Architecture pour l'implantation hautes performances des protocoles de communication de niveau transport

Christophe Diot

THESE

présentée par

Christophe DIOT

pour obtenir le titre de **DOCTEUR**

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 23 novembre 1988)

(Spécialité: Informatique)

==========

ARCHITECTURE POUR L'IMPLANTATION HAUTES PERFORMANCES DES PROTOCOLES DE COMMUNICATION DE NIVEAU TRANSPORT

==========

Date de soutenance : Mercredi 30 janvier 1991

Composition du jury :	J.P. VERJUS	Président
	G. JUANOLE	Rapporteur
	P. ROLIN	Rapporteur
	M. DANG	Directeur de thèse
	A. DANTHINE	Examinateur
	G. MICHEL	Examinateur

TABLE DES MATIERES

REMERCIEMENTS

Il y a les gens que l'on doit remercier.
Par chance, il n'en est aucun dans ceux qui suivent et, s'il en existe, on les rajoutera au crayon.

Il y a les gens qu'on a envie de remercier;
il y a ceux que l'on ne remerciera de toute façon jamais assez;
il y a enfin mon frère et mes parents; ce document, et ce qu'il représente leur est dédié.

Ainsi, que tous les gens avec qui j'ai collaboré plus ou moins régulièrement ces trois dernières années se sentent concernés par ces remerciements. Avec des sentiments tous particuliers ...

• Aux membres du jury,
Jean Pierre Verjus; merci d'avoir accepté de présider ce jury.
Guy Juanole et Pierre Rolin; *"Rapporteur: adj. et n. Qui rapporte par indiscrétion ou par malice (?) ce qu'il a vu ou entendu"* [Larousse 90].
André Danthine, présente le triple avantage d'être un très grand spécialiste, européen, et francophone!

• Ceux à qui je dois tout, ou presque; les travailleurs de l'ombre; les architectes de ces travaux:
Gérard Michel; il aurait mérité d'être universitaire; d'hommages.
Michel Dang; ne le répétez pas, c'est un ancien guciste.

• Mon environnement de travail, ceux que j'ai le plus sollicité durant ces travaux, Universitaires ou Industriels, du premier étage du batiment B aux couloirs d'APTOR en passant par le batiment D:

Une mention spéciale pour mon industriel préféré via son P.D.G. Claude Otrage, mais aussi via Eric, Yves, Yang, Pierre, Patrick, Hassen, Claude Hélène, ...
Ainsi que pour mon environnement de recherche, Imad, Jorge, Luc Mac +; sans oublier Serpaggi, la bande à Nicole, ...

• Enfin la vieille garde, les piliers de cette thèse, mes potes, ...
Jacques, Nico, Lina, Brubru, Sabine, Jean-Marc, Marion, Olivier, Rafaele, Blaireau, Youness, Paola c.c., Béru, Tante Adel, Pierre, Martine, ...

ainsi que tous les skieurs, surfeurs, plongeurs, canyonneurs, bikeurs, joyeux campeurs, glandeurs, beurs, travailleurs (???), chercheurs, dragueurs, gros mangeurs, casseurs, râleurs avec qui j'ai vécu l'autre partie (qui a dit la plus importante?) de ces trois années de thèse.

Rien ne sert de soutenir
quelque chose qui ne tient pas debout

Pierre Dac

Chapitre 0: INTRODUCTION

0 INTRODUCTION

De par sa situation dans le modèle OSI, la *couche Transport* a toujours été considérée comme une *couche charnière*. Elle occupe en effet une position intermédiaire entre les couches basses, chargées de la transmission et du routage des informations sur le réseau, et les couches hautes à vocation plutôt applicatives. Elle se retrouve hériter, à ce titre, d'un ensemble de fonctionnalités et services lourds et complexes, qui en rende l'implémentation délicate:

. On sait implanter les couches basses sur circuit de communication (microcontrôleur INTEL 83C152, circuit FDDI par AMD, etc...) et les couches hautes par logiciel; la situation de la couche Transport reste mal définie. Les tentatives d'implantation et de conception d'environnement hôte dédié étudiées (paragraphe1.1.2) le prouvent [Fraser 79][Ansade 88].

. La couche quatre constitue un *goulot d'étranglement dans la pile OSI*. Il est difficile de dire si cela est dû à ses fonctionnalités ou aux implémentations qui en sont faites.

. Le débit des couches basses augmente très vite; les performances de la couche Transport doivent donc suivre cette progression.

. Les besoins en terme d'application évoluent eux aussi (vers le temps réel en particulier). Les protocoles développés il y a 20 ans deviennent mal adaptés aux nouveaux besoins.

. La technologie évolue, les média de communication deviennent plus fiables et les problèmes de sécurité ne se posent plus de la même façon.

Ces constatations nous ont amené à nous intéresser aux problèmes posés par les *critères d'implémentation hautes performances des protocoles de Transport*. Problème qui passe par la remise en cause:

. Du protocole et de ses fonctionnalités.

. De l'environnement d'*implantation* et des techniques d'*implémentation*.

Nous considérerons donc *le protocole, son environnement d'implantation et les techniques d'implémentation utilisées comme un tout indissociable*, en dehors duquel il est très difficile de réunir les conditions qui permettent d'atteindre des performances élevées. L'idée qui consiste à faire évoluer, durant la conception, l'environnement (de développement et d'implantation) vers

le protocole et le protocole vers l'environnement, nous semble primordiale et indispensable à la conception d'un système performant.

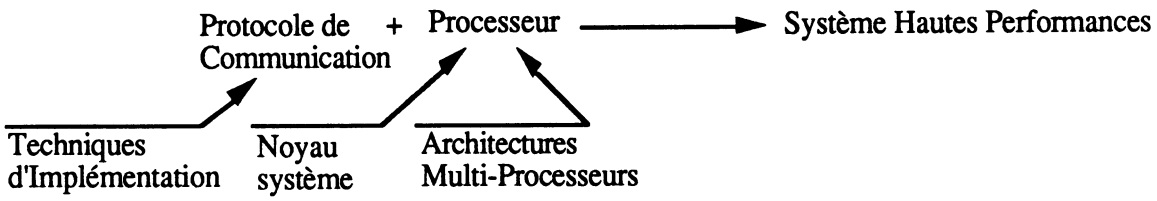
Nous nous proposons d'aborder le problème sous la forme d'une équation chimique. A savoir que les éléments intervenant dans l'équation sont des éléments de départ :

.le protocole de Transport.
.l'environnement (logiciel et matériel).

Il faut leur associer des "catalyseurs" ou éléments extérieurs qui participent à l'association des éléments de base sans modifier ces éléments de base (ces éléments sont donc intacts après la réaction):

.les techniques d'implémentation (structures de données dédiées, algorithmes optimaux).
.un noyau système (dit encore Operating System) spécialement conçu pour être performant dans les conditions de l'implémentation.
.des choix techniques pour l'exploitation optimale de l'environnement (logiciel et matériel)

Le produit obtenu après réaction est un système hautes performances pour l'implémentation d'un protocole de transport. Soit pour synthétiser le tout :



Notons enfin que l'équation doit être réversible afin de pouvoir, par action sur les catalyseurs, étudier et optimiser, selon des critères de hautes performances, les protocoles implémentés.

Pour résoudre ce problème, ou tout au moins y apporter des éléments de réponse, nous avons entrepris d'implémenter un protocole de Transport standard, dans un environnement lui-même standard, en utilisant des moyens adaptés pour obtenir des performances élevées:

. Techniques d'implémentations dédiées.
. Etude d'algorithmes optimaux pour l'implémentation des fonctionnalités coûteuses du protocole.
. Conception d'un noyau élémentaire de primitives systèmes.
. Définition d'environnement multi-processeurs dédié.

Nous avons choisi de travailler dans un environnement standard pour mieux montrer l'importance de la phase d'implémentation et pour prouver que des performances élevées (dans la zone des 100 Méga-bits par seconde, ce qui correspond aux performances actuellement atteintes par les cartes FDDI proposées dans le commerce par des constructeurs tels AMD) sont possibles avec des protocoles classiques, reconnus pour leur complexité (OSI TP4 et TCP pour ne citer que les deux plus populaires).

• *L'environnement d'étude* a été choisi suffisamment *flexible* pour nous permettre d'étudier:

. l'opportunité des services implémentés et, en conséquence, l'analyse des qualités requises par un protocole de Transport à vocation temps réel, ou hautes performances;
. les différents aspects du parallélisme dans le traitement du protocole;
. des architectures multi-Transputers performantes pour implémenter un protocole de niveau trois ou quatre;
. l'efficacité des techniques d'implémentation mises au point
. les problèmes liés à la gestion des ressources manipulées par le protocole

• Des *résultats de cette étude* en environnement mono-processeur nous aboutirons à la définition d'une *plate-forme d'évaluation de protocoles* à base de processeurs standards qui sera utilisée ultérieurement pour :

. effectuer une évaluation plus précise des performances de notre application en environnement parallèle et continuer ainsi l'étude d'opportunité du parallélisme dans l'implantation des protocoles de communication et l'optimisation pour de tels environnements des techniques d'implémentation utilisées;
. l'étude d'architectures hautes performances pour protocoles de niveau Transport;
. analyser les performances de divers protocoles implémentés selon des critères de hautes performances;
. utiliser enfin les résultats sur les études de parallélisme et d'architectures dédiées pour aider à la conception de circuits ASIC spécialisés.

Par conséquent, les travaux relatés dans ce document se décomposent en deux parties principales:

Le *chapitre 1* qui se divise lui même en deux parties. La *section 1.1* présente une étude comparative des protocoles de niveau quatre et de leur fonctionnalité; suivi d'une étude de l'état

de l'art dans le domaine de l'implémentation des protocoles de Transport en environnement matériel dédié (circuits de communications spécialisés ou machines à base de processeurs standard).

Nous utiliserons cette étude préliminaire pour justifier le choix du protocole standard et la définition de l'environnement hôte du protocole sur lequel vont s'appuyer ces travaux (qui, pour des raisons de flexibilité, et de coût, sera un processeur commercial standard, auquel on associera un langage de programmation et des outils de mise au point).

Dans la *section 1.2*, nous commencerons par étudier les caractéristiques du protocole et de l'environnement d'étude afin de mettre au point un ensemble d'outils d'implémentation hautes performances (et de règles d'utilisation de ces outils) adapté aux fonctionnalités caractéristiques d'un protocole de Transport et dédiées au microprocesseur choisi.

Ces outils d'implémentation hautes performances regroupent:

. la définition d'un modèle de structures de données dédié qui optimise le traitement des informations dans le protocole, et les algorithmes de gestion de ressources associés;
. la conception d'un noyau de primitives systèmes élémentaires, adaptées à la fois au processeur et au protocole;
. la proposition de mécanismes efficaces de synchronisation et de communication de processus;
. des algorithmes performants pour l'implémentation des fonctionnalités du protocole;

Ces techniques sont utilisées en fin de chapitre pour produire une implémentation hautes performances d'un protocole de Transport standard.

Dans le *chapitre 2*, nous entreprenons l'évaluation et l'analyse en environnement mono-processeur de l'implémentation réalisée, selon deux méthodes distinctes:

. mesure des performances de bout en bout;
. étude du comportement de ce même protocole en environnement multi-processeurs.

Ces mesures nous permettent de réaliser une étude comportementale du système (considérations sur le processeur hôte et étude de l'opportunité du parallélisme), qui nous mènera, dans la *section 2.2*, à la définition de l'architecture d'une plate-forme parallèle, configurable, pour l'évaluation de protocoles de niveaux Transport et Réseau.

Plate-forme qui nous sera utilisée ultérieurement pour poursuivre ces travaux vers les objectifs que nous nous sommes définis (les limites de l'environnement mono-processeur ne nous permettant plus de progresser).

Dans un dernier *chapitre de synthèse*, nous regroupons un ensemble de réflexions sur l'intérêt de notre démarche, ainsi que sur les limites des protocoles de Transport standard face aux domaines d'applications naissants.

Notons que ce document utilise quelques fois des anglicismes; la signification des termes en anglais étant bien souvent plus précise que la signification de leur équivalent français, quand ce dernier existe. Nous différençierons en particulier les mots *implémentation* et *implantation*. Implémentation désigne les phases de conception d'un logiciel (structure, algorithmes, techniques dédiées) alors qu' implantation caractérise la répartition du logiciel dans son environnement hôte.

Chapitre 1: IMPLEMENTATION D'UN PROTOCOLE DE TRANSPORT STANDARD DANS UN ENVIRONNEMENT HAUTES PERFORMANCES

1 IMPLEMENTATION D'UN PROTOCOLE DE TRANSPORT STANDARD DANS UN ENVIRONNEMENT HAUTES PERFORMANCES

Ce premier chapitre nous permet de choisir un protocole de Transport et un environnement d'implémentation à partir desquels il soit possible d'atteindre les objectifs énoncés en introduction. Nous procédons en deux étapes:

-Etude des protocoles de niveau Transport existants, ainsi que des projets qui traitent de l'implémentation d'un protocole de Transport dans un environnement matériel dédié.

