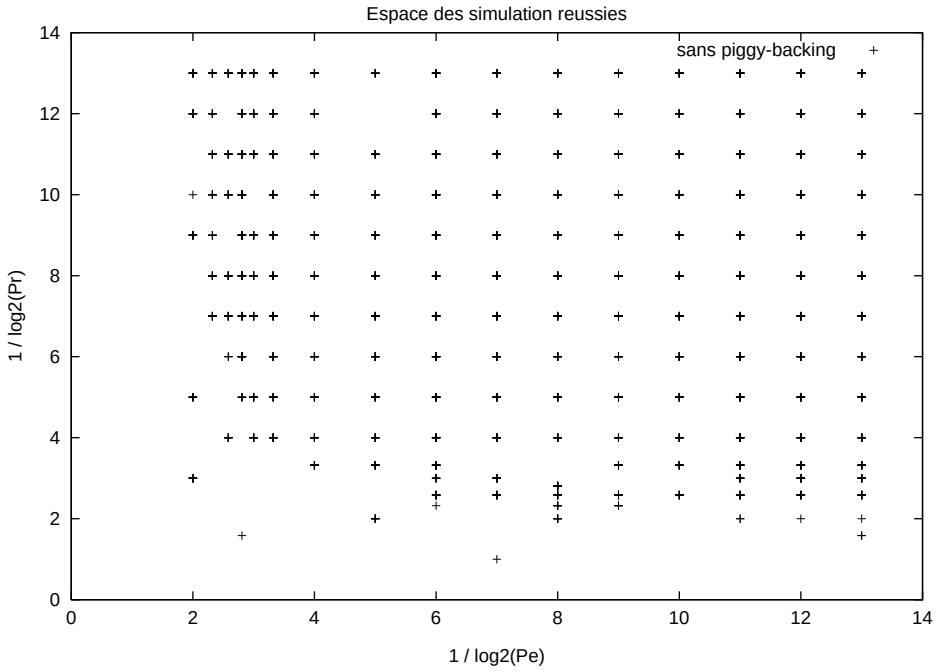


Fig. 5.10 : Stabilité du protocole non optimisé

De manière générale, le protocole offre un fonctionnement correct si les taux de perte en émission et en réception sont inférieurs à 1/10. L'effet des pertes en réception est plus accentué que celui lié à la non−émission des paquets, ce qui confirme les résultats théoriques obtenus au chapitre .

La figure suivante montre les résultats obtenus avec la version améliorée du protocole :

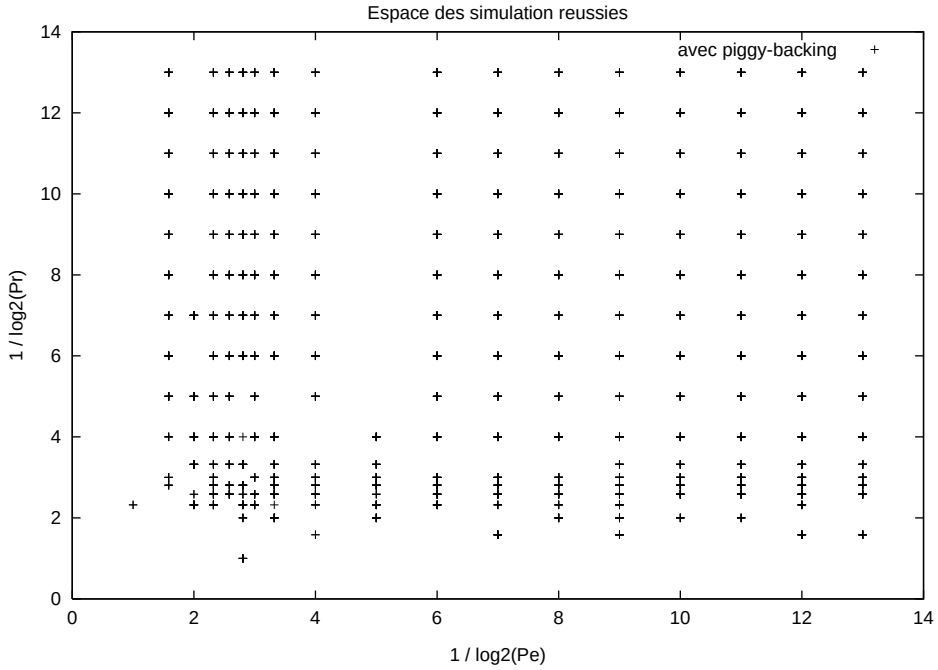


Fig. 5.11 : Stabilité du protocole optimisé

On remarque immédiatement le gain apporté par les diverses optimisations, en particulier le protocole souffre beaucoup moins lors de pertes de messages en réception. Excepté de mauvais résultats sur la colonne correspondant à des pertes en émission de 1/32, un artefact lié au fonctionnement interne de l'émulateur, le protocole est désormais stable si les taux de pertes en émission et en réception sont inférieures à 1/5. Le protocole se comporte aussi nettement mieux lorsque ces deux facteurs augmentent simultanément (lorsque l'on se rapproche de l'origine du graphe).

V.4.3 Influence sur les redemandes

Dans cette sous−section nous examinons l'impact des transformations faites au protocole sur les redemandes. Ces requêtes sont émises depuis un site lorsqu'une association entre estampille globale et estampille locale est reçue, et que ce message ne peut être délivré pour plusieurs raisons :

- soit l'estampille globale reçue ne correspond pas à celle qui doit être délivrée en séquence. Dans ce cas un message de redemande est émis pour toutes les associations manquantes. Par exemple si l'estampille globale 5 est reçue, et que le site n'a délivré les messages que jusqu'à 2 inclus, un message de redemande sera émis à destination des sites de stockage pour les estampilles 3 et 4.
- soit le message n'est pas trouvé, c'est−à−dire que la recherche dans le buffer des messages reçus mais non délivrés de l'estampille locale a échoué. Cela signifie en pratique que le message émis par le site de départ n'a pas été reçu sur ce site, auquel cas un message de redemande est envoyé vers les sites de stockage pour en recevoir une copie.

Pour analyser le taux de redemandes, la métrique utilisée consiste à diviser le nombre total de redemandes effectuées sur chaque site au cours d'une simulation par le nombre de messages effectivement délivrés localement. La figure suivante montre les résultats obtenus sur l'ensembles des simulations parvenues à leur terme pour la version initiale du protocole :

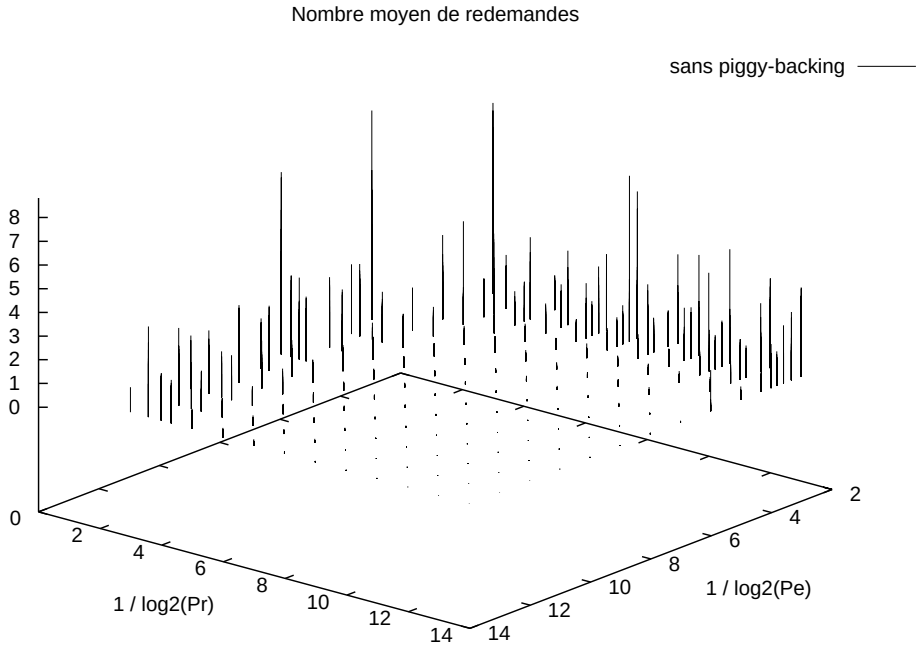


Fig. 5.12 : Taux de redemandes pour le protocole original

On constate que si le taux de redemandes est très faible dans de bonnes conditions, il augmente rapidement lorsque les pertes en réception ou en émission atteignent des valeurs supérieures à 1/20. A la frontière de la zone de stabilité du protocole, on constate parfois que les clients redemandent jusqu'à huit fois en moyenne les informations avant de pouvoir les délivrer, ce comportement anormal amplifie les problèmes déjà dûs aux mauvaises conditions de fonctionnement du réseau. Le graphe suivant illustre les résultats obtenus avec la version modifiée du protocole :