-Analyse des techniques utilisables pour réunir des conditions d'implémentation nécessaires à l'obtention de performances élevées. Adaptations de ces techniques aux contraintes de l'environnement hôte et de l'application.

1.1 Panorama des Protocoles et de leurs Implémentations

1.1.1 Panorama des Protocoles de Communication

1.1.1.1 Panorama des Protocoles de niveau Transport et de leur fonctionnalités

Les candidats étaient nombreux et pour certains d'entre eux, notre choix difficile à justifier. Nous n'avons cependant considéré que les ***protocoles orientés connexions*** (un protocole de Transport peut être avec ou sans connexion [Meister 85]). Un protocole orienté connexions propose un niveau de sécurité supérieur à un protocole sans connexion. Les fonctionnalités de la couche Transport font que les protocoles classiques sont définis en mode connexion. Nous reprendrons, pour cette analyse des fonctionnalités de niveau quatre, la classification utilisée dans [Watson 87], qui sépare les protocoles en deux catégories :

Les protocoles généraux, au rang desquels on peut inscrire ***OSI Transport classe 4*** [ISO 84] [ISO 86], ***TCP*** [Postel 81][Comer 88], ***SNA*** [Martin 87]. Ces protocoles sont plus ou moins considérés comme des standards incontournables et n'ont pas été conçus pour atteindre de hautes performances mais plutôt pour fournir aux utilisateurs une base de communication sûre et commune.

Les préoccupations ayant évolué quant aux domaines d'utilisation de ces réseaux et la sûreté des média s'étant accrue, on a vu naître une seconde génération de protocoles dits ***protocoles spécialisés*** conçus quant à eux pour des applications plus ciblées (implantation VLSI, temps réel).

On peut citer pour les plus connus ***NETBLT*** (Network Block Transfer Protocol)[Clark 88], ***VMTP*** (Versatile Message Transfer Protocol) [Cheriton 89] et ***XTP*** (Xpress Transfer Protocol) [Cohn 88][XTP 90]; avec en marge ***GAM-T 103*** [GAM-T 87] modèle de réseau local temps réel militaire destiné à évoluer en environnement clos.

Ces protocoles, plus ou moins inspirés de leurs ancêtres TCP et TP4 ont pour point commun avec les précédents de reprendre les mêmes fonctionnalités. Les mécanismes incontournables, communs à ces protocoles, sont analysés dans la suite de ce chapitre, et comparés en terme de coût (une fonctionnalité est dite coûteuse si son exécution provoque une diminution importante du flux de données traitées par le protocole).

La gestion de connexion

Une connexion est définie comme une *"association établie par la couche N entre deux ou plusieurs entités pour le transfert de données"* [CCITT 84].

Le mode connexion permet à l'utilisateur de contrôler la cohérence des données transférées, de régler le flux ou de traiter des erreurs.

Ainsi, en mode datagramme (terme consacré pour signifier sans connexion), un contexte unique est retenu pour tout le système. En mode avec connexion, un contexte est conservé par connexion, chaque événement devant être affecté à une connexion avant de pouvoir être traité par le protocole.

Tous les protocoles cités fonctionnent principalement en mode connexion, même s'ils offrent de façon marginale la possibilité d'envoyer des informations en datagramme (TP4, TCP, GAM-T 103, XTP).

Ouverture et Fermeture de connexion

L'efficacité d'un mécanisme d'ouverture et de fermeture de connexion est jugée sur deux critères: la ***rapidité*** et la ***sûreté.***

Deux mécanismes d'ouverture et fermeture normale de connexions sont couramment utilisés.

-La poignée de main nécessite un échange de TPDU entre les entités éloignées. Cette solution est fiable, mais peut devenir lourde (dans le cas de TP4 par exemple qui nécessite 3 échanges à l'ouverture d'une connexion et 2 à la fermeture pour considérer l'opération

-***L'ouverture implicite***, ou encore basée sur temporisation [Watson 81], permet d'ouvrir plus rapidement une connexion mais est peu fiable. Une connexionest considérée ouverte à la réception du premier TPDU et fermée sous contrôle de temporisations.

D'autres temporisations supportent les mécanismes d'ouverture et de fermeture de connexion. Elles assurent la sécurité de l'opération en cas de perte d'un TPDU (ou ne non réception), mais n'interviennent pas en cas d'ouverture ou de fermeture normale.

L'effet direct du mécanisme choisi peut être quantifié par la mesure du ***nombre minimum de TPDU à transmettre pour transférer, en mode connexion, un unique caractère***. Plus ce nombre est faible, et plus le protocole d'ouverture est rapide; mais aussi moins il est fiable.

Le choix du mécanisme d'ouverture / fermeture de connexion est donc une décision complexe, à considérer en fonction de l'usage auquel le protocole est destiné et de la fiabilité du réseau.

Ainsi le nombre minimum de paquets nécessaire au transfert d'un caractère s'échelonne-t-il de 2 (cas du protocole Delta-t [Watson 83] qui ouvre automatiquement les connexions) à 6 (OSI TP4, NETBLT fonctionnent avec le mécanisme de poignée de main) en passant par 3 pour XTP (qui mixe les deux en un mécanisme de poignée de main contrôlé par chien de garde). Notons que TCP permet de combiner ensembles les TPDU d'ouverture et de fermeture de connexions pour accélerer le temps de transfert d'un unique message utilisateur. Trois TPDU sont alors nécessaires pour transmettre un caractère.

Services fournis par le protocole

La couche Transport reste à considérer comme une couche charnière. Son rôle est de fournir un niveau de services qui ne peuvent être assurés ni par la couches inférieure (orientée routage sur le réseau), ni par les couches hautes (orientées application).

Les performances d'un protocole sont directement liées aux services qu'offre ce protocole. Ces services sont en effet négociés entre les protagonistes de la connexion et consomment donc un temps de traitement non négligeable (surtout si le protocole autorise la modification de ces services en cours de fonctionnement d'une connexion).

L'exemple le plus classique est la négociation des paramètres à l'ouverture d'une connexion. Parmi ces paramètres, on peut différencier ceux qui n'ont théoriquement pas besoin d'être mis à jour après ouverture de la connexion. Ils sont réunis, autour d'une appellation propre à l'OSI,

temporisations, du débit maximum et moyen, etc...

Les autres paramètres nécessitent quant à eux une mise à jour régulière. On compte parmi eux les paramètres de contrôle de flux, les numéros de séquence, ...

Si tous les protocoles analysés offrent la possibilité de traiter des paramètres ou de négocier les services, tous ne le font pas de la même manière. TP4 permet d'insérer dans tout type de TPDU en nombre quasi illimité des paramètres (jusqu'à 255), dans la partie dite variable de l'en-tête d'une TPDU, en respectant le format consacré (Figure 1.1.1). Il est donc impossible de prévoir la taille d'une en-tête dans TP4, et par conséquent de traiter rapidement une TPDU.

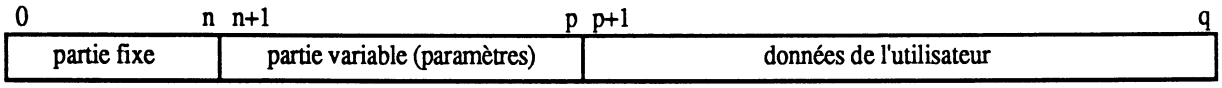


figure 1.1.1.a; TPDU OSI Transport classe 4

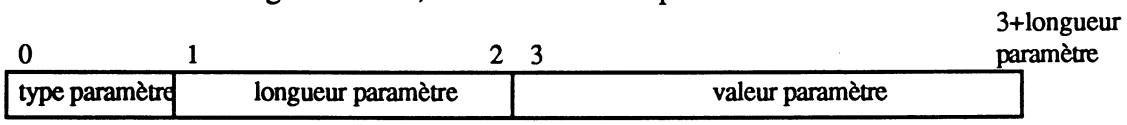


figure 1.1.1.b; format du champs paramètres

La formule retenue par XTP est différente: le format des TPDU est fixe. XTP offre donc moins de liberté quant au type de paramètres mais permet un décodage des informations contenues dans la TPDU beaucoup plus rapide (Figure 1.1.2).

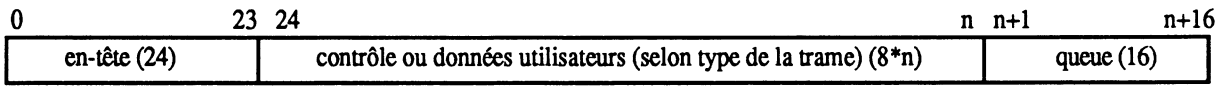


figure 1.1.2; TPDU du protocole XTP

Une solution consiste à convenir préalablement entre les utilisateurs du réseau d'une liste finie de paramètres, dont la valeur serait fixée par défaut de sorte que :

. ne seraient modifiés que les paramètres dont la valeur initiale est différente de la valeur par défaut;
. le nombre de ces paramètres étant limité, la structure du champ paramètre pourrait être fixe et traitable rapidement.

Le Multiplexage

L'une des fonctionnalités classiques d'un protocole de niveau quatre permet de "multiplexer" plusieurs connexions au niveau Transport sur une même connexion au niveau inférieur (généralement une couche Réseau).
Cette procédure est particulièrement intéressante quand on dispose au niveau trois d'un protocole en mode *sans connexion*, mais nécessite lors du transfert Transport vers Réseau, de multiplexer les TPDU vers un même point d'accès. Et réciproquement de démultiplexer, soit d'affecter chaque événement entrant à une connexion Transport avant de pouvoir la traiter (figure 1.1.3).

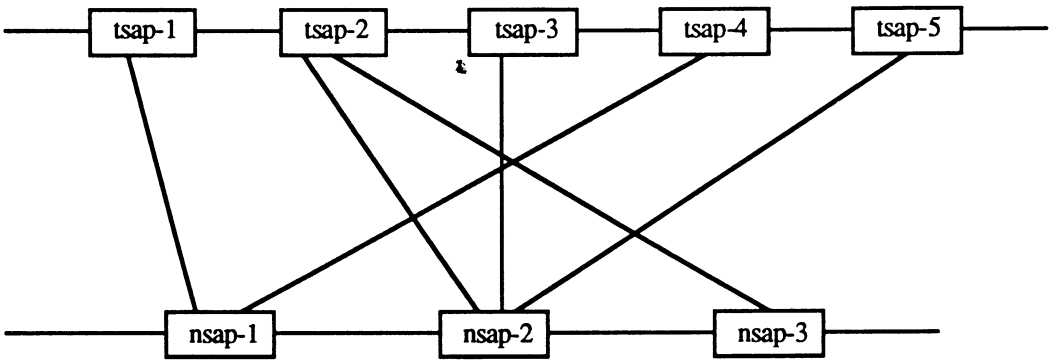


figure 1.1.3; multiplexages de connexions sur les NSAP
Dans cet exemple, trois connexions sont multiplexées sur le NSAP numéro 2.

Tous les protocoles cités offrent cette facilité qui est indispensable dès l'instant où le nombre de connexions susceptibles d'être ouvertes est supérieur au nombre de NSAP.

Stratégie d'acquittement

L'acquittement est utilisé par l'entité réceptrice d'une information pour signaler à l'émetteur la transmission correcte des informations ; les informations de contrôle véhiculées par une AK TPDU permettent

. soit d'effectuer le contrôle de flux
. soit de permettre la réémission des TPDU non reçues.

Les informations de base sont, selon le protocole,

. le numéro des TPDU de données reçues ou le nombre d'octets consécutifs reçus sur

une même connexion (pour le traitement des erreurs).
. Les paramètres de contrôle de flux.

La stratégie d'acquittement est donc un point sensible du protocole puisque, de la fréquence et de la complexité des TPDU d'acquittement dépend la charge de travail des entités connectées. Deux stratégies sont communément utilisées :

i) les acquittements sont générés par le récepteur qui décide, selon une stratégie à déterminer, d'émettre des acquittements vers l'entité émettrice. Cette technique, utilisée par TP4, rend l'émetteur indépendant de la stratégie d'acquittement; avantage qui se trouve balancé par la fréquence nécessairement élevée des acquittements pour produire un système fiable.

ii) les acquittements sont générés par le récepteur sur requête de l'émetteur. Ces requêtes peuvent être implicites (soit liées à des numéros de séquence particuliers) ou explicites (cas de XTP qui bascule un flag dans une TPDU de donnée pour réclamer à l'entité réceptrice un acquittement après réception de la TPDU courante; ou de LAP-B). Cette solution a bien entendu l'avantage de diminuer la fréquence des acquittements et de réduire la charge algorithmique du récepteur qui n'a plus à décider quand il doit émettre des acquittements.

Citons enfin le cas de TCP qui permet, lors d'un transfert bidirectionnel d'informations sur une connexion, d'insérer l'acquittement dans une TPDU de données circulant dans l'autre sens. Cette technique nommée *"piggy-backing"* est d'un intérêt limité puisque utilisable uniquement dans un contexte de transfert bidirectionnel de données sur une même connexion.

Le choix d'une stratégie d'acquittement reste délicat. Le transfert d'un acquittement sur le réseau est encombrant et doit à ce titre être minimisé. Nous comparerons d'un point de vue performances, ces différents mécanismes après évaluation de notre implémentation.

Le contrôle de flux

Associé à la stratégie d'acquittement, le contrôle de flux est connu pour être une fonctionnalité parmi les plus coûteuses pour un protocole de Transport, sans garantir pour autant un très haut niveau de sûreté. Or, vu les débits élevés ciblés par les nouveaux protocoles, le contrôle de flux est plus que jamais nécessaire, même s'il reste à redéfinir dans son fonctionnement pour s'adapter aux contraintes des protocoles temps réels à haute performance. Le contrôle de flux n'est plus seulement obligé de gérer le flux de données traitées par l'entité réceptrice mais surtout le taux de données transmises sur les lignes à chaque instant par l'émetteur [Cheriton 89].

Le rôle du contrôle de flux est donc d'éviter la surcharge du système et la pénurie de ressources chez l'une des entités communicantes. Il doit intervenir, au niveau d'une entité Transport, pour que des congestions ne se produisent pas en réception, durant le traitement des données, ou encore en émission [Gerla 80].

Le contrôle de flux s'effectue entre les deux entités Transport connectées par échange de TPDU de contrôle (c'est le contrôle de flux dit ***de bout en bout***).

En marge du contrôle de flux, certains protocoles implantent un ***contrôle d'accès*** . Ce type de contrôle permet au protocole de Transport de réguler son flux d'informations vers la couche inférieure. L'absence de contrôle d'accès est, en général, compensé par un contrôle de bout en bout au niveau de la couche Réseau (selon un principe similaire à celui employé au niveau supérieur). TCP et VMTP offrent un moyen détourné d'effectuer un contrôle d'accès. Ils utilisent une valeur estimée du temps d'aller-retour d'une TPDU sur le réseau (équivalente au temps d'utilisation d'une ressource par la couche 4); si cette valeur devient trop grande, le protocole entreprend de réduire considérablement le taux de TPDU émises vers la couche réseau [Jacobson 88].

Les méthodes de contrôle de flux de bout en bout sont au nombre de deux :

-Le ***contrôle de flux par fenêtre*** (ou Window Flow Control): à l'ouverture de la connexion, le récepteur spécifie à l'émetteur (et réciproquement) la quantité maximum de données qu'il est prêt à recevoir; cette valeur est nommée fenêtre et est, en général, corrélée directement à la quantité de buffers dont le récepteur dispose (cas d'allocation statique en mémoire) pour la connexion en cours d'ouverture. L'émetteur doit arrêter d'émettre des données lorsque la fenêtre est remplie et attendre une TPDU de contrôle en retour pour mettre à jour les bornes de la fenêtre du récepteur; il peut dès lors reprendre l'émission de données. Le récepteur peut ainsi réduire la taille de la fenêtre quand il le souhaite et selon ses disponibilités en ressources. Cette technique, appelée ***réduction de crédit*** est, en général, très coûteuse de par le nombre d'échanges de TPDU de contrôles qu'elle provoque.

-Le ***contrôle de débit*** (ou rate control): l'émetteur utilise une temporisation pour limiter ses envois. Cette technique impose à l'émetteur de connaître au préalable, la fréquence à laquelle il peut émettre des données, ainsi que la taille des séquences de données. Ces informations lui permettent de calculer le temps qui doit séparer deux flots de données, de sorte que les données sont transmises au récepteur par vagues successives [Clark 87]. Une variante consiste à spécifier directement à l'émetteur le temps minimum à respecter entre deux flots de données.

TP4, TCP et XTP proposent un contrôle de flux par fenêtre ; XTP offre en plus un contrôle de débit. VMTP quant à lui permet le contrôle de débit avec spécification du temps d'intervalle entre les données.
Une analyse simple de ces mécanismes montre que dans certaines situations, le contrôle de flux par fenêtre n'est pas suffisant [Jacobson 88].

La technique adoptée par TCP pour résoudre ce type de problème est connue sous l'appellation "Slow-start algorithm": la fenêtre est initialement choisie de taille réduite et est incrémentée tant que les acquittements arrivent dans un délai spécifié. La fenêtre prend ainsi une taille adaptée au rythme auquel les données peuvent être transmises sur le réseau et auquel le récepteur est capable de les traiter et de les transmettre.

Le contrôle de flux par fenêtre, bien qu'efficace, reste lent puisqu'il oblige le récepteur à émettre des acquittements fréquents, qui encombrent le réseau. De plus, en cas de perte ou ralentissement des acquittements, cette technique devient vite pénalisante pour le débit du protocole.

Le contrôle de débit semble donc meilleur et plus fiable puisqu'il évite la transmission de TPDU de contrôle (si ce n'est pour réajuster la valeur du taux en cas de nécessité). Cependant il est délicat à mettre au point, puisqu'il nécessite la connaissance parfaite du réseau afin d'évaluer précisément la taille et le rythme des flots de données; paramètres qui peuvent éventuellement varier en cours d'activité selon la configuration du réseau et le taux d'occupation de chaque noeud.

Traitement des erreurs

Un protocole de niveau 4 doit être capable, vu sa position charnière entre les couches hautes et basses, de détecter, signaler et, quand cela est possible, corriger des erreurs sur les TPDU traitées. Deux types d'erreurs sont à distinguer:

. les pertes de TPDU;
. les erreurs dans le contenu d'un TPDU

La détection des TPDU perdues, délivrées dans le désordre ou dupliquées est, en général, réalisée grâce aux informations de numéro de séquence. La taille des champs de la TPDU et le checksum permettent quant à eux de détecter les TPDU érronés.

Les numéros de séquence sont basés sur le numéro de paquet (TP4 numérote chaque TPDU de donnée qu'il transmet sur le réseau; il associe à ce numéro la longueur des données utiles transmises) ou sur le nombre d'octets (XTP transmet en en-tête le numéro du premier octet transmis et en queue le numéro du premier octet de la TPDU suivante).

La numérotation par octet permet de localiser plus précisément les octets perdus. Associée, comme dans XTP, à un système de retransmission sélective des octets perdus, elle accroît considérablement l'efficacité du protocole en terme de débit et de gestion mémoire.

Le checksum enfin est sans doute la méthode de détection d'erreur la plus populaire; mais aussi, tel que conçu dans TP4, la plus controversée [Bricker 86], et la plus coûteuse.

Son efficacité est liée à :

. la complexité de l'algorithme de calcul;
. aux champs de la TPDU inclus dans le calcul du checksum;
. sa situation physique dans la TPDU;
. l'implémentation de la procédure (par logiciel ou matériel).

Ainsi dans TP4 (checksum en en-tête, généralement traité par logiciel qui ne couvre pas la partie fixe de l'en-tête), la vérification du checksum est très coûteuse. XTP dont la philosophie est l'intégration VLSI du protocole, propose un checksum beaucoup plus performant (d'autant plus qu'il est situé en queue de TPDU), incluant ou non l'en-tête et la queue.

Ces limites quant à l'efficacité du checksum justifient sans doute qu'il soit optionnel dans la plupart des protocoles cités. De surcroît, le checksum est généralement traité aux niveaux 2 et 3 du modèle OSI (ou équivalent), ce qui n'en justifie pas une troisième évaluation redondante au niveau Transport.

Enfin le problème de la correction des erreurs détectées est traité différemment par les protocoles étudiés. La technique de l'acquittement négatif peut être utilisée pour signifier à l'émetteur les données non reçues ou erronées.

XTP, NETBLT et VMTP préfèrent une technique d'acquittement renvoyant un compte rendu de l'état de la fenêtre de réception (avec les trous dans les données reçues).

TCP et TP4 provoquent la retransmission non sélective pour toutes les informations émises depuis le dernier acquittement. L'émetteur arme donc une temporisation suite à l'émission de

chaque TPDU de donnée; temporisation qu'il désarme à la réception de l'acquittement correspondant; et qui, arrivée à échéance, provoque la réémission de toutes les données émises depuis le dernier acquittement reçu.

XTP et sa technique de retransmission sélective (jusqu'à 16 trous peuvent être signifiés à l'émetteur) ne provoque qu'une retransmission minimale des octets perdus, mais possède une structure de TPDU d'acquittement plus complexe.

La procédure de Segmentation-Réassemblage

Procédure classique de niveau Transport, son rôle est de découper les TSDU transmises par l'utilisateur du service transport en TPDU de taille plus petite de sorte qu'elles puissent être transmises sur le réseau sans subir d'autre découpage.

Ainsi la taille de segmentation est de 1,5 kilo-octets pour un réseau Ethernet et de 4 kilo-octets pour un réseau FDDI. Cette procédure est par expérience coûteuse [Watson 87] et nécessite d'être implémentée avec soin.

Le protocole peut faciliter le traitement de cette procédure en proposant un format de TPDU de données unique, avec en-tête de taille fixe. Mais c'est surtout l'algorithme retenu, lié aux structures de données utilisées (grâce auxquelles ont va minimiser les transferts de données lors du multiplexage ou de la segmentation) qui détermine le coût de cette procédure.

Synthèse

Ce panorama des fonctionnalités classiques (synthétisé table 1.1.3) des protocoles de Transport couvre de manière quasi-exhaustive :

. les protocoles de Transport les plus connus;
. les services et fonctionnalités classiques de cette couche.

D'autres services existent sur les protocoles étudiés mais ils sont trop spécifiques pour être intégrés à une telle étude.

Citons uniquement la diffusion et la concentration qui offrent un moyen d'ouvrir une connexion d'un unique émetteur vers un groupe de récepteurs, et réciproquement d'un groupe d'émetteurs vers un unique récepteur. Cette facilité, laborieuse à implémenter sur TP4 ou TCP (on doit fonctionner sur n connexions si un des deux groupes compte n membres), est proposée par

XTP [Chesson 90]ou GAM-T. Les problèmes posés par la diffusion (et la concentration) fiable sont nombreux (adressage de groupe, fiabilité, récupération d'erreurs).

	TP4	TCP	NETBLT	VMTP	XTP
Gestion de connexions	avec	avec	avec	avec	avec
Ouvertutre de connexions	poignée de main	poignée de main	poignée de main	implicite	implicite
Fermeture de connexions	poignée de main	poignée de main	poignée de main	implicite	poignée de main
Echange minimum de données	6	3	6	2	3
Stratégie d'acquittement	basée sur la réception de donnée				sur demande de l'émetteur
Contrôle de flux	fenêtre	fenêtre	fenêtre et taux	taux	fenêtre et taux
Checksum	en tête	en tête	en queue	en queue	en queue
Numéro de séquence	num. des PDU	num. des octets	num. des PDU	num. des PDU	num. des octets

table 1.1.3; synthèse des fonctionnalités des protocoles de Transport étudiés

Une étude plus exhaustive des protocoles et de leur fonctionnalité ainsi qu'une comparaison de ces protocoles pour déterminer leur aptitude à atteindre des hautes performances est disponible dans [Doeringer 90].

Enfin cette étude bibliographique des protocoles de niveau Transport et de leurs fonctionnalités ne serait pas complète sans un rappel des tâches connues pour être coûteuses dans l'implémentation d'un tel protocole. La littérature spécialisée est unanimement d'accord sur le fait que [Svodobova 88] [Watson 87]:

i) le protocole doit éviter de manipuler les informations utiles des TPDU et TSDU qu'il traite.

ii) allocation et restitution mémoire consomment une proportion du temps de traitement d'un événement plus importante que les activités propres au protocole.

iii) un protocole de ce niveau s'appuie nécessairement, pour fonctionner correctement, sur une certaine quantité de temporisateurs; les fonctions les plus fréquemment utilisées étant l'armement et l'arrêt avant échéance de ces temporisations. Citons TP4 qui recommande l'ouverture minimum de 4 temporisateurs pour chaque connexion, l'un d'eux étant armé à

l'émission de chaque DT TPDU et stoppé sur chaque acquittement. Une mauvaise gestion de temporisations peut devenir catastrophique pour les performances de l'application.

iv) certaines fonctionnalités du protocole sont plus sensibles à l'implémentation que d'autres. Ainsi Segmentation-Réassemblage, procédure qui permet à un protocole de découper une TSDU utilisateur en plusieurs DT TPDU avant de la remettre au niveau inférieur est très sensible aux problèmes évoqués dans les points précédents. Son utilisation peut réduire les performances de 10 % sur XTP; et jusqu'à 30 % sur TP4. Il est donc particulièrement intéressant de l'implémenter correctement et surtout de paramétrer le protocole pour en minimiser l'influence.