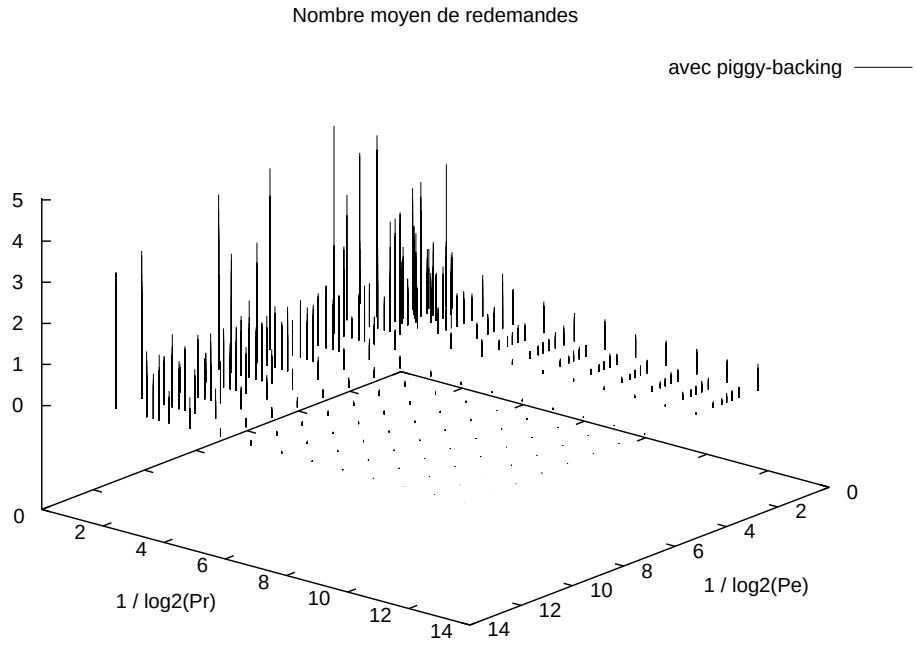


Fig. 5.13 : Taux de redemandes après optimisations

L'amélioration obtenue par rapport à la version initiale est immédiatement visible :

- le taux de demandes ne dépasse jamais 5 au lieu de 8 précédemment, et les valeurs supérieures à 1 ne sont que rarement atteintes.
- l'augmentation des pertes en émission n'a plus qu'une influence modérée.

Le gain est principalement dû au mécanisme de piggy−backing des associations entre estampille globale et estampille locale. Ce mécanisme assure que les cas de redemandes sont alors quasi systématiquement dûs à la perte en réception du message initial. L'influence des pertes en émission sur les redemandes est limitée au message final (MSG_DELIVER) − en effet toutes les autres pertes en émission bloquent la génération de l'estampille globale − or le contenu de ce message de contrôle est dupliqué dans tous les autres messages du protocole. Cela explique l'amélioration radicale constatée lorsque les pertes en émission augmentent.

V.4.4 Influence sur les réémissions

Nous examinons dans cette sous−section l'impact des changements apportés au protocole sur le nombre moyen de réémissions. Une réémission est effectuée par un site émetteur lorsqu'il considère qu'un message à destination du groupe émis précédemment ne sera pas délivré. Cela survient si le site émetteur ne reçoît pas le message de délivrance (MSG_DELIVER) en provenance du séquenceur après un délai de garde, ou si le séquenceur lui notifie que l'émission précédente a échoué (MSG_FAIL).

Pour mesurer le taux de réémissions, nous avons divisé le nombre de messages en réémission (MSG_RESENT) émis depuis chaque site lors d'une simulation réussie par le nombre de messages transmis par ce même site (MSG_SENT). La figure suivante donne les résultats obtenus pour la version initiale de protocole :

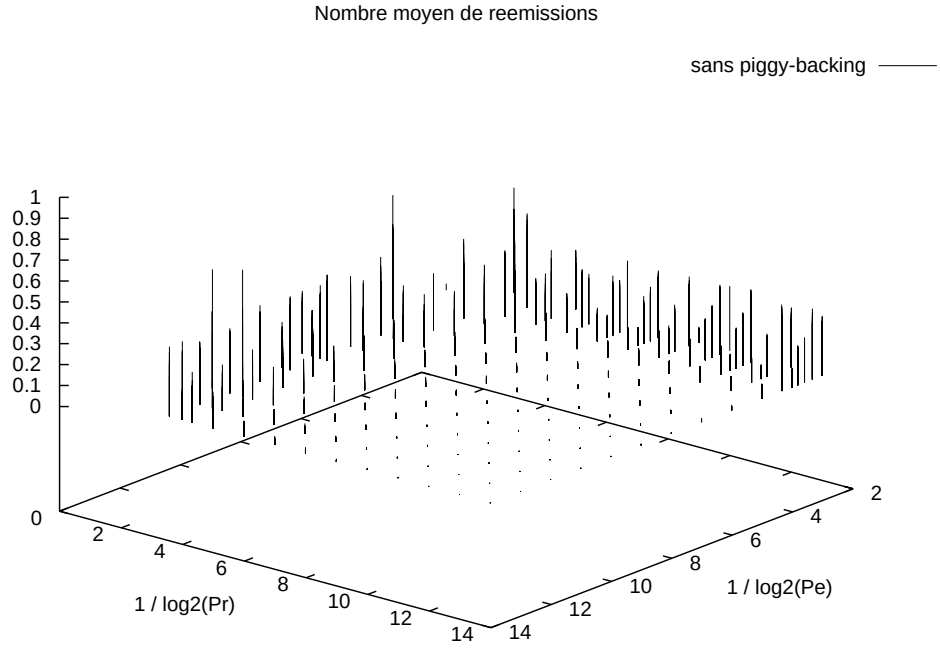


Fig. 5.14 : Taux de réémissions pour la version initiale du protocole

Contrairement au taux de redemandes, un grand nombre de réémissions n'est pas fréquent. C'est principalement dû à la mise en place d'un mécanisme de veille qui déclenche une phase de resynchronisation s'il y a accumulation de messages en émission sur un site et que les messages de délivrance associés tardent trop à arriver. C'est le principe de base utilisé pour détecter la mort ou la surcharge du séquenceur. De ce fait on n'observe pas de simulation considérée comme réussie si le taux de réémission dépasse 1, car un taux de redemande élevé entraîne de fréquentes resynchronisations qui finissent par faire échouer la simulation. On constate aussi que la baisse du nombre de réémissions lorsque les conditions s'améliorent n'est pas très franche. Par contre une fois que les pertes en émission ou en réception sont inférieures à une pour 50, les réémissions sont très rares, seul un point représente la mesure sur le graphe.

Voici les résultats obtenus avec la version modifiée du protocole :

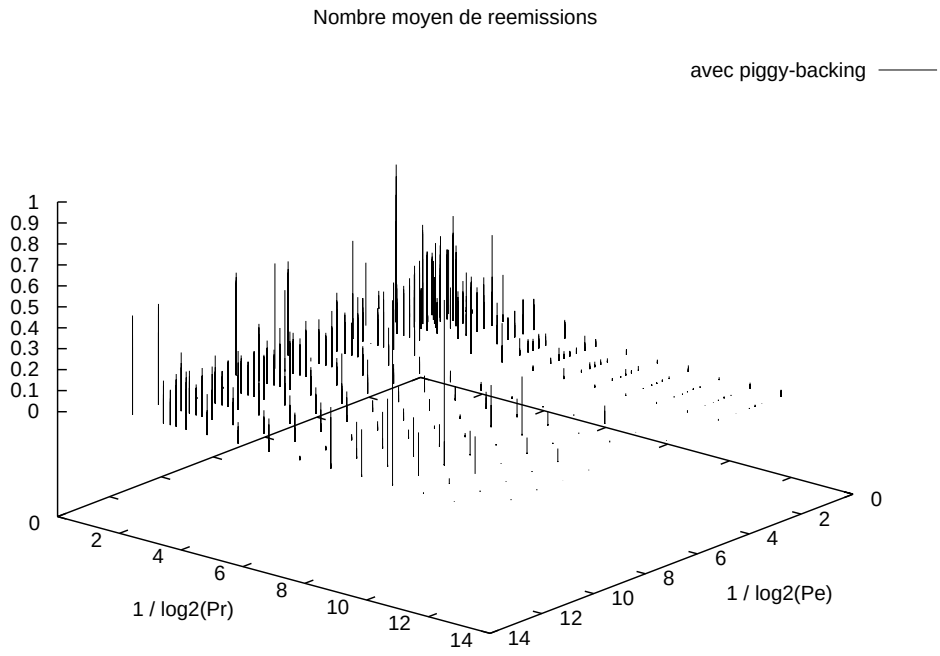


Fig. 5.15 : Taux de réémissions après modification du protocole

On constate une très nette amélioration des taux moyens de réémission dans les cas de mauvaises conditions de transmissions. Tout comme pour les redemandes, les progrès les plus notables sont obtenus lors de forte pertes en émission. C'est principalement dû à la mise en place d'un mécanisme de piggy−backing des acquittements sur tous les messages en provenance des sites de stockage et destinés au séquenceur. De cette manière la perte d'un acquittement ne bloque plus la délivrance de l'estampille globale associée au message, et la réémission d'un message n'est plus nécessaire que lorsque l'un des site de stockage ou le séquenceur n'a pas reçu le message initial. Cette dernière condition peut survenir si le site émetteur n'arrive pas à émettre le message initial ou si une perte survient lors de la réception du message sur le site séquenceur ou un site de stockage. L'impact des pertes en émission sur les retransmissions se limite désormais au seul cas de l'envoi du message initial.