L'analyse précise des algorithmes et autres techniques utilisables pour optimiser l'implémentation sera réalisée dans la phase d'implémentation du protocole choisi. Il est cependant utile d'avoir à l'esprit ces quelques points névralgiques des protocoles de communication pour mieux comprendre les choix effectués ultérieurement.

Il nous reste, consécutivement à cet état de l'art des protocoles de niveau Transport, à justifier le choix du protocole de Transport OSI classe 4 pour la suite de nos travaux

1.1.1.2 OSI Transport Classe 4

1.1.1.2.1 TP4 versus TCP

Une sélection préliminaire devait nous amener à choisir parmi le groupe des protocoles généraux au rang desquels on compte TCP et TP4, plutôt que des protocoles dédiés à un usage spécifique. Les raisons de ce choix sont:

. Ils sont considérés tous deux comme standard (TP4) ou standard de fait (TCP). Leurs spécifications sont donc stables.

. Bon nombre d'études relatent d'expériences d'implémentation de TP4 et TCP, de mesures de performances, ou encore d'études comparatives.

Les autres protocoles (VMTP, NETBLT, GAM-T, XTP) sont beaucoup plus récents et en pleine évolution, donc beaucoup plus difficiles à manier pour appuyer une démonstration de ce type. On connaît, en effet, les performances de TCP et TP4 dans des environnements classiques; les autres protocoles sont encore en cours d'évaluation.

. La nouvelle génération de protocole a été conçue sur inspiration de TCP et TP4. Ils reprennent donc, en les améliorant ou en en transformant les fonctionnalités et services, TP4 et TCP.

Seul XTP aurait pu concurrencer TCP et TP4 parce que défini comme un protocole de Transfert temps réel, proche de nos préoccupations, qui intègre un certain nombre de nouveautés. Malheureusement, ce protocole n'avait pas atteint, au début de nos travaux, un niveau de spécification suffisant (en particulier, les Machines d'Etats Finis n'étaient pas encore spécifiées formellement) et, bien que très attrayants, nous avons dû y préférer OSI TP4.

Le choix entre TP4 et TCP était plus délicat. Nous avons sélectionné le protocole OSI Transport classe 4 parce que :

- Plus récent que TCP, TP4 a reçu l'approbation de la communauté scientifique internationale par sa normalisation. Ses spécifications sont donc définies semi-formellement de manière unique par les documents officiels des organismes de normalisation (ISO, CCITT pour les plus importants).

- Du point de vue des fonctionnalités, TP4 et TCP sont très voisins; l'argument technique n'était donc pas décisif dans le choix, même si [Doeringer 90] semble montrer que TP4 est plus prédisposé au domaine des applications hautes performances que TCP.

- Afin d'accroître les performances (connues pour être peu élevées) de TP4 (dues à sa complexité et à sa dépendance du noyau système de l'environnement hôte), des groupes d'industriels se sont formés pour définir sur TP4 des ***profils***. Citons, pour les trois connus ***MAP*** (General Motors [Kaminsky 86]), ***TOP*** (projet similaire chez BOEING) et pour l'Europe, ***CNMA*** (profil défini dans le cadre d'un projet Esprit [CNMA 87]). Ces profils se contentent d'émettre un certain nombre de ***recommandations*** qui sont censées réduire la complexité du protocole et en accroître les performances. Ils restent, cependant, dans les limites fixées par le standard OSI puisque les recommandations n'affectent que des options du protocole. Ils "recommandent" en effet d'accepter tous les services et fonctionnalités définis par l'ISO, mais de n'en utiliser qu'une partie. Et chacun de ces profils est capable de communiquer avec toute implémentation d'un protocole de Transport certifié conforme à la norme OSI Transport classe 4.

Il est de surcroît intéressant de voir que, bien que la définition de MAP et TOP ait été réalisée parallèlement, les recommandations sont sensiblement les mêmes puisque guidées par les mêmes intérêts. Pour des raisons de proximité entre les recommandations de ces trois profils, et

parce que défini par un consortium européen, nous avons choisi de suivre le profil CNMA. Suivre les recommandations d'un profil nous permettait de nous rapprocher, d'un point de vue complexité et performances, des protocoles de nouvelle génération (bien que TP4 selon le profil CNMA reste un protocole général adapté, par l'intermédiaire du profil, aux besoins spécifiques des applications industrielles), tout en conservant l'étiquette OSI.

1.1.1.2.2 Présentation du Protocole de Transport OSI, classe 4

Généralités

Le protocole de Transport OSI classe 4 est défini par les normes ISO 8072 (Service Transport) et ISO 8073 (Protocole de Transport).

Les recommandations propres au profil CNMA seront énumérées et analysées à la fin de ce paragraphe.

La couche Transport apparaît comme une couche clé dans le modèle OSI (figure 1.1.4).

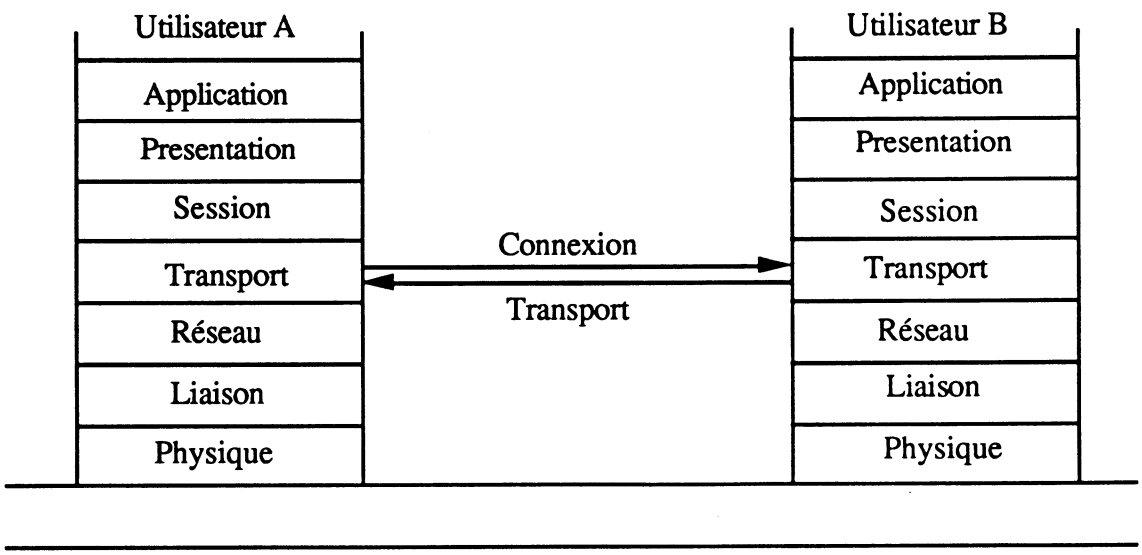


figure 1.1.4; modèle OSI en 7 couches des protocoles de communication

Son rôle est, en effet, de permettre un ***transfert de donnée de bout en bout*** du réseau qui soit ***transparent à l'utilisateur*** du service transport.

Les couches supérieures sont orientées vers l'application, alors que les couches 1 à 3 travaillent au routage des informations et nécessitent la connaissance exacte du réseau. La couche

Transport, divisée en deux parties (le Service et le Protocole de Transport) gère l'établissement de connexions et la communication des deux entités utilisatrices sur la dite ***connexion en mode point à point***, quelque soit le ou les réseaux à parcourir (une connexion est à considérer comme un ***chemin virtuel*** entre deux utilisateurs distants d'un service Transport).

Le service Transport transmet dans l'ordre et sans erreurs les données qui lui ont été remises.

Pour chaque connexion est négociée une Qualité de Service ou QoS (Quality of Service).

Cette Qualité de Service permet à l'utilisateur du Service Transport de fixer les conditions d'établissement de connexion en terme de délais, débits, taux d'erreurs et probabilité. Cependant, l'utilisateur distant, pas plus que les deux Services Transport concernés, ne sont tenus de respecter ces paramètres. Ceci justifie que les paramètres soient négociés à l'ouverture entre les utilisateurs.

Ainsi le QoS proposé par l'utilisateur du Service Transport qui réclame l'ouverture d'une connexion ne peut qu'être affaibli par l'utilisateur distant et les deux Services Transport impliqués; quitte à ce que l'utilisateur de départ refuse les propositions de ses partenaires en retour.

L'utilisateur du Service Transport accède à ce dernier par un ***point d'accès au service Transport*** appelé encore TSAP (ou "Transport Service Access Point") ; de même que le protocole de Transport accède au service réseau par un NSAP (Network Service Access Point). TSAP et NSAP sont identifiés par leurs adresses, uniques sur le réseau. Une adresse de NSAP est construite par rajout d'un champ de n octets sur l'adresse du LSAP, selon la norme ISO 8348 ADD 2 [ISO 87]. Alors que deux octets suffisent pour coder un adresse de TSAP. Mais la connaissance des adresses de TSAPs ne suffit pas pour identifier une connexion Transport et l'on a recours à une ***référence de connexion*** codée sur 2 octets. Cette référence est définie localement. Elle permet d'identifier une connexion sans faire référence aux adresses de TSAP, et permet d'identifier deux connexions distinctes établies entre un même couple de TSAP. La connaissance de cette référence permet d'accéder au ***contexte de la connexion*** qui contient:

. les adresses de TSAP et NSAP locaux (l'adresse de TSAP est unique);
. l'adresse du TSAP distant;
. la référence distante de la connexion.

entre deux TSAPs de manière transparente à l'utilisateur du service Transport.

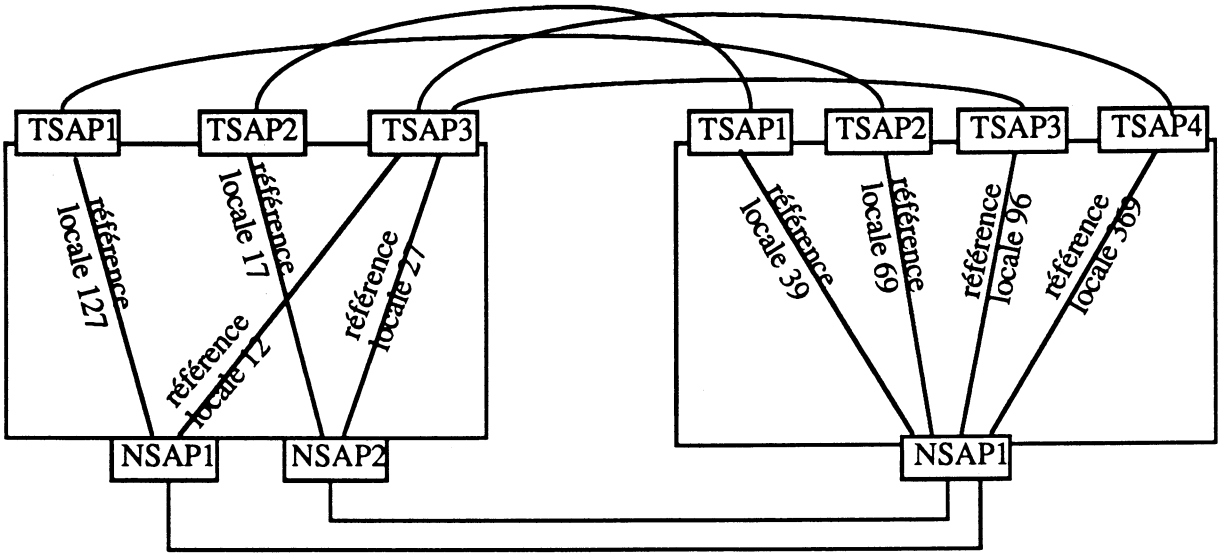


Figure 1.1.5; identification des connexions Transport par leurs références locales

Prenons par exemple la connexion établie entre le TSAP2 et le TSAP1 de l'entité distante; Elle est identifiée dans chaque entité par la connaissance de sa référence locale (respectivement 17 et 39) à partir de laquelle elle peut reconstituer une partie de l'itinéraire parcouru. Chaque utilisateur ne connaît quant à lui que l'adresse des TSAP locaux et distant ainsi que la référence locale. La référence est donc l'élément clé de la connexion puisqu'elle permet l'accès au contexte de connexion qui lui même contient toutes les informations vitales à la connexion (pour la connexion référencée 17):

. adresses des points d'accès locaux (TSAP2 et NSAP2);
. identification de l'entité distante (référence distante 39, TSAP distant 1);
. QoS négocié;
. variables d'état de la connexion.

Le contexte est en quelque sorte la mémoire de la connexion. Il doit être consulté puis mis à jour pour chaque événement traité.

Enfin, la référence permet de contrôler l'état d'une connexion selon les trois phases de la vie d'une connexion: active, gelée ou fermée. Le gel de la référence pendant un intervalle de temps contrôlé par un temporisateur (après fermeture de la connexion) garantit que si une TPDU égarée arrive sur la connexion fermée, elle ne vienne pas perturber une nouvelle connexion qui aurait été établie avec la même référence locale.

Revenons à la couche Transport d'un point de vue général. Nous l'avons dit, cette couche a une situation intermédiaire dans la pile OSI et, à ce titre, doit remplir deux fonctions :

. adapter le format des TSDU reçus de l'utilisateur aux caractéristiques du réseau et délivrer ces TSDU sans erreur et dans l'ordre initial

. combler les carences des couches inférieures, soit fournir les services et fonctionnalités qui ne sont pas assurés par les couches inférieures afin de garantir à l'utilisateur un niveau de sécurité élevé mais aussi de faciliter le traitement des couches une à trois.

Dans cet esprit, TP4 est capable de multiplexer plusieurs connexions sur un même NSAP (si la couche réseau fonctionne en mode sans connexion par exemple) ou pour d'autres raisons d'éclater une connexion sur plusieurs NSAP. Ces fonctionnalités nécessitent, *au niveau Transport, de* ***grosses capacités de stockage*** *de données ainsi qu'un système de* ***gestion mémoire performant***, qui risque fort d'être complexe.

Enfin, et afin de s'adapter aux besoins et qualités du réseau sous-jacent, le protocole propose cinq classes assurant chacune un niveau de sûreté différent.

Classe 0 dite classe de base

La classe 0 n'offre pas de récupération d'erreur, de multiplexage, ni de contrôle de flux. Le service est minimum. La classe 0 est encore définie par la norme T70 du CCITT pour les terminaux Télétex. Cette classe est prévue pour être utilisée sur une couche trois fiable.

Classe 1 dite classe de base avec reprise sur erreur

Elle offre un niveau de service plus élevé avec des procédures élémentaires de détection et de reprise des erreurs signalées par la couche inférieure.

Classe 2 dite classe avec multiplexage

Cette classe introduit un contrôle de flux et le multiplexage. Par contre, la classe 2 n'offre aucun moyen de détecter et de corriger des erreurs.

Classe 3 dite classe avec multiplexage et reprise sur erreur

La classe 3 offre les fonctions de la classe 2 avec utilisation du contrôle de flux explicite, plus la possibilité de reprise après un incident signalé par la couche Réseau.

Classe 4 dite classe avec détection d'erreur et reprise sur erreur

On y retrouve toutes les fonctionnalités de la classe 3, auxquelles on ajoute la détection de TPDU perdues, dupliquées ou hors séquence, et la détection d'erreurs non signalées par la couche inférieure. C'est la classe la plus performante qui permet de se passer, dans un réseau local, de couche 3. Et, dans tous les cas, de s'adapter sur une couche trois de faible puissance.

Nous allons, dans un deuxième temps, étudier les interactions existantes entre la couche transport et son environnement dans un réseau (schématisées figure 1.1.6) :

Le Service Transport

Le Service Transport effectue l'interaction avec la couche supérieure (norme ISO 8072). Par Service Transport, on entend l'ensemble des services que la couche quatre fournit à l'utilisateur de cette couche. Le dialogue avec l'utilisateur du Service Transport (ou TS User) est assuré par les ***primitives de service.***

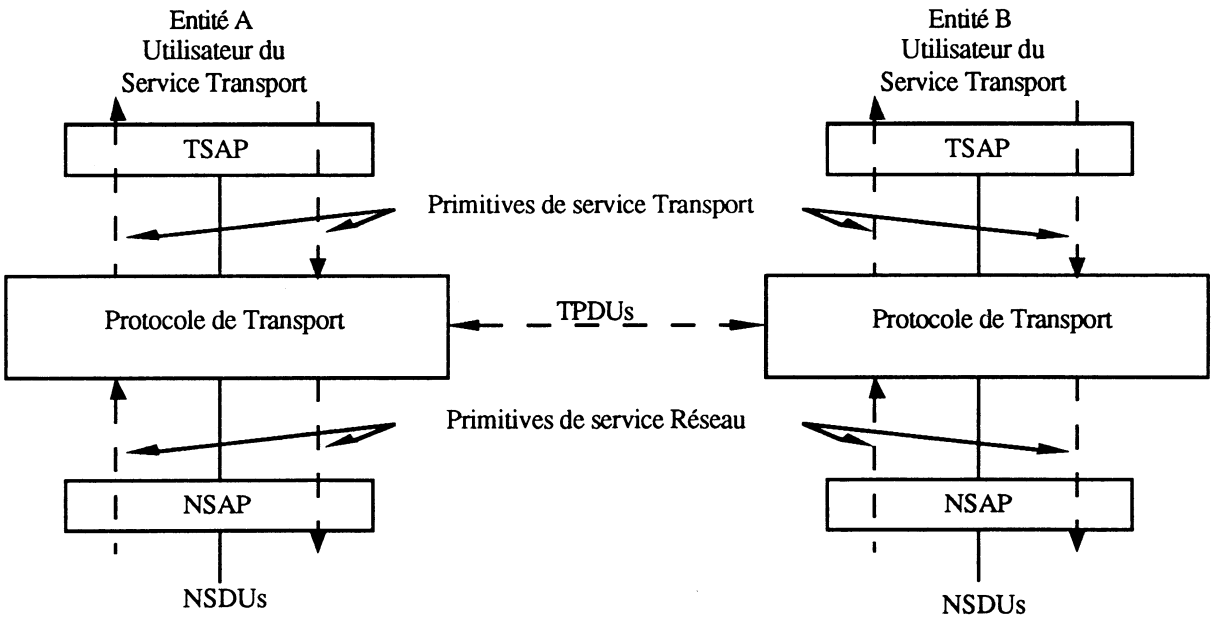


Figure 1.1.6; Communication entre le protocole de Transport et les couches adjacentes

Ces primitives peuvent être de quatre types différents selon le type de la requête et son régime (Figure 1.1.7).

Requête	commande effectuée par le TS User
Indication service	requête effectuée par le protocole de Transport vers l'utilisateur de son
Réponse	réponse à une requête du protocole de Transport; réponse à une indication
Confirmation	réponse à une requête de l'utilisateur; réponse à une requête

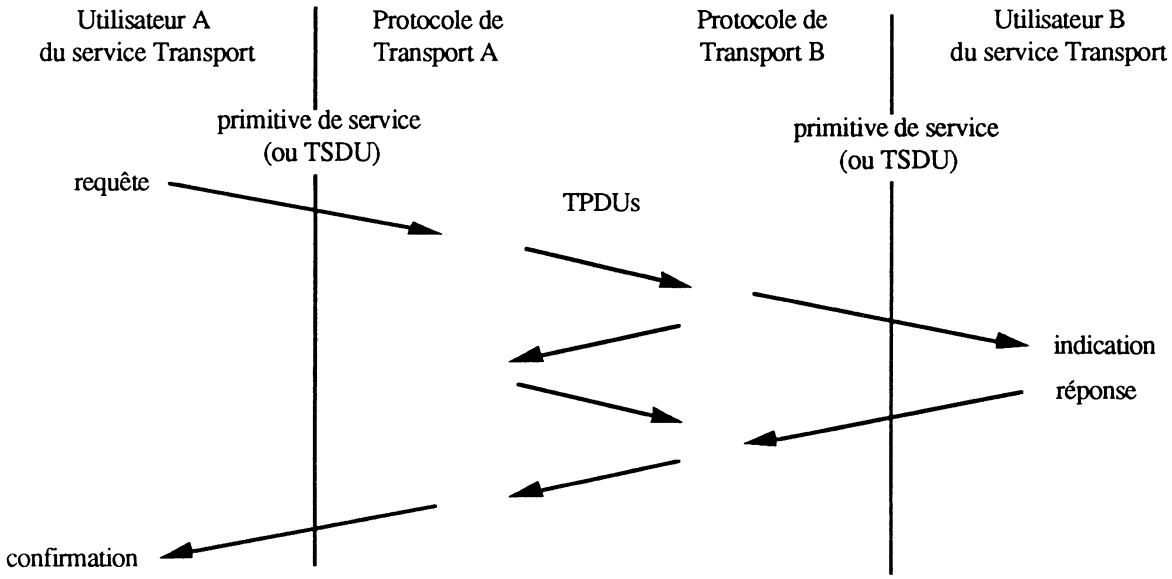


Figure 1.1.7; fonctionnement des primitives service

Ces primitives peuvent enfin être utilisées dans les situations suivantes :

. ouverture de connexion;
. fermeture de connexion;
. transfert de données (normales et express).

Les primitives de service ont une existence locale puisqu'elles sont échangées entre deux couches adjacentes d'une même entité. Leur format est donc totalement indépendant du format des TPDU échangées sur le réseau et elles peuvent revêtir, selon les contraintes de l'environnement et l'implémentation, différentes formes :

. descripteur construit en mémoire partagée pointant sur les données à transférer;
. file d'attente de style FIFO entre les deux couches;
. messages transmis par bus ou lien série, construits par rajout d'une entête de service (par exemple) sur la PDU de niveau inférieur ou supérieur.

Le Protocole de Transport

Le protocole de Transport définit la procédure de transfert point à point de données entre deux utilisateurs de Service Transport.
Cette procédure veut définir *«un protocole de Transport qui soit simple, mais suffisamment général pour convenir à toute la gamme des qualités de services de réseaux possibles sans restreindre les extensions futures »* (citation issue de la norme ISO 8073). Le protocole utilise pour ce faire des unités de données appelées ***TPDU*** pour ***Transport Protocol Data Unit***.

Une TPDU est échangée entre deux entités distinctes de transport. Elles sont donc "véhiculées", "routées" sur le réseau par les couches inférieures pour atteindre selon un chemin que ne maîtrise ni ne connaît la couche Transport d'origine, l'entité destinataire.

Si les TPDU peuvent ne pas être délivrées dans l'ordre d'émission à l'entité transport distante, les TSDU doivent par contre être remises dans l'ordre par le service distant à l'utilisateur. Les TPDU sont, pour le protocole de Transport OSI classe 4, au nombre de 7 et se répartissent les diverses phases de transfert de la façon suivante :
Ouverture de connexion:

CR	TPDU	demande d'ouverture de connexion
CC	TPDU	confirmation d'ouverture de connexion

Transfert de données:

DT	TPDU	TPDU contenant des données utiles.
ED	TPDU	TPDU contenant des données express (données de priorité et contrôle de
flux		différents des précédentes)
AK	TPDU	TPDU d'acquittement des DT TPDU
EA	TPDU	TPDU d'acquittement des ED TPDU

Erreurs et Déconnexion:

ER	TPDU	report vers l'entité émettrice d'une erreur détectée
DR	TPDU	demande de déconnexion
DC	TPDU	confirmation de déconnexion

Les échanges de TPDU entre deux entités Transport sont contrôlés par le protocole de Transport .

Pour ne pas surcharger cette étude, le lecteur aura loisir de consulter, s'il ne les connaît pas déjà, le protocole d'échange des TPDU dans la norme ISO 8073 publiée par l'ISO, l'AFNOR ou encore le CCITT sous l'appellation *recommandation X225* ; ainsi que dans l'ouvrage de Pierre Rolin, intitulé *"Réseaux Locaux ; Normes et Protocoles"* aux éditions Hermès [Rolin 90].

La nature locale des primitives de service ne nécessite pas qu'un format soit préalablement convenu entre les entités distantes. Le format des TPDU doit quant à lui être standardisé afin d'être accepté par toutes les entités transport du réseau. La structure d'une TPDU selon la norme ISO est pour le moins classique et se divise en deux champs (voir figure 1.1.8):

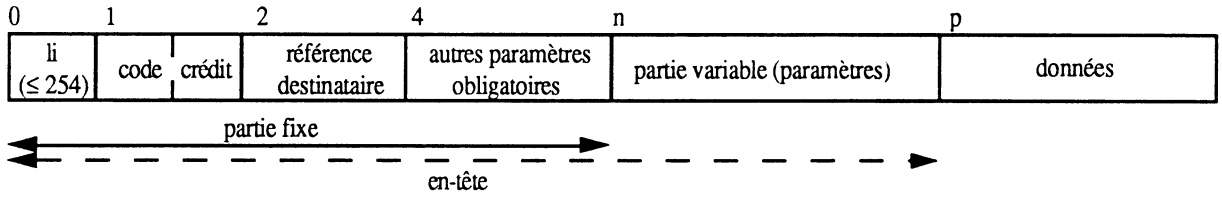


figure 1.1.8; structure d'une TPDU définie dans le modèle OSI

- l'en-tête qui se divise elle-même en deux parties

  .la partie fixe qui contient les informations nécessaires à l'identification d'une TPDU (code, longueur, adresses).

  .la partie variable contenant les paramètres. Cette zone peut être vide (on calcule son existence à partir du champ taille de la partie fixe). Les paramètres y sont exprimés en trois champs: le code, suivi de la longueur du paramètre et enfin la valeur du paramètre.