Toutefois, la partie du graphe représentant les bonnes conditions semble s'être dégradée. En effet, on observe aléatoirement des valeurs non proches de zéro dans la plage représentée par des taux de pertes en réception inférieurs à 1 pour 50, alors que toutes les mesures obtenues pour le protocole initial étaient parfaites dans cette zone. Ces irrégularités trouvent leur source dans la conjonction d'une des optimisations faites et des spécificités de l'implantation du simulateur. Celui−ci émule tour à tour les différents sites : la sélection d'un nouveau site avant un changement de contexte est tiré au hasard pour éviter des phénomènes annexes qui peuvent se produire si on utilise un ordre fixe de séquencement des sites. De ce fait, un acquittement peut être transmis au séquenceur avant même que celui−ci n'ait reçu le message initial ; la figure suivante illustre ce cas :

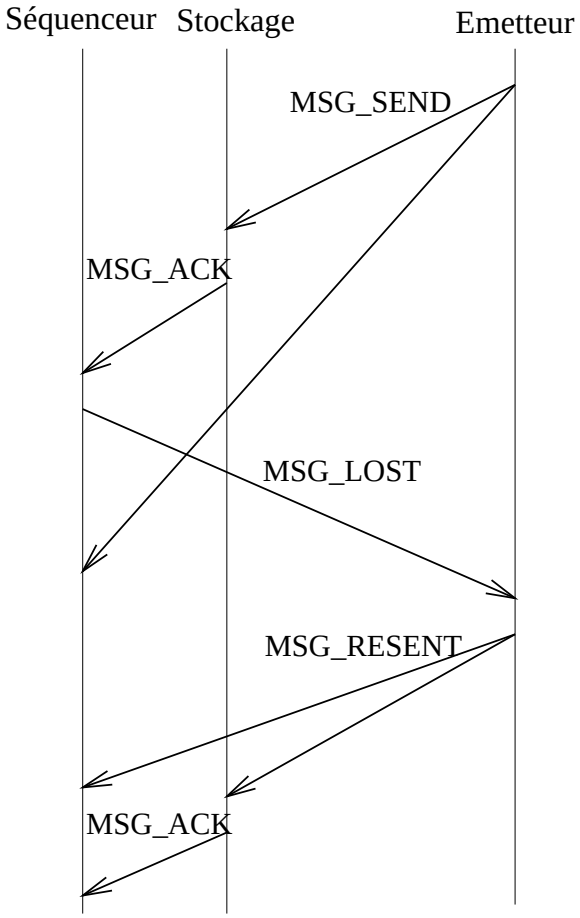


Fig. 5.16 : Cas d'inversion de messages

Ce problème ne se rencontre pas en pratique, sauf lors des expérimentations au−dessus du noyau Mach, où le message initial émis en diffusion est considéré comme moins prioritaire que les acquittements transmis en point−à−point. L'optimisation qui pose problème dans ce cas est celle qui lors de la réception d'un acquittement par le séquenceur non précédée par la réception du message initial, le séquenceur envoie au site émetteur un message de perte (MSG_LOST) pour l'informer que son message ne peut pas être délivré. Cette décision est justifiée par le fait que cet acquittement va être considéré comme perdu, donc l'estampille globale ne pourra pas être délivrée. L'optimisation a pour but d'améliorer la réactivité du protocole lors de pertes de messages MSG_SEND sur le site séquenceur.

Lors d'une inversion au cours d'une simulation, la version initiale du protocole n'effectue la réémission qu'après l'attente du délai de garde, alors que la version optimisée va provoquer une réémission immédiate. De ce fait l'impact des inversions sur le nombre de réémissions effectuées lors d'une simulation augmente dans le deuxième cas. Les inversions sont rares en pratique car Ethernet utilise un médium commun et si deux paquets consécutifs sont bien reçus sur un site, l'interface les traite dans le même ordre. L'inversion ne peut se produire que si le système d'exploitation sous−jacent accumule des paquets puis les délivre dans un ordre différent de celui d'arrivée.

V.5 Conclusion

L'analyse des mesures menée dans ce chapitre et concernant la mise en œuvre du protocole a validé les choix d'implantation du protocole, à savoir :

- l'utilisation du mécanisme de diffusion physique offert par le réseau local Ethernet.
- la mise en place du protocole sous la forme d'un serveur s'exécutant en mode utilisateur au sein d'un processus Unix.

L'utilisation de la diffusion a permis d'obtenir de bonnes performances, suffisantes pour servir de mécanisme de base pour la construction de systèmes répartis ou d'applications interactives et coopératives. Le léger surcoût dû à l'implantation en mode utilisateur est provoqué par une ou deux recopies en mémoire et aux mécanismes systèmes liés aux transitions entre noyau et espace utilisateur. Cette surcharge tend à disparaître avec l'évolution technologique des processeurs.

Ces mesures ont aussi permis de valider les diverses optimisations faites au protocole original, qui augmentent nettement sa résistance lorsque les pertes liées à la surcharge du réseau ou à la saturation des sites se multiplient. Enfin nous avons pu dégager des règles importantes concernant la répartition des tâches sur les divers sites, en fonction de la puissance de calcul disponible localement.

Conclusion

Le concept de groupe de processus communicant permet de structurer les mécanismes de coopération au sein de systèmes répartis. Notre travail a porté sur les protocoles de diffusion qui fournissent les primitives de transport d'information au sein de tels groupes. C'est donc un des éléments de base permettant de réaliser aisément des systèmes distribués ou des applications coopératives.

Un grand nombre de protocoles de diffusion existent, basés sur plusieurs sémantiques de délivrance en fonction de l'ordonnancement, de leurs propriétés de résistance aux pannes, ou des hypothèses faites sur les caractéristiques du réseau de communication utilisé. De nombreux projets de recherche se sont attachés à définir et à réaliser de tels protocoles. La construction d'environnements de développement permettant de bâtir aisément des protocoles ayant des caractéristiques données reste un sujet de recherche actif. D'autre part, l'avènement des réseaux rapides et la possibilité de communiquer de façon interactive à longue distance fournit de nouveaux thèmes de réflexion. En particulier les problèmes liés au facteur d'échelle, qu'il soit géographique avec l'allongement des distances ou qu'il soit lié à l'augmentation du nombre de sites participant aux communications, nécessitent de développer de nouvelles abstractions et des mécanismes en adéquation à cette expansion.

Ce travail a consisté à concevoir et implanter un protocole de diffusion pour réseaux locaux. Dans un premier temps, nous avons étudié le principe de groupe opaque dans lequel la liste des participants ne peut être déterminée. Seule la composition du quorum formé des quelques sites prenant une part active dans le déroulement du protocole est disponible. Nous avons adapté le protocole de diffusion atomique initialement implanté dans le système Amœba pour supporter les groupes opaques. L'utilisation du mécanisme de diffusion physique fourni par un réseau Ethernet est rendue nécessaire pour transférer de manière aveugle les informations à tous les sites abonnés à un groupe. Un tel protocole est plus à même de supporter le facteur d'échelle lié à l'augmentation du nombre de sites, car il ne nécessite pas le maintien d'une liste cohérente de tous les sites connectés, ce qui constitue un des principaux problèmes tant algorithmique que pratique pour les protocoles existants. Des optimisations au protocole initial sont aussi suggérées, principalement basées sur le mécanisme de piggy−backing.

Lors de la phase de mise en œuvre, nous avons d'abord isolé les interfaces systèmes fondamentales permettant de construire des protocoles portables. Une bibliothèque offrant tous les types de base facilitant la construction de protocoles a été bâtie au dessus de cette interface minimale. Elle comprend les opérations essentielles d'envoi et de réception de messages typés de taille quelconque, des primitives évoluées de gestion de la mémoire, une bibliothèque portable de coroutines, et le support de délais de garde. La portabilité de l'ensemble a été testée en mode utilisateur dans une tâche Mach, dans le noyau Mach, et en mode utilisateur sur diverses plate−formes Unix standard (Ultrix, Linux, SunOs et AIX). Nous pouvons donc conclure qu'il est possible de concevoir des protocoles portables

s'exécutant en mode utilisateur, et utilisant des ressources de bas niveau, même si les systèmes d'exploitation utilisés sont "propriétaires". Il suffit de se restreindre à un petit jeu de primitives universelles mais dont les interfaces sont souvent spécifiques à chaque constructeur.

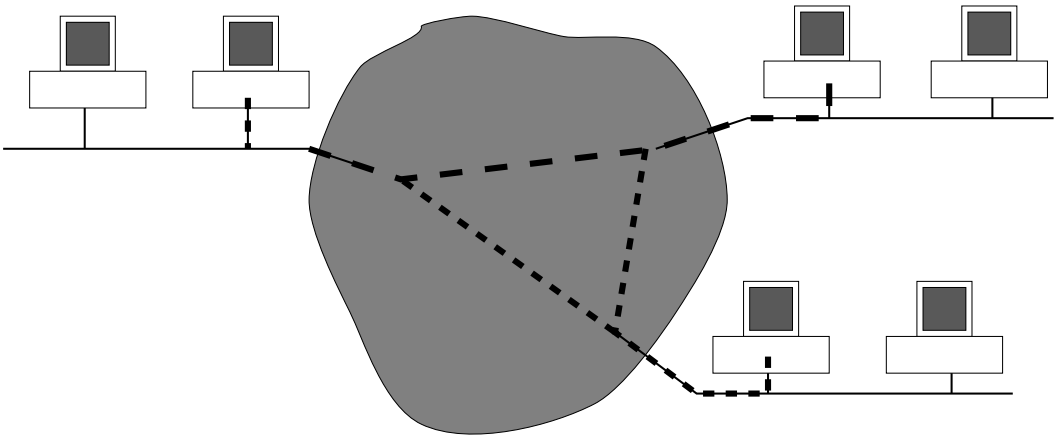
Deux protocoles ont été implantés au dessus de cette bibliothèque. Le premier est un protocole point−à−point fournissant un service de transport fiable, basé sur le principe de fenêtre d'acquittement glissante. Le second est un protocole de diffusion atomique, résistant aux pannes d'un nombre borné de sites, et s'appuyant sur le principe des groupes opaques. La réalisation a été grandement facilitée par la construction d'un émulateur de réseau local Ethernet fonctionnant dans un processus Unix et sur lequel la bibliothèque générique a été portée. Un ensemble d'outils de déboggage de haut niveau est fourni qui a permis de développer et de tester les protocoles en s'affranchissant des contraintes liées au caractère distribué de ce type de programmes. L'émulateur permet aussi de créer à la demande des conditions de trafic perturbé, situation difficile à obtenir en pratique : les protocoles peuvent donc être testés et optimisés pour ces cas limites.

Un ensemble de mesures ont été effectuées sur le protocole de diffusion. Nous avons dans un premier temps vérifié les bonnes performances de celui−ci : des débits de 400 à 500 diffusions fiables par seconde sont usuellement obtenus lors d'une utilisation mettant en œuvre plusieurs stations de travail Unix. Nous avons pu vérifier l'innocuité de l'utilisation de la diffusion physique et le gain en performance qu'elle permet d'obtenir. En particulier, les performances ne dépendent plus du nombre de stations connectées au groupe, mais uniquement du nombre de sites assurant la résistance aux pannes. L'étude de l'influence de divers paramètres illustre l'importance du placement du séquenceur sur une machine rapide, pour éviter qu'il ne devienne le goulot d'étranglement limitant le débit. Ces tests ont aussi mis en évidence l'impossibilité théorique de fournir un contrôle de flux au sein de groupes opaques, mais ce problème peut être contourné en limitant le flux au niveau du quorum.

L'ensemble des temps d'exécution des couches logicielles utilisées ont été mesurés, depuis les routines d'interruption associées au périphérique Ethernet, jusqu'au protocole en mode utilisateur. Elles ont été effectuées sur le système Linux et pour toute la gamme des processeurs Intel©. Une étude fine des temps d'exécution du protocole est donc fournie, en particulier l'impact de l'expatriation du protocole hors du noyau et les coût associés en terme de changements de contexte, de recopies en mémoire et de transfert du contrôle entre espaces utilisateur et noyau sont analysés. Elle montre que le surcoût engendré par la réalisation du protocole en mode utilisateur reste faible, malgré l'inadéquation des primitives d'Unix. On observe aussi que le coût des opérations liées au passage entre mode utilisateur et mode noyau s'est plus amélioré avec les nouvelles générations de processeurs que la moyenne du jeu d'instruction, diminuant encore le coût relatif de la réalisation de services hors du noyau. Les résultats sont donc en faveur des nouvelles architectures de systèmes, où la majorité des services sont implantés sous la forme de tâches séparées au dessus d'un micro−noyau.

Le principe des groupes opaques permet de simplifier la définition théorique de la sémantique d'un protocole, mais il nécessite d'avoir un mécanisme de diffusion physique sous−jacent. Le problème de mise à l'échelle se réduit alors à un problème de distance, car il

n'existe pas de telle primitive en dehors des réseaux locaux ou métropolitains. Un sujet de recherche ouvert est celui qui permettrait d'étendre le principe de la diffusion à un ensemble de réseaux locaux. L'utilisation de connexions fiables basées des protocoles standards tels que TCP/IP semble raisonnable, un routeur présent sur chaque sous−réseau étant chargé de diffuser les informations vers les autres routeurs, la diffusion physique restant utilisée sur chaque sous−réseau. La gestion des membres est ramenée à celle d'un ensemble de sous−réseaux et des routeurs associés.



On peut distinguer deux cas de figure : soit tous les membres du quorum sont situés sur un même sous−réseau auquel cas il suffit de propager de manière fiable aux autres réseaux la succession ordonnée des messages prêts à être délivrés, soit les membres du quorum sont distribués sur un ensemble de sous−réseaux et les acquittements doivent être transmis via les routeurs vers le séquenceur. On peut s'attendre à une baisse sensible des performances dans le deuxième cas, à cause de latences induites par la propagation des messages, le gain étant une meilleure disponibilité lors des ruptures de connexions entre sous−réseaux

Un autre sujet d'étude est la réalisation d'interfaces standardisées permettant de développer des protocoles spécifiques indépendamment du système d'exploitation existant et donnant un accès complet aux possibilités des nouvelles interfaces à haut débit tel qu'ATM. Malgré l'augmentation régulière de la puissance des processeurs, construire des interfaces matérielles et logicielles configurables, au sens où elle permettent d'implanter des protocoles non fournis par l'interface, et supportant des débits approchant le gigabit par seconde reste un problème ardu, voire impossible.