- les données. Cette zone est optionnelle et de longueur limitée dans certaines TPDU. Ou de longueur négociée à l'ouverture de la connexion (pour les DT TPDU). Nous renvoyons le

lecteur aux ouvrages cités pour obtenir le format exact de chaque type de TPDU et leur fonctionnement.

Avant de passer à l'étude des fonctionnalités offertes par le protocole de Transport OSI classe 4, nous aimerions clarifier la différence existant entre données express et données normales. Les TPDU de données express transfèrent sur le réseau des TSDU express d'une longueur inférieure ou égale à 16 octets avec une priorité supérieure à celle des DT TPDU. Ainsi une ED TPDU de numéro plus petit ou égal à une DT TPDU est prioritaire sur celle-ci, même si arrivée ultérieurement à la couche 4. De plus, chaque ED TPDU doit être acquitté individuellement par une EA TPDU; le transfert de nouvelles DT TPDU ne pouvant reprendre que lorsque toutes les ED TPDU émises ont été acquittées. Ainsi le système de numérotation (et le contrôle de flux) est-il indépendant pour les deux familles.

Les fonctionnalités de la couche Transport OSI classe 4

Outre les fonctions de base (définies par la norme comme «Fonctions utilisées en toutes circonstances, selon la classe»):

. transfert de TPDU;
. multiplexage et démultiplexage;
. détection d'erreurs;
. reprise sur erreur.

Le protocole de Transport OSI classe 4 offre les fonctions suivantes :

concaténation et séparation permet de réunir plusieurs TPDU dans un même NSDU au niveau de l'entité Transport expéditrice. Cette procédure n'est pas indispensable quant la taille des TPDU a été choisie en accord avec les contraintes de la couche réseau.

segmentation et réassemblage; (décrite précédemment) cette procédure, qui permet de découper une TSDU entrante en plusieurs TPDU (puis de les regrouper) est normalement indispensable car elle permet d'adapter la taille des TSDU reçus de l'utilisateur aux contraintes du réseau.

éclatement et recombinaison permet de transférer les TPDU d'une même connexion sur plusieurs connexions réseau. Cette procédure est généralement inutilisée, dans la mesure ou TP4 offrant une qualité de service suffisante, la couche réseau qui lui est associée fonctionne en mode sans connexion.

Contrôle de Flux. Cette procédure, ainsi que les techniques qui lui sont associées a été amplement développée dans le panorama du protocole de ce début de chapitre. Rappelons que dans TP4, la méthode utilisée est la fenêtre. Les limites de cette fenêtre sont mises à jour par acquittement (ou AK TPDU). C'est enfin le récepteur qui décide seul de la fréquence des acquittements.

Temporisateurs utilisés par TP4

Nous avons déjà évoqué le problème des temporisations dans un tel protocole. OSI TP4 est particulièrement dépendant de la stratégie de gestion des temporisations ; il ne requiert pas moins de 5 types de temporisations différents par connexion.

T1 est le délai de réexpédition; soit le temps limite après lequel une entité réémet une TPDU si cette dernière n'a toujours pas été acquittée.
Cette temporisation est donc utilisée à l'émission de chaque TPDU. Afin d'en optimiser le rendement, deux stratégies, jugées équivalentes par [Svodobova 88] s'affrontent.

i) une temporisation unique est associée à chaque DT TPDU; à la réception d'un acquittement chaque temporisation correspondant à une TPDU acquittée doit être arrêtée.

ii) une temporisation unique est utilisée pour la connexion ; elle est donc réinitialisée à chaque émission de DT TPDU; la réception d'un acquittement provoque soit l'arrêt de la temporisation si toutes les données émises ont été acquittées, soit sa réinitialisation.

R est le délai de persistence ; ou temps maximum pendant lequel une connexion s'autorise à réémettre un ensemble de TPDU. R est évalué par rapport à T1 de la façon suivante :

$$R > T1 * N \text{ où N est le nombre maximum de réexpéditions.}$$

L délai minimum de réutilisation d'un numéro de séquence ou d'une référence : l'appellation étant suffisamment explicite, justifions l'existence de cette temporisation.

Afin qu'un numéro de séquence ne soit pas réutilisé trop rapidement et risque de perturber le contrôle de flux, il est gelé pendant un certain temps. La justification au niveau des références de connexions est sensiblement la même.
Cependant on peut se passer de L pour les numéros de séquence en utilisant un "compteur d'époque" qui, s'il est de taille suffisante joue le même rôle que L.

I délai d'inactivité: si une entité Transport ne reçoit aucune TPDU pendant un temps I, elle est autorisée à lancer la procédure de libération de connexion.

W délai de réexpédition d'informations de contrôle de fenêtre ou encore si après le délai W, l'entité réceptrice n'a pas envoyé d'information de mise à jour de sa fenêtre et de son crédit à l'entité émettrice, elle transmet alors un acquittement de contrôle de flux (même numéro que le précédent différencié par la sous numérotation).

La mise en marche, l'arrêt ou l'échéance de ces temporisations est considéré comme un événement sur une connexion, au même titre que la réception d'une TPDU ou d'une primitive de service. L'utilisation des temporisateurs est donc définie formellement dans les tables d'état du protocole, en annexe de la norme ISO 8073.

1.1.1.2.3 Le profil CNMA

Cette présentation du profil CNMA se base sur le document de référence, en l'occurrence le ***CNMA Implementation Guide 3.1***, p. 5.1 à 5.3. [CNMA 87].

CNMA est défini pour utiliser une couche réseau sans connexion et propose pour la couche Transport de suivre le protocole de Transport OSI classe 4 pour lequel il recommande une série d'options, et des valeurs de paramètres. Nous citerons pour les recommendations les plus importantes :

. ne pas utiliser le checksum, excepté pour les CR TPDU;
. ne pas utiliser les ER TPDU pour signaler une erreur (détectée selon la procédure 6.22 de la norme ISO 8073) ; utiliser pour ce faire une DR TPDU;
. ne pas utiliser la procédure de concaténation (mais bien sûr accepter de séparer);
. éviter de demander des réductions de crédit ; cette procédure est jugée trop coûteuse;
. ne pas envoyer de paramètres de contrôle de flux dans les AK TPDU;
. fermer la connexion si des paquets inconnus sont reçus.

Enfin CNMA propose (table 1.1.9) des valeurs pour les temporisateurs et compteurs décrits dans le paragraphe précédent.

paramètre	valeurs pa défault	situation	valeurs limites
N (compteur)	5	Optionnel	entre 1 et 10
T1 (en s)	10 + / - 2	Recommandé	entre 0.5 et 30
W (en s)	60 + / - 5	Recommandé	entre 1 et 180
I (en s)	300 + / - 10	Recommandé	entre 5 et 900

table 1.1.9; valeurs proposées par le profil CNMA pour les temporisateurs de classe 4

Toutes ces recommandations sont censées permettre de meilleures performances sans altérer le niveau de sûreté requis pour cette couche. Le guide d'implémentation de CNMA précise cependant que les défauts du profil n'apparaîtront qu'après expérimentation.
Enfin, un tel profil n'a d'intérêt que lorsque tous les éléments du réseau suivent ce même profil. C'est l'hypothèse retenue pour ces travaux. Par conséquent, les tests seront effectués entre deux entités qui appliquent scrupuleusement les recommendations du profil.

Ceci clôt la description du protocole de Transport OSI Classe 4 appelé plus communément TP4. Notre but était de donner au lecteur éventuel de la thèse des éléments de compréhension du protocole afin d'appréhender plus facilement le chapitre sur l'implémentation.

1.1.2 Panorama des implantations

Comme nous l'avons fait pour les protocoles de communications, nous ne pouvions commencer nos travaux sans étudier les projets parallèles d'implémentation hautes performances de protocoles de niveau Transport liés à une architecture cible.

Et, si la littérature est prodigue en études liées à des implémentations logicielles, ou encore à des implémentations en environnement classique (microprocesseur CISC hôte d'un mini-ordinateur) utilisant un système d'exploitation tout aussi classique [Aronoff 87] [Clark 89] [Heatley 89] [Meister 87] [Mills 85] [Simon 86], elle est par contre beaucoup plus avare d'expériences matérielles.

La conception de circuits digitaux dédiés est une tâche complexe; le sujet a été à ce jour abordé sous quatre angles différents:

. Les circuits de communication dédiés où le protocole est intégralement cablé, conçus à partir de l'assemblage de composants discrets spécifiques et commerciaux. L'exemple retenu est DATAKIT [Fraser 79].

. à base de microprocesseurs standard (où le protocole est généralement programmé), que ce soit sur support traditionnel [Kaiserswerth 90] [Lasker 84] ou dans une machine spécialement conçue (comme dans le projet européen ESPRIT Supernode II [Esprit 89]).

. les circuits intégrés entièrement cablés où le protocole est en général micro-programmé [Krishnakumar 89] [Ansade 88] [Aoki 86].

. la conception de carte ou de circuit qui résulte de l'assemblage d'éléments ASIC et/ou de circuit VLSI standard [Smith 86] [Chesson 87] [Kanakia 88].

L'intérêt de ces quatre expériences réside dans le fait qu'elles partent de constatations communes:

. le débit des réseaux augmente très vite et nécessite l'implémentation sur circuit spécialisé des protocoles;

. les protocoles existants ne sont pas adaptés aux besoins temps réels, donc aux débits élevés.

Les solutions proposées sont voisines, même si elles mettent en oeuvre des moyens propres:

. concevoir des protocoles "allégés" (Datakit et PE) ou implémenter les protocoles existants avec des techniques dédiées aux hautes performances (MC3);
. exécuter ces protocoles sur des circuits spécialisés;
. exploiter le parallélisme dans les protocoles.

1.1.2.1 Protocol Engines, un circuit de communication en quatre boîtiers ASIC

Protocol Engines Incorporated est la compagnie qui est à l'origine de l'étude, de la conception et du développement du protocole de "Transfert" (i.e couche Transport et Réseau confondues) XTP [XTP 90] (ou Xpress Transfer Protocol) et de son circuit hôte, PE (pour Protocol Engine) [Chesson 89].

-XTP appartient à la prochaine génération de protocoles. Il a été conçu pour faciliter son implémentation sur circuit de communication VLSI.

-PE est un circuit de communication utilisant la technique VLSI destiné à implanter des

protocoles de communication et des systèmes d'interface.

Les deux projets sont développés parallèlement par PEI, bien que PE soit complètement indépendant du protocole XTP. Le but ultime de PEI étant d'associer XTP et PE dans un environnement de communication de niveau trois et quatre et d'atteindre, dans un premier temps, des performances de l'ordre de 100 Méga-bits par seconde (et du Giga-bit à moyen terme).

XTP représente un compromis entre débit, temps de réponse, complexité, contraintes d'implémentation VLSI, services à rendre par un tel protocole, interfaçage avec l'environnement externe et prise en compte de l'évolution future des besoins en réseaux de communication. Nous avons maintes fois évoqué XTP dans le panorama sur les protocoles. Le lecteur trouvera une analyse plus précise du protocole dans [Sanders 90]. Le circuit de communication Protocol Engine reste intéressant à analyser.

PE est un environnement VLSI hautement parallèle pour l'implémentation de protocoles de niveaux Réseau et Transport [Schwaderer 90]. Il représente un compromis entre la flexibilité de la programmation et les performances du hardware. PE peut être configuré pour supporter simultanément divers types de média, de bus hôtes ou encore de protocoles.

PE exploite le parallélisme dans la structure du protocole. Les concepteurs de PE sont pour cela parti du constat qu'un protocole peut être divisé en machines indépendantes; par exemple la machine de flux, machine d'erreurs, machine de résolution des adresses, machines d'interfaces. Chacune de ces machines s'articule autour de fonctions qui peuvent être, selon leur coût, leur nature et leur complexité,

-cablées:

. calcul et vérification du Checksum;
. détection d'erreurs;
. séquencement des TPDU;
. modification des champs dans une TPDU;
. calculs d'adresses.

-programmées (en faisant référence à des fonctions de base cablées):

. construction d'en-tête;
. transmission de paquets;

. DMA hôte;
. gestion des buffers.

Afin de garantir la flexibilité de PE, ses quatre circuits de base sont construits sur le même modèle:

-une partie cablée qui contient un ensemble de machines d'états finis, de séquenceurs spécialisés ou de circuiterie d'interfaçage de bus;
-un microprocesseur RISC très rapide à usage général;

-une zone de mémoire pour implanter le code à exécuter.

L'architecture de base de PE, organisée autour de deux bus (un bus de données et un bus de contrôle), est décrite figure 1.1.10.