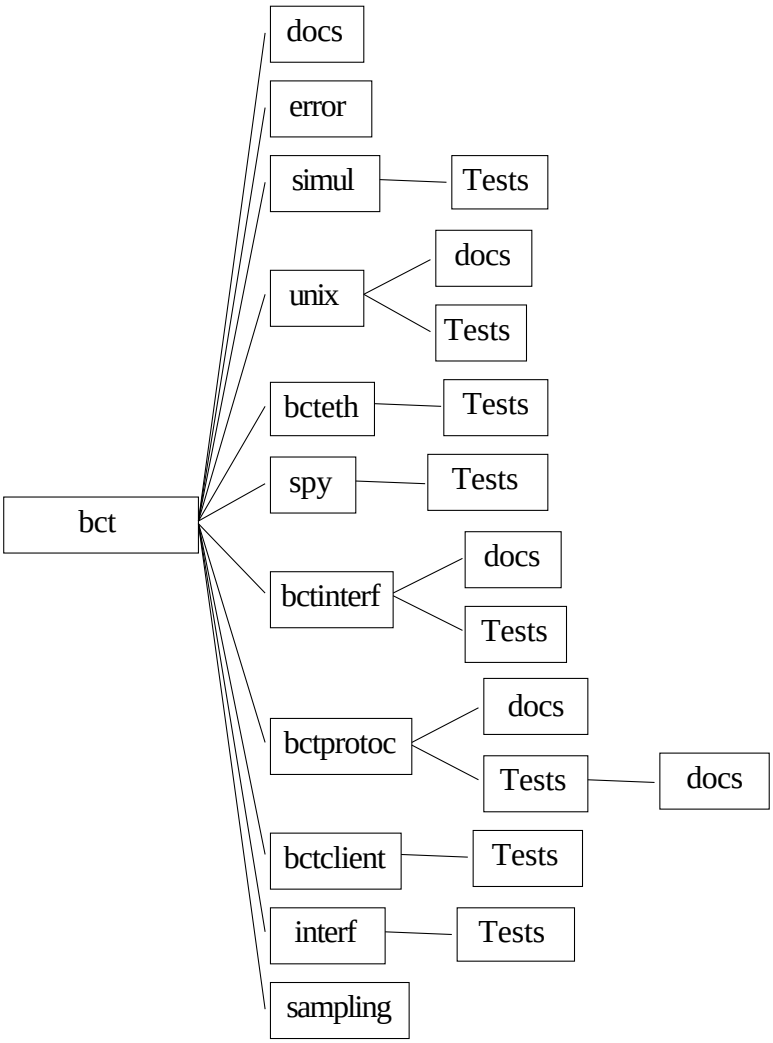
De fait, la standardisation des interfaces semble un phénomène irréversible, pour des raisons de coûts de développement et d'interopérabilité. L'absence de standard concernant les protocoles permettant la diffusion d'information se traduit par l'utilisation de protocoles ad−hoc bâtis au sein même des applications à l'aide des protocoles point−à−point disponibles. Il peut en résulter une perte de performances notable, en particulier si les mécanismes de diffusion physique disponibles ne sont pas utilisés, et la multiplication des protocoles existants. Il se pourrait que la mise en œuvre d'applications coopératives reste longtemps un sujet d'avenir en l'absence d'un standard de communication spécifique. La décision de normaliser au sein d'ATM des protocoles de diffusion permettra peut−être de réaliser les applications supportant le travail coopératif.

Annexe A : Le code source

Cette annexe fournit des informations sur l'organisation du code réalisé, son volume et des informations statistiques sur le code C produit durant cette thèse :

1 Organisation du source

L'arborescence du source suit le schémas suivant, inspiré des normes de programmation du prototype Guide−2 :



Chaque sous arborescence traite un composant logique de l'ensemble logiciel, et est constitué généralement d'un répertoire de sources et d'un sous−répertoire comportant les programmes de test. Si nécessaire un sous−répertoire comprend des figures ou de de la

documentation sur le fonctionnement du composant et ses interfaces. Chaque répertoire comprend aussi un sous répertoire référencant la base RCS utilisée pour la gestion de versions. De plus les objets résultant de la compilation pour un type de machine et une configuration spécifique sont stockés dans un sous−répertoire spécifique.

Nous détaillons ci−dessous le contenu de chaque composant :

- simul : comprend l'émulateur Ethernet utilisé lors du déverminage. Il est formé d'un noyau de thread, un émulateur Ethernet, un dispositif de gestion d'événements, et d'un déboggeur.
- unix : offre l'accès au réseau Ethernet et implante les threads dans le cadre de l'utilisation du protocole au dessus d'Unix.
- bcteth : routines d'interface réseau, thread et synchronisation lors de l'utilisation dans une tâche Mach ou lors de l'intégration du protocole dans le micro−noyau.
- spy : routines d'analyse et d'affichage des informations greffé au dessus de l'interface réseau lors de l'utilisation d'une station espionnant le protocole.
- bctinterf : implantation de la couche générique au dessus des diverses interfaces (Mach, Unix ou simulateur). Offre la fragmentation et le réassemblage des messages, les buffers, l' interface de processus légers, et une gestion de mémoire unifiée.
- bctprotoc : réalisation du protocole de diffusion et du protocole point−à−point.
- bctclient : interface avec les processus clients du protocole.
- interf : module comprenant des widget et le code d'interface vers un interprète Tcl/Tk embarqué. Est utilisé pour l'affichage des information lors des phases d'émulation sous Unix ou l'analyse via un site espion.
- sampling : module d'analyse temporelle basé sur des appels à la routine gettimeofday() et fonctionnant indifféremment en mode utilisateur ou dans le noyau de Linux.

Le résultat de la commande Unix du(1) permet d'avoir un aperçu du volume nécessaire à l'arborescence :

```
32      bct/interf/Tests
250     bct/interf
97      bct/bctclient/Tests
231     bct/bctclient
1       bct/spy/Tests
208     bct/spy
43      bct/unix/Tests
16      bct/unix/docs
359     bct/unix
22      bct/docs
60      bct/simul/Tests
644     bct/simul
78      bct/error
95      bct/bcteth/Tests
```

```
147     bct/bcteth
149     bct/bctinterf/Tests
36      bct/bctinterf/docs
784     bct/bctinterf
6       bct/bctprotoc/Tests/docs
522     bct/bctprotoc/Tests
14      bct/bctprotoc/docs
1354    bct/bctprotoc
86      bct/sampling
4178    bct
```

La section suivant fournit une analyse statistique sur l'ensemble du code source réalisé.

2 Analyse statistique du code

Les résultats suivant sont les résultats obtenus après l'analyse du code source par l'utilitaire C−count version 0.7 disponible dans les archives du GNU. Chaque ligne indique le nombre de lignes et d'instructions C correspondant à un fichier. Le nombre d'instructions est compté suivant la méthode du point−virgule.

```
 107    15   |bctclient/clnt_server.h
  75     2   |bctclient/error.h
  87    11   |bctclient/extern.h
  74     0   |bctclient/global.h
 147    10?  |bcteth/endian.h
  80     2   |bcteth/error.h
 140    27   |bcteth/ethernet.h
  82     0   |bcteth/extern.h
  80     0   |bcteth/global.h
  46     1   |bcteth/mach_error.h
  58     5   |bcteth/testnset.h
 584   125?  |bctinterf/R_machinter.h
 239     2   |bctinterf/error.h
 379    65   |bctinterf/extern.h
 131     0   |bctinterf/global.h
 186    19   |bctinterf/madata.h
 163     7   |bctinterf/mathread.h
  96     2   |bctinterf/memory.h
  98     0   |bctinterf/msg_type.h
 114     9   |bctinterf/net_mem.h
 469   111   |bctinterf/network.h
 233    90   |bctinterf/timer.h
  98     2   |bctinterf/types.h
1485   393   |bctprotoc/R_bcast.h
 245     0   |bctprotoc/config.h
 108    16   |bctprotoc/connexion.h
 213     2   |bctprotoc/error.h
 192    17   |bctprotoc/extern.h
```

123	0	|bctprotoc/global.h
239	39	|bctprotoc/group.h
230	2	|error/error.h
53	0	|error/extern.h
54	0	|error/global.h
129	0	|interf/default.h
60	4	|interf/extern.h
233	90	|sampling/sampling.h
54	17	|simul/buffer.h
20	0	|simul/config.h
63	17	|simul/debug.h
19	1	|simul/debug_sh.tab.h
92	3	|simul/extern.h
77	0	|simul/global.h
22	13	|simul/interf.h
30	6	|simul/mshell.h
30	4	|simul/recording.h
309	84	|simul/reseaux.h
38	5	|simul/simul.h
62	15	|simul/site.h
58	17	|simul/swap.h
130	56	|simul/thread.h
32	4	|simul/types.h
18	4	|simul/verify.h
13	0	|simul/y.tab.h
130	9	|spy/GroupLoad.h
164	50	|spy/GroupLoadP.h
99	2	|spy/GroupSpy.h
125	19	|spy/GroupSpyP.h
134	24	|spy/clientgroup.h
86	2	|spy/extern.h
87	0	|spy/groupstate.h
227	44	|spy/ipcomm.h
111	10	|spy/maingroup.h
168	12?	|unix/endian.h
80	2	|unix/error.h
150	32	|unix/ethernet.h
87	3	|unix/extern.h
63	0	|unix/global.h
48	1	|unix/mach_error.h
227	90	|unix/thread.h
645	204	|bctclient/clnt_client.c
274	57	|bctclient/clnt_defaults.c
936	313	|bctclient/clnt_server.c
86	3	|bctclient/error.c
91	3	|bcteth/error.c
550	162	|bcteth/ethernet.c
39	12	|bcteth/mach_error.c
253	3	|bctinterf/error.c
1145	346	|bctinterf/hostinfo.c

```
 314    64   |bctinterf/init.c
3869  1336   |bctinterf/madata.c
2332   592   |bctinterf/mathread.c
 760   198   |bctinterf/memory.c
1027   293   |bctinterf/message.c
 810   292   |bctinterf/net_mem.c
2670   771?  |bctinterf/network.c
 820   222   |bctinterf/recv_msg.c
 105    35   |bctprotoc/bogomips.c
1634   704   |bctprotoc/boot.c
1218   349   |bctprotoc/broadcast.c
2079   669   |bctprotoc/connexion.c
 323    49   |bctprotoc/daemon.c
 224     3   |bctprotoc/error.c
 718   162   |bctprotoc/errors.c
1516   466   |bctprotoc/groupinfo.c
 847   288   |bctprotoc/grp_create.c
 846   269   |bctprotoc/grp_join.c
 666   217   |bctprotoc/grpcreate.c
 879   319   |bctprotoc/grpjoin.c
 129     8   |bctprotoc/headers.c
1050   306   |bctprotoc/incoming.c
 265    60?  |bctprotoc/init.c
 224     3   |bctprotoc/proterror.c
 195    27   |bctprotoc/protinit.c
 412   128   |bctprotoc/single.c
 498   139   |bctprotoc/storages.c
2057   590   |bctprotoc/synchro.c
  11     6   |bctprotoc/tst.c
 510    70   |error/error.c
 419    51   |error/simerror.c
1623   600   |interf/GLoad.c
 939   228   |interf/Gauge.c
1469   495   |interf/Load.c
 204    55   |interf/interf.c
 118    16   |interf/main.c
 192    63   |sampling/ktimer.c
  33     8   |sampling/sampling.c
 492   114?  |simul/buffer.c
 783   242   |simul/debug.c
1220   418   |simul/debug_sh.tab.c
 283    88   |simul/ethernet.c
 296   101   |simul/interf.c
 710   160   |simul/lex.yy.c
 390   207   |simul/mshell.c
 550   194   |simul/recording.c
1581   517   |simul/reseaux.c
 390   119   |simul/sema.c
  25     6   |simul/simul.c
 418   139   |simul/site.c
```

```
  166    47   |simul/swap.c
 1082   339   |simul/thread.c
   58    13   |simul/timer.c
  617   261   |simul/y.tab.c
  518   188   |spy/GroupLoad.c
  498   101   |spy/GroupSpy.c
  653   138   |spy/client.c
  294    54   |spy/clientcomm.c
  557   153   |spy/clientgroup.c
   23     5   |spy/groupload.c
  322    60   |spy/groupspy.c
  120     5   |spy/main.c
  454   125   |spy/maincomm.c
  321   120   |spy/maingroup.c
  143    14   |spy/spy_init.c
   91     3   |unix/error.c
   64    25   |unix/essai.c
 1042   357   |unix/ethernet.c
  367   102   |unix/sema.c
 1395   423   |unix/thread.c
   16     4   |unix/tst.c
    8     0   |unix/tst2.c
   58     5   |bcteth/Tests/testnset.h
    4     1   |bcteth/Tests/types.h
   61     0   |bctinterf/Tests/global.h
   17     0   |simul/Tests/algo.h
   14     2   |bctclient/Tests/bctclient_init.c
108   23   |bctclient/Tests/bctclient_server_init.c
   80    33 |bctclient/Tests/tst_sock_client.c
   23     3  |bctclient/Tests/tst_sock_server.c
   37    14   |bcteth/Tests/ethernet_recv.c
   61    19  *|bcteth/Tests/ethernet_send.c
   15     6   |bcteth/Tests/getpid.c
  183    70   |bcteth/Tests/myfilter.c
  543   161   |bcteth/Tests/tst_filter.c
  179    68   |bcteth/Tests/tst_mach.c
  182    86   |bcteth/Tests/tst_speed.c
  106    53   |bcteth/Tests/tst_thrd.c
  123    13   |bctinterf/Tests/net_init.c
  363   185   |bctinterf/Tests/tst_buf.c
  108    45   |bctinterf/Tests/tst_critic.c
   77    40   |bctinterf/Tests/tst_elec.c
   57    18   |bctinterf/Tests/tst_mem.c
  153    69   |bctinterf/Tests/tst_msg.c
  237   118   |bctinterf/Tests/tst_site.c
  123    58   |bctinterf/Tests/tst_thrd.c
  167    81   |bctinterf/Tests/tst_time.c
  169   107   |bctinterf/Tests/tst_tout.c
  109    22   |bctprotoc/Tests/bct_init.c
  175    79   |bctprotoc/Tests/tst_broad.c
```

```
   135    59   |bctprotoc/Tests/tst_group.c
   133    62   |bctprotoc/Tests/tst_single.c
   155    69   |bctprotoc/Tests/tst_single1.c
   164    73   |bctprotoc/Tests/tst_single2.c
   165    74   |bctprotoc/Tests/tst_single3.c
   272   122   |bctprotoc/Tests/tst_speed.c
     3     0   |interf/Tests/gmain.c
    50     6   |interf/Tests/main.c
   158    23   |interf/Tests/tst_interf.c
    77    25   |simul/Tests/algo.c
     9     1   |simul/Tests/no_net.c
    35    15   |simul/Tests/tst_buff.c
   105    66   |simul/Tests/tst_jmpb.c
    19     6   |simul/Tests/tst_simul.c
    94    61   |simul/Tests/tst_site.c
    54    23   |simul/Tests/tst_thrd.c
   185    64   |unix/Tests/sendd.c
    15    11   |unix/Tests/set_sticky.c
    39    32   |unix/Tests/tst_jmp.c
    97    39   |unix/Tests/tst_rw.c
   188    60   |unix/Tests/tst_send.c
   103    49   |unix/Tests/tst_thrd.c
    22     7   |unix/Tests/tst_time.c
-----------------
   69963 20036? * total lines/statements

 16793  lines had comments        24.0 %
  2356  lines had history          3.4 %
  2489  comments are inline       -3.6 %
  9546  lines were blank          13.6 %
  7019  lines for preprocessor    10.0 %
 36732  lines containing code     52.5 %
 69957  total lines               100.0 %
362975  comment-chars             17.6 %
 66727  history-chars              3.2 %
 91231  nontext-comment-chars      4.4 %
519971  whitespace-chars          25.3 %
120976  preprocessor-chars         5.9 %
896006  statement-chars           43.5 %
2057886 total characters          100.0 %

98792  tokens, average length 7.45

  0.36  ratio of comment:code
    14  ?:illegal characters found
     2  *:unterminated/nested comments
```

Pour résumer, ces statistiques concernent environ 200 fichiers C, totalisant 70000 lignes, soit 2 mégaoctets et commentés à hauteur de 36 pour cent. Les 36000 lignes de code représentent 20000 instructions selon la méthode du point-virgule. Elles ne prennent pas en

compte les outils de gestion annexes, Makefile, RCS et diverses expérimentations non intégrées.

Annexe B : un exemple de programme de test du protocole

Cette annexe présente le programme de test utilisé pour évaluer le comportement du protocole sous forte charge.

1 Principe

Ce test du protocole consiste à faire émettre depuis chaque site un message. Lors de la réception d'un message en provenance du groupe, si celui−ci a été émis depuis le site local, il est automatiquement retransmis. De cette manière, on s'assure des deux points suivants :

- que la charge du groupe est maximale, car il y a toujours des messages en cours de transmission dans le groupe.
- que tous les sites participent à cette charge car il y a toujours un message de chaque site en transit dans le groupe.

De plus ce programme teste le protcole dans des conditions FIFO, car les messages émis depuis un sites sont toujours reçu dans l'ordre où ils ont été transmis, les débit obtenus correspondent à un comportement séquentiel des applications. Les résultats obtenus sont donc une borne inférieure des débits que l'on est en droit de constater dans les conditions de l'expérience. La figure suivante illuste le principe du protocole dans le cadre où seuls deux sites sont utilisé pour le test :

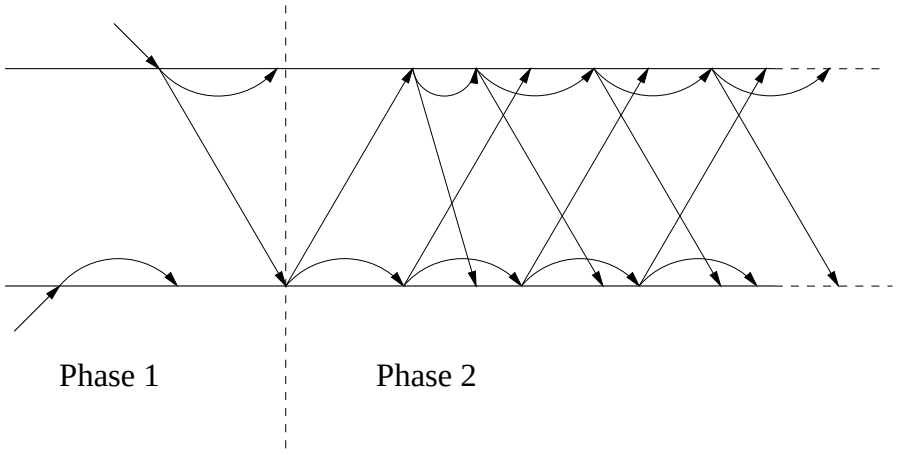


Figure 1 : Le protocole de test

La séparation en deux phase découle du fait que mesurer le débit au sein d'un groupe à un seul participant ne peut guère être interprété que comme une mesure de vitesse du site local. Aussi dans un premier temps les sites se contentent−ils d'émettre à intervalle régulier des

messages. la phase active (variable `full_speed` dans le source) n'est déclanchée que lors de la réception d'un message provenant d'un site différent. A partir de ce moment seulement il y aura réémission des messages locaux reçus.