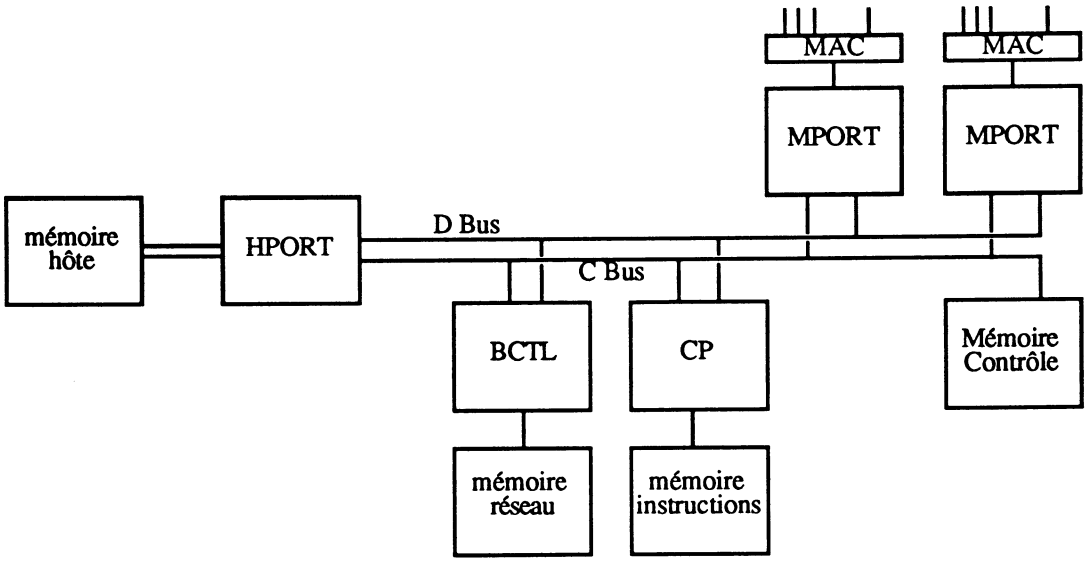


figure 1.1.10; architecture du circuit PE

A Chaque composant est affectée une tâche bien précise:

-HPORT fournit un Contrôleur de DMA intelligent pour connecter PE à un hôte. HPORT s'occupe en outre d'évaluer le checksum des TPDU transmises et de gérer le transfert vers la mémoire hôte.

-MPORT occupe le même rôle vis à vis de la couche MAC; chaque MPORT étant dédié à un type de médium unique. En plus de ses fonctions d'interfaçage, il effectue les tâches suivantes:

vérification des TPDU, étude des en-têtes de TPDU, identification du contexte, détection d'erreurs, vérification du numéro de séquence, contrôle de flux, résolution d'adresses et gestion des Fifo sur le D Bus.

-BCTL contrôle le stockage des TPDU et autres TSDU en mémoire. Toute information transférée passe nécessairement par le BCTL et la mémoire qui lui est associée. Le BCTL est encore l'arbitre du D Bus, il gère des queues de priorité différente et traite les paquets hors séquence.

-CP est un composant principalement programmable qui intègre les fonctions plus complexes à cabler telles la gestion des connexions, la mise à jour du contexte de connexion, la gestion des acquittements.

Le circuit dispose de surcroît de plusieurs mémoires, chacune dédiée à une tâche bien précise:

-La mémoire réseau reçoit toutes les données transmises par le circuit.

-La mémoire de contrôle fournit des informations sur le contexte de mémorisation des informations, sur la résolution des adresses et sur les contextes de connexion.

1.1.2.2 Parallel Protocol Engine, une architecture parallèle à base de Transputers

Un second projet, parallèle au notre à la fois en ce qui concerne le but des travaux et la chronologie, traite de l'étude d'architectures de circuits de communication pour des protocoles de niveau Transport à base de microprocesseurs RISC. Ce projet, nommé Parallel Protocol Engine est mené au centre de recherche IBM de Zurich [Kaiserswerth 89]. Partant d'une implémentation en C du protocole LLC type 2 [IEEE 85] et de l'analyse de ce protocole, ils ont déduit une architecture flexible à base de Transputers [INMOS 89a] (rappelons que LLC se trouve dans la pile OSI entre la couche MAC et la couche 3).

Le choix d'une architecture flexible à base de microprocesseurs standard permet l'implémentation logicielle performante de n'importe quel protocole, et surtout permet d'implémenter une pile complète de protocoles; ce qui représente un compromis coût-flexibilité très intéressant.

Le fait d'avoir choisi pour cette architecture le Transputer fournit un système puissant, aisément reconfigurable qui favorise la communication entre les processus. Le Transputer permet en effet

et de faire coïncider la notion de processeur avec celle de processus.

Ainsi la démarche suivie pour définir l'architecture cible est issue de l'analyse des parallélisations possibles dans un protocole de communication. A savoir

-Exécution en pipe-line des diverses couches.

-Au sein d'un protocole, le parallélisme peut être exploité de diverses manières:

. le protocole peut être divisé en plusieurs processus plus ou moins indépendants (par exemple divisés par connexion), processus qui peuvent être exécutés parallèlement moyennant des outils de synchronisations sur des ressources communes;
. certaines fonctionnalités du protocole peuvent être associées à un processeur particulier. C'est typiquement le cas du checksum.

Chaque module parallélisable du protocole est alors assimilé à un processus (donc à un processeur selon la philosophie du Transputer). Ces processus qui sont synchronisés par lien série, sont divisés pour fonctionner indépendamment. Ils ont la possibilité, en cas de besoin, de partager des ressources pour communiquer.

Pour des problèmes d'utilisation de ressources communes, tous les parallélismes cités ne sont pas nécessairement opportuns. C'est à partir de l'étude du protocole LLC qu'a été définie l'architecture cible PPE, basée sur le principe du "pipe-line" (figure 1.1.11). Le but de PPE est d'implémenter un ou une pile de protocoles en y exploitant tous les types de parallélisme, mais en minimisant les synchronisations et autres accès aux ressources partagées.

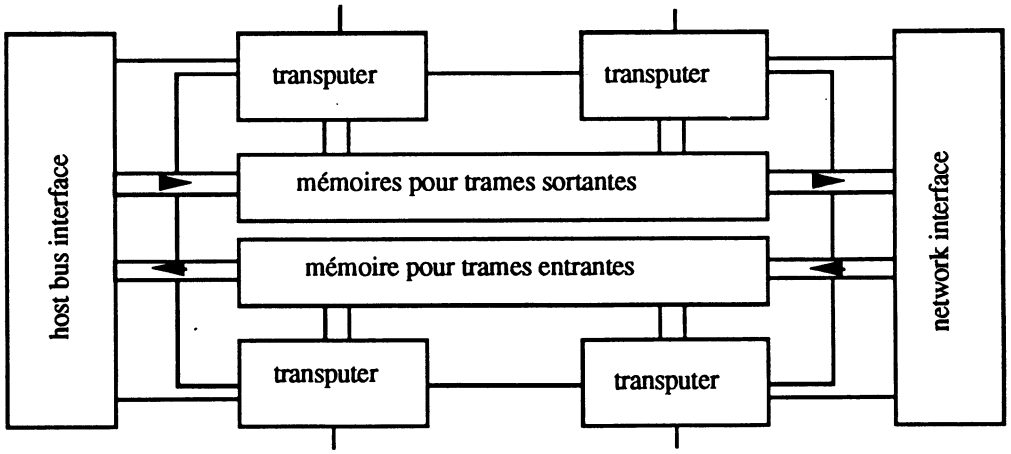


figure 1.1.11; architecture de PPE

L'architecture de base comporte deux mémoires dédiées chacune à une direction de données; mémoires qui sont partagées par le processeur qui implémente le protocole, le système hôte et l'adaptateur MAC.

1.1.2.3 DATAKIT, l'expérience cablée des Bells Laboratories

Datakit est le plus ancien, et sans doute le plus ambitieux des projets étudiés [Fraser 79]. Le but des Bells Laboratories étant de proposer un réseau universel complètement câblé, peu coûteux, et utilisable dans tout type d'application. Datakit est conçu comme un assemblage de briques, ou modules, à partir desquels on construit la complexité d'un noeud ou d'un élément terminal du réseau. La flexibilité du réseau est garantie dans la mesure où un noeud peut être remplacé par un réseau sans apporter d'autre modification au réseau général. Seule l'interface avec l'hôte reste programmée.

Le protocole à commutation de paquets conçu pour être supporté par hardware résulte de l'analyse des besoins futurs, des contraintes d'implémentation câblée et des objectifs du projet en terme de simplicité, rapidité, flexibilité et sécurité. Le modèle proposé fonctionne en trois couches:

-A correspond au niveau physique de l'ISO.

-B fournit des fonctions qui permettent de multiplexer et commuter les flux d'octets. Il vérifie et éventuellement détruit les données mauvaises au fur et à mesure de leur transfert sur le réseau et fournit en plus les facilités d'adressage nécessaires.

-C traite chaque circuit virtuel (ou encore connexion) séparément de bout en bout sur le réseau.

Pour simplifier l'interface entre B et C, C impose à son utilisateur une fenêtre par circuit virtuel. Ainsi, il n'y a pas de contrôle de flux en cours de transfert. A et B sont les niveaux minimum au dessus desquels un utilisateur peut interfacer n'importe quel protocole.

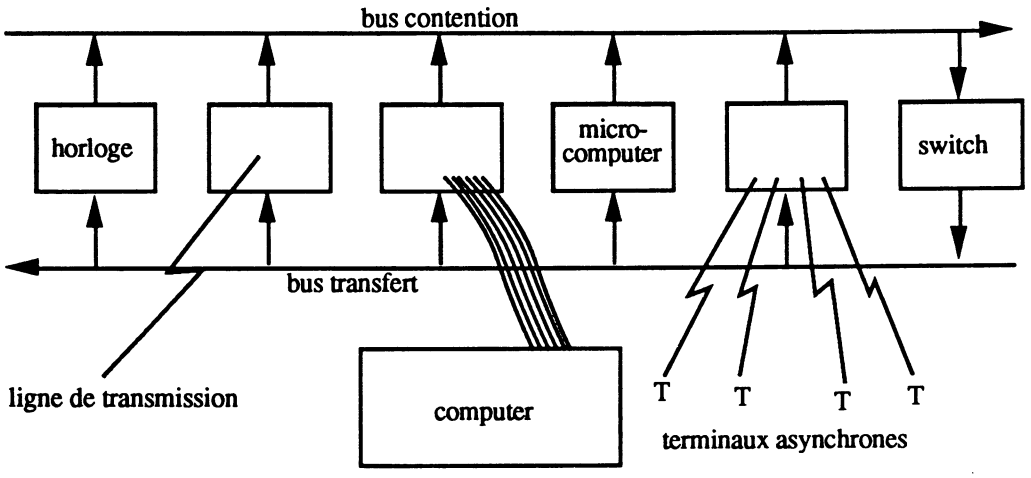


figure 1.1.12; architecture de DATAKIT

Un noeud Datakit (figure 1.1.12) est formé par deux bus entourant un nombre de modules variables selon la complexité du noeud. Un module qui veut émettre un paquet doit obtenir l'accès au bus contention, faire passer son paquet au module switch qui le met enfin sur le bus de Transfert. Les modules sont organisés sur le bus par adresses croissantes de sorte que l'accès au bus est réglé par priorité associée à l'adresse des modules. Le routage des informations entre éléments terminaux est classique, chaque noeud intermédiaire étant chargé de la gestion des "troncs de ligne" auxquels il accède. Cette interface garantit à chaque circuit virtuel assez de mémoire pour stocker au moins une fenêtre de données (telle que définie par le niveau C).

Le temps de transfert dans un noeud élémentaire à été évalué par [Luderer 81] à 100 µs minimum. Sur un réseau expérimental de 380 terminaux asynchrones connectés au réseau par interface RS232, le débit du réseau plafonne à 48 Kilo-bits par seconde en transfert de fichier.

Malgré l'échec de cette tentative de conception du réseau universel (les performances obtenues sont en effet très faibles), cette expérience à permis à ses auteurs de définir les conditions d'une communication performante, simple à implanter sur circuit de communication câblé:

. Chaque circuit virtuel définit une connexion point à point.
. Les informations transportées sont de deux types seulement: contrôle ou données.
. Un circuit virtuel ne duplique pas ou ne réarrange pas les données; la séquence est préservée.
. Destruction des informations corrompues sur le réseau.
. Les données perdues sont retransmises de bout en bout sur le circuit virtuel.
. Chaque circuit virtuel a une capacité de stockage définie à l'ouverture du circuit.

1.1.2.4 MC3, conception d'un circuit de communication pour l'implémentation des couches trois à cinq

Le micro-contrôleur configurable MC3 est un circuit ASIC conçu pour des applications dans le domaine des communications [Ansade 88]. MC3 est primitivement prévu pour intégrer les couches 3 à 5 (selon les scénarios 3,4,5 ou 3,4 ou 4,5 ou 3 ou 4 ou 5) des normes OSI de l'ISO.