Un autre point important est le respect de la symétrie entre les différents sites participants au protocole. Il est important de vérifier que tous les sites contribuent dans une part à peut près égale à la charge globale du groupe. Pour ce faire deux compteurs distincts `current_global_stamp` et `current_local_stamp` comptent repectivement le nombre total de paquets reçus et le nombre de paquets en provenance du site local. Les statistiques affichées à intervalles réguliers indiquent les deux débits correspondants.

2 Source

Le source compend deux routines, ProtoMain correpond au programme principal exécuté sur ce site après le démarrage des couches basses du protocole. Cette routine ne se termine que sur arrêt du programme et sert essentiellement à initier la connexion au groupe et les émissions. La deuxième routine IncomingDataReceiver est appelée lors de la réception d'un message en provenance du groupe, et reçoit en paramètre les données transmises. C'est elle qui maintiendra l'activité du protocole en réémettant des messages vers le groupe quand nécessaire.

```
/*
 * COPYRIGHT (C) Bull-IMAG/Systemes 199x[ ,199x]
 *       All Rights Reserved
 */
/*
 * FILE         : src/bct/bctprotoc/tst_speed.c
 * AUTHOR       : Daniel VEILLARD
 * PURPOSE      : module used for testing the Protocol
 */

#ifdef SIMULATION
#include <stdio.h>
#endif
#  include "common/inc/types.h"
#  include "common/err/extern.h"
#  include "bct/bctinterf/extern.h"
#  include "bct/bctprotoc/global.h"

#define STRING_SIZE 10

typedef struct t_message {
 short from;
 short number;
 char diffusion[STRING_SIZE];
 T_Site myself;
} t_message;

T_Bit32 current_global_stamp;/* nb messages recus */
T_Bit32 current_local_stamp; /* nb messages emis */
T_protocol_Grp grp;          /* le groupe groupe */
```

```
t_message *message;          /* le message envoye */
T_machinter_Type type;       /* le type associe */
int full_speed = 0;          /* 1ere ou 2eme phase */

/*
 * routine appelee lors de la reception d'un message
 * dans le groupe
 */
T_Void IncomingDataReceiver(T_protocol_Grp grp,
 T_machinter_Data data, T_Bit32 global_stamp,
 T_Bit32 local_stamp,T_Bool boolean)
{
  /* on recupere les donnees recues */
  t_message *incoming_data = (t_message *) data->data;
  T_Error return_val;

  current_global_stamp++;

  /*
   * on distingue le cas d'un message emis localement.
   */
  if (ntohl(incoming_data->myself) ==
      protocol_MySiteNumber) {
    current_local_stamp++;
    if (!full_speed) return;
    /*
     * si la phase d'echange a ete declanchee renvoyer
     * le message au groupe.
     */
    return_val = machinter_MkData(&data,(T_Void *)
                    message, sizeof(t_message),1,type);             if
  (return_val.code != ERR_OK) {
     err_Print(return_val,"machinter_MkData failed\n");
     exit(1);
    }
    return_val = protocol_BCastSend (grp,data);
    if (return_val.code != ERR_OK) {
     err_Print(return_val,"BCastSend failed ...");
     exit(1);
    }
    return_val = machinter_KillData(data);
  } else {
    if (!full_speed) {
      /*
       * les echanges sont ammorces ici, a partir de ce
       * moment tous les processus connectes vont
       * saturer le groupe Admin.
       */
      return_val = machinter_MkData(&data,(T_Void *)
                    message, sizeof(t_message),1,type);
      if (return_val.code != ERR_OK) {
       err_Print(return_val,"MkData failed\n");
       exit(1);
      }
      return_val = protocol_BCastSend (grp,data);
      if (return_val.code != ERR_OK) {
       err_Print(return_val,"BCastSend failed ...");
       exit(1);
      }
```

```
        return_val = machinter_KillData(data);
        full_speed = 1;
      }
    }
  }

  /*
   * routine principale appelee lorsque les couches
   * inferieures du protocole ont ete initialisees.
   */
  T_Void protocol_ProtoMain(T_Void) {
    T_Error return_val;
    T_protocol_GroupData grpinfo;
    T_machinter_Data data;
    T_Bit32 last_global_stamp;
    T_Bit32 last_local_stamp;
    T_time_t current_time,last_time;
    float delta_t;
    char c = '0';
    int i,j,k;
  #ifdef SIMULATION
    /*
     * dans le cas de simulations, il faut signaler
     * les variables globales non partagees.
     */
    IS_PRIVATE(current_global_stamp);
    IS_PRIVATE(current_local_stamp);
    IS_PRIVATE(grp);
    IS_PRIVATE(message);
    IS_PRIVATE(type);
  #endif

   /*
    * on se connecte ou on cree le groupe Admin.
    */
   while (1) {
    return_val = protocol_ProtoGroupCreate("Admin");
    if (return_val.code == ERR_OK) break;
    return_val = protocol_ProtoGroupJoin("Admin");
    if (return_val.code == ERR_OK) break;
   }

   return_val = protocol_GetGroupInfoByName(&grp,"Admin");
   if (return_val.code != ERR_OK) {
     err_Print(return_val,"protocol_ProtoMain");
     return;
   }
   fprintf(stderr,"\nProtocol running on %X\n",
           protocol_MySiteNumber);

   /*
    * creation et initialisation du message
    * envoye au groupe.
    */
   message = (t_message *)
             machinter_Malloc(sizeof(t_message));
   if (!message) {
     fprintf(stderr,"cannot allocate message\n");
     exit(1);
```

```
}
message->from = htonl(protocol_MySiteNumber);
message->number = 0;
message->myself = htonl(protocol_MySiteNumber);
for (i = 0;i<STRING_SIZE;i++,c++) {
  if (c > '9') c= '0';
  message->diffusion[i] = c;
}

/*
 * creation du type associe au message.
 */
return_val = machinter_MkType(&type);
if (return_val.code != ERR_OK) {
  err_Print(return_val,"machinter_MkType failed\n");
  exit(1);
}

return_val = machinter_AddTypeField(type,MACHINTER_TYPE_BIT16,2);
if (return_val.code != ERR_OK) {
  err_Print(return_val,"machinter_AddTypeField failed\n");
  exit(1);
}

return_val = machinter_AddTypeField(type,MACHINTER_TYPE_CHAR,
                                    STRING_SIZE);
if (return_val.code != ERR_OK) {
  err_Print(return_val,"machinter_AddTypeField failed\n");
  exit(1);
}
type->nb_refs++;

/*
 * diverses initialisation necessaires aux statistiques.
 */
grpinfo = grp->userdata;
current_local_stamp = last_local_stamp = 0;
current_global_stamp = last_global_stamp = grpinfo->global_stamp;
machinter_GetTime(&last_time);

/*
 * on met en place la routine a appeler lors de la reception d'un
 * message.
 */
grpinfo->output_func = IncomingDataReceiver;

/*
 * Creation et envoi d'un premier message
 */
return_val = machinter_MkData(&data,(T_Void *) message,
          sizeof(t_message),1,type);
if (return_val.code != ERR_OK) {
  err_Print(return_val,"machinter_MkData failed\n");
  exit(1);
}

return_val = protocol_BCastSend (grp,data);
if (return_val.code != ERR_OK) {
  err_Print(return_val,"machinter_BCastSend failed ...");
```

```
    exit(1);
  }

  return_val = machinter_KillData(data);

  /*
   * Tant qu'un autre site ne s'est pas manifeste, on envoie un
   * message a intervalle regulier. Une fois les echanges inities,
   * on se contente d'afficher periodiquement des statistiques.
   */
  while (1) {
    /*
     * attente d'un dizieme de seconde.
     */
    machinter_Sleep(1);
    if (!full_speed) {
       /*
        * Creation et envoi d'un message.
        */
       return_val = machinter_MkData(&data,(T_Void *) message,
                    sizeof(t_message),1,type);
       if (return_val.code != ERR_OK) {
         err_Print(return_val,"machinter_MkData failed\n");
         exit(1);
       }
       return_val = protocol_BCastSend (grp,data);
       if (return_val.code != ERR_OK) {
         err_Print(return_val,"machinter_BCastSend failed ...");
         exit(1);
       }
       return_val = machinter_KillData(data);
     }
     /*
      * a intervalle regulier (toute les secondes) des
      * statistiques de debit sont afffichees.
      */
     machinter_GetTime(&current_time);
     if ((current_time - last_time > 10) &&
         (current_global_stamp - last_global_stamp > 0)) {
       delta_t = current_time - last_time;
       delta_t /= 10;
       err_Msg("recv rate %f, send rate %f broadcast/sec\n",
               (current_global_stamp - last_global_stamp) / delta_t,
               (current_local_stamp - last_local_stamp) / delta_t);
       last_global_stamp = current_global_stamp;
       last_local_stamp = current_local_stamp;
       last_time = current_time;
     }
     message->number++;
   }
 }
```

Ce code fonctionne de manière identique sur les différentes plateformes logicielles qui ont servi de cible, en particulier dans le noyau Mach. Seul l'utilisation dans le cadre du simulateur nécessite deux petites adaptations conditionnées par la constante SIMULATION, en effet il

faut dans ce cadre indiquer toutes les variables globales non partagées par les contextes des différentes stations émulées au sein du même processus.

Bibliographie

[1] Lamport, ''Time, clocks, and the ordering of events in a distributed system'', *Conn. of the ACM*, 21(7), pp. 125−133, july 1978.

[2] Chang, Maxemchuk, ''Reliable Broadcast Protocols'', *ACM Trans. Computer Systems*, 2(3), pp. 251−273, August 1984.

[3] Birman, Joseph, ''Reliable Communication in the Presence of Failures'', *ACM Trans. Computer Systems*, 5(1), pp. 47−76, February 1987.

[4] Kenneth P. Birman et T. Joseph, ''Exploiting virtual synchrony in distributed systems'', *ACM Operating System Review*, 21(5), pp. 123−138, Novembre 1997.

[5] K. Birman, A. Schiper et P. Stephenson, ''Lightweigth Causal and Atomic Group Multicast'', *ACM Trans. on Computer System*, Vol 9 (No. 3), pp. 272−314, August 1991.

[6] Kenneth P. Birman, ''The Process Group Approach to Reliable Distributed Computing'', *Communications of the ACM*, 6(12), pp. 37−53, Decembre 1993.

[7] Aletta Riciardi et Kenneth P. Birman, ''Using process group to implement failure detection in asynchronous environment'', *Proceeding of the 11th ACM Symposium on Principles of Distributed Computing*, Montréal Québec, Août 1991.

[8] Kaashoek, Tanenbaum, Hummel et Bal, ''An Efficient Reliable Broadcast Protocol'', *Operating System Review*, 23(4), pp. 5−20, October 1989.

[9] Kaashoek, Tanenbaum, ''Group Communication in the Amoeba Distributed Operating System'', *Proc. 11th ICDCS*, pp. 222−230, May 1991.

[1 0] Andrew S. Tanenbaum, M. Frans Kaashoek et Henri E. Bal, ‘‘Parallel Programming Using Shared Objects and Broadcasting’’, *IEEE Computer*, 25(8), pp. 10−19, Août 1992.

[1 1] Melliar−Smith, Moser, Agrawala, ‘‘Broadcast Protocols for Distributed Systems’’, *IEEE Trans. Parallel and Distributed Systems*, 1(1), pp. 17−25, January 1990.

[1 2] R.Balter, J.−P. Banâtre et S. Krakowiak, *Construction des systèmes d’exploitation répartis*, Collection Didactique de l’INRIA, juillet 1991.

[1 3] Mogul, Rashid, Accetta, ‘‘The Packet Filter : An Efficient Mecanism for User−level Network Code’’, *SIGCOMM 87*, édité par , pp. , , , .

[1 4] S. B. Davidson, H. Garcia−Molina et D. Skeen, ‘‘Consistency in partitioned Network’’, *Computing Survey*, Vol. 17 (No 3), pp. 341−369, september 1985.

[1 5] David K. Gifford, ‘‘Weighted voting for replicated data’’, *Proc. 7th Symp. on Operating Systems Principles*, ACM SIGOPS pp. 150−159, december 1979.

[1 6] Bal, Kaashoek et Tanenbaum, ‘‘Replication Techniques For Speeding Up Parrallel Applications on Distributed Systems’’, *?*,

[1 7] Tanenbaum, Kaashoek et Bal, ‘‘Parallel Programming Using Shared Objets and Broadcasting’’, *IEEE Computer*, 25(8), pp. 10−19, August 1992.

[1 8] Lee et Hudak, ‘‘Memory Coherence in Shared Virtual Memory Systems’’, *ACM trans. on Comp. Syst.*, 7, November 1989.

[1 9] P.Decouchant, P. Le Dot, M. Riveil, C. Roisin et X. Rousset de Pina, *A Synchronization Mecanism for an Objetc Orineted Distributed System*, Bull−IMAG Systèmes, 2 av. de Vignate 38610 Gières, Mars 1991.

[2 0] Mike Accetta, Robert Baron, David Golub, Richard Rashid, Avadis Tevanian et Michael Young, ‘‘Mach : A New Kernel Foundation For UNIX Development’’, *Prooceedings of the Summer 1986 USENIX Conference*, Atlanta 1986.

[2 1] Chandramohan, A. Thekkath, Thu .D Nguyen, Evelyn Moy et Edward D. Lazowska, ‘‘Implementing Network Protocols at User Level’’, *ACM SIGCOMM Proceedings*, 23(4), pp. 64−73, October 1993.

[2 2] N. C. Hutchinson et L. L. Peterson, ‘‘The x−Kernel : An architecture for implementing network protocols’’, *IEEE Transactions on Software Engineering*, 17(1), pp. 64−76, Janvier 1991.

[2 3] S. W. O’Malley et L. L. Peterson, *TCP extension considered harmful*, (RFC 1263), University of Arizona, Octobre 1991.

[2 4] L. L. Peterson, N. Buchholtz et R. D. Schlichting, ‘‘Preserving and using context information in interprocess communication’’, *ACM Transactions on Computer Systems*, 7(3), pp. 217−246, Août 1989.

[2 5] S. Mishra, L. L. Peterson et R. D. Schlichting, ‘‘Consul : a communication substrate for fault−tolerant distributed programs’’, *Distributed Systems Engineering Journal*, 1(2), pp. 1050−1075, Octobre 1993.

[2 6] P. Druschel, L. L. Peterson et B. S. Davie, ‘‘Experience with a high−speed network adaptator : a software perspective’’, *Proceedings of the SIGCOMM’94 Symposium*, Août 1994.

[2 7] C. Bredan S. Traw et Jonathan M. Smith, ‘‘Hardware/Software Organisation of a High Performance ATM Host Interface’’, *IEEE Selected Areas in Communication*, Février 1993.

[2 8] Chandramohan A. Thekkath et Henry M. Levy, ‘‘Limits to Low−Latency Communication on High−Speed Networks’’,

[2 9] Michael K. Reiter, Kenneth P. Birman et Robbert van Renesse, ‘‘A Security Architecture for Fault−Tolerant Systems’’, *ACM Transactions on Computer systems*, 12(4), pp. 340−371, Novembre 1994.

[3 0] Robbert van Renesse et Kenneth P. Birman, *Protocol Composition in Horus*, (TR95−1505), Cornell University, Mars 1995.

[3 1] R. Balter, J. Bernadat, D. Decouchant, A. Duda, A. Freyssinet, S. Krakowiak, M.Meysembourg, P. Le Dot, H. Nguyen Van, E. Paire, M. Riveil, C. Roisin, X. Rousset de Pina, R.Scioville et G. Vandôme, *Design and implementation of Guide an object oriented distributed system*, Bull−IMAG Systèmes, 2 av. de Vignate 38610 Gières, novembre 1990.

[3 2] F.Boyer, ‘‘Coordinating Software Development Tools with Indra’’, *7th Conference on Software Engineering Environments SEE 95*, IEEE Computer Society Press pp. 1−14, Avril 1995.

[3 3] Dan Hildebrand, ‘‘An architectural overview of QNX’’, *Proceeding of the USENIX Workshop on micro−kernel and other kernel architectures*, Avril 1992, Seatle.

[3 4] Adrew Valencia, ‘‘An overview of the VSTa Microkernel’’, *Draft*, .

[3 5] Bomberger, Alan et al., ‘‘The KeyKOS NanoKernel Architecture’’, *Proceedings of the USENIX Workshop on Micro−Kernel and Other Kernel Architectures*, USINIX Association, pp. 95−112, Avril 1995.

Chapitre I

État de l'art

Chapitre II

Conception d'un protocole de diffusion

Chapitre III

Mise en œuvre

Chapitre IV

Méthode de développement

Chapitre V

Résultats des mesures