L'architecture multi-processeurs du micro-contrôleur est la suivante [MC3 87] (figure 1.1.13):

. l'Interpréteur de Processus (IP), à architecture RISC
(76 instructions dont 75 % peuvent s'exécuter en 2 cycles);
. le Noyau Système NS;
. les Machines de Transfert (reliées à des interfaces séries);
. une interface parallèle compatible avec la famille 68000 [EF68000 79];
. une mémoire RAM interne de 4 Kilo-octets et une mémoire ROM interne de 16 Kilo-octets;
. deux bus systèmes internes de 16 bits et un bus mémoire privé de 16 bits.

L'Interpréteur de Process est un processeur 16 bits orienté machine à pile. Il utilise un jeu d'instructions réduit. Les principales caractéristiques de l'IP sont les suivantes:

. Son architecture RISC lui permet d'atteindre de hautes performances grâce à l'utilisation d'un jeu d'instructions réduit et adapté au traitement des protocoles de communication. Le jeu d'instructions inclut des instructions de contrôle, de manipulation de pile et des registres programmables, d'accès mémoire, d'opérations arithmétiques et logiques. Il comporte encore un certain nombre de primitives systèmes dont l'exécution est confiée au Noyau Système.

. L'IP utilise de la mémoire interne rapide (avec un temps d'accès d'un cycle) et un haut degré de parallélisme matériel; il peut, en plus des 3 espaces mémoires auxquels il a accès (correspondant à sa mémoire ROM interne, à sa mémoire privée, aux piles de données et de retour et à la mémoire de contexte), accéder aux zones mémoires externes éventuellement partagées avec un microprocesseur externe .

. La commutation de tâches est très rapide; l'IP l'exécute en 5 cycles au plus et est en cela 54 fois plus rapide qu'un 80386 (d'INTEL) à horloge équivalente. Une des

principales caractéristiques de l'IP est de pouvoir faire résider plusieurs processus en interne sans nécessiter de ressources extérieures, ainsi que d'intégrer un mécanisme câblé de commutation de contexte.

Le Noyau Système est un processeur câblé chargé d'exécuter des primitives systèmes de gestion de processus, de gestion des ressources (canaux de communication entre les processus, temporisations), ainsi que les interruptions externes. La commutation de processus est effective à chaque exécution d'une primitive système; excepté pour les primitives de manipulations du temps (*Set.timer* et *Reset.timer*) qui ne provoquent pas l'arrêt du processus qui les exécute (fonctionnement justifié par le rôle de démarrage des alarmes de ces primitives);

Les Machines de Transfert sont des machines câblées capables d'assurer les transmissions de données sur des structures de données prédéfinies. Deux machines de transfert (une en émission et une en réception) sont utilisées pour la communication avec les couches hautes (respectivement basses) relativement à celle(s) à traiter par MC3

Les différents types de transmission implémentés sont:

. mémoire (interne ou externe) MC3 vers une ligne série;
. mémoire (interne ou externe) MC3 vers une mémoire (interne ou externe);
. ligne série vers mémoire externe MC3.

Dans le cas d'une communication par mémoire partagée, la gestion de la synchronisation entre MC3 et le microprocesseur est de type asynchrone . Dans le cas d'une communication par ligne série, la synchronisation est gérée au niveau des interfaces séries et est de nature synchrone.

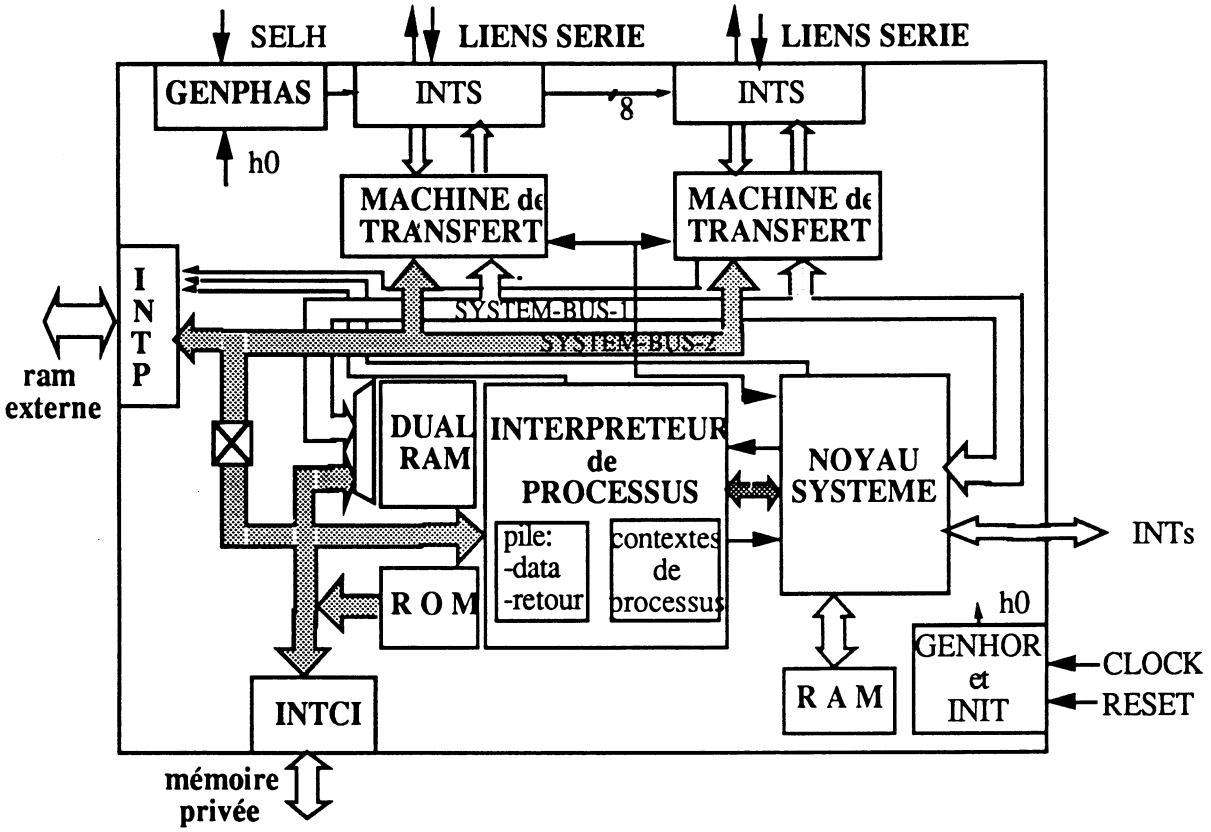


figure 1.1.13; architecture de MC3

L'architecture de MC3 a fait l'objet d'une simulation logique effectuée machine par machine (IP, NS, MT, Interface Parallèle, Interface Série) jusqu'au niveau des portes logiques. Pour réaliser cette simulation fonctionnelle, nous avons dû construire une maquette de simulation fonctionnelle, réalisée à partir de microprocesseur Transputer [Diot 88].

Le Transputer est un des microprocesseurs cible du circuit de communication MC3. Ils cultivent ainsi bon nombre de similitudes qui permettent l'utilisation directe du Transputer pour la simulation de MC3 (i.e. la notion de canal physique, ou encore de processus sont identiques). Deux modules propres au fonctionnement de la maquette et aux exigences d'une telle simulation ont dû être réalisés:

-un module d'observation des états des différents modules de MC3 et,

-un module de simulation de l'environnement de fonctionnement des différents modules de MC3 (initialisation, génération d'interruptions).

Les validations fonctionnelles et comportementales du circuit, menées à partir de ces outils,

nous ont permis de montrer que MC3 permettait d'atteindre, pour un protocole de type Transport classe 0, des performances de l'ordre de 4000 TSDU par seconde, observés indépendant de la taille des TSDU.

MC3 démontre que l'implémentation des protocoles de niveau trois et quatre sur circuit de communication spécialisé permet d'en accroître considérablement les performances.

1.2 Sélection de Critères d'implémentation Hautes Performances

1.2.1 Les environnements d'implémentation

1.2.1.1 Caractéristiques de la fonction Transport

La couche Transport, de par ses services et fonctionnalités, peut être implémentée dans un environnement réseau restreint, sans que cela ne nuise à l'étude de performances ultérieure. La présence d'un utilisateur du service Transport est cependant nécessaire pour la mise au point et l'étude de la couche. Nous avons implémenté un processus d'intéraction avec l'utilisateur du protocole de Transport qui nous a permis selon l'avancement de l'étude:

. de tester les différentes branches de la machine d'état fini du protocole par écriture de jeux de tests sélectifs;
. de faire fonctionner l'application entre deux utilisateurs, en temps réel, sur deux sites hôtes distants
. de saturer l'application pour en étudier le débit et les limites sur les architectures étudiées.

Le problème du réseau sous-jacent est tout autre. La présence des couches Physique à Réseau n'est pas nécessaire quand il s'agit de développer des techniques d'implémentations propres à la couche Transport. L'effet des couches basses sur notre protocole pourrait même devenir nuisible dans certaines conditions d'analyse (les délais introduits par les performances des couches basses pourraient réduire le débit d'information en réception basse, ce qui fausserait les évaluations au niveau du protocole de Transport).

Afin de tester le fonctionnement du modèle de structures de données sur plusieurs protocoles consécutifs, et d'étudier la complexité des algorithmes développés à cet effet, nous avons dû implémenter un niveau inférieur. Nous avons donc construit, en lieu et place du service de couche Réseau, un processus qui permet de traiter à la fois les NSDU transmis par le protocole de Transport, et les NPDU transmis entre deux entités paires éloignées. Ceci avec de multiples avantages:

. vérification des structures de données construites au niveau 4 et analyse du transfert de données par listes partagées;

. transmission, indépendamment de la couche quatre, des TPDU reçues de cette même couche vers l'entité distante;

. vérification de la cohérence des NPDU reçues de l'entité éloignée et formatage selon les structures de données dédiées.

Le protocole de Transport implémenté et les processus du "noyau réseau" forment un système qui fonctionne sur le modèle de structures de données et qui communique:

-Avec l'utilisateur du service transport selon trois modes différents:

. Transfert sur canal physique octet par octet;
. Transfert sur canal physique paquet par paquet (variante du mode précédent);
. Transfert par mémoire partagée.

-Avec l'entité distante par transfert de NPDU complets sur les liens physiques du Transputer.

Le mode d'interface sélectionné a fait l'objet, selon les caractéristiques de l'environnement hôte, d'une négociation préalable entre l'utilisateur du service Transport et le protocole. Nous évaluerons, dans le chapitre suivant, les performances des processus d'interface de manière à déterminer le mode le plus performant. La structure générale de l'application et de son environnement logiciel est synthétisée figure 1.2.1, indépendamment de l'implantation matérielle que l'on peut en faire.

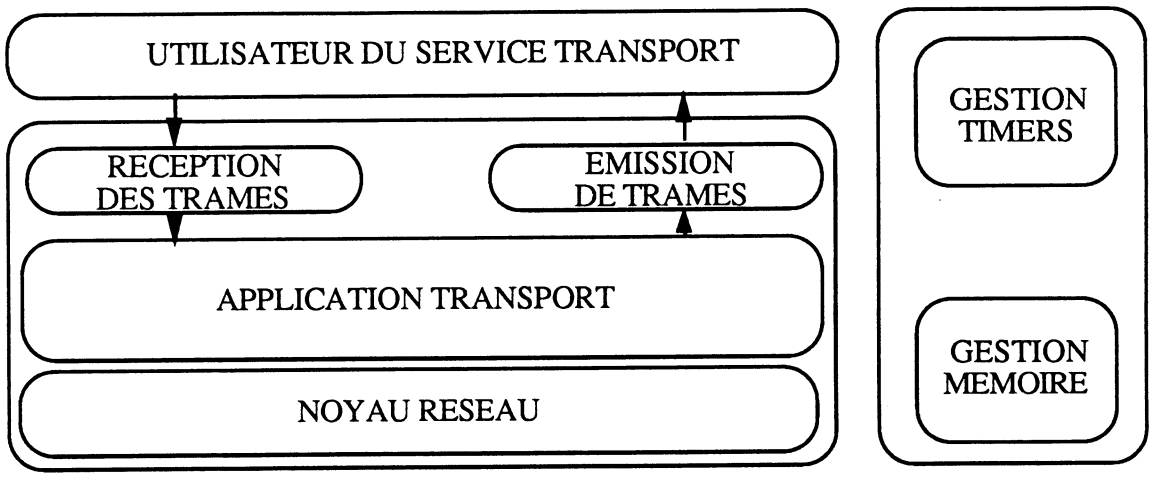


figure 1.2.1; organisation des processus développés autour du protocole de Transport

1.2.1.2 L' environnement de développement à base de Transputer

Suite à l'analyse des implantations (décrites au paragraphe 1.1.2), nous avons une connaissance plus précise:

-Différents aspects du parallélisme à envisager pour l'implémentation d'un protocole de Transport.

-Qualités requises pour concevoir un environnement d'implémentation flexible et performant.

-Problèmes liés à la difficulté de concevoir l'architecture d'un circuit de communication, compte tenu de la technologie, des caractéristiques du protocole et des contraintes de coût.

Le Transputer nous est naturellement apparu comme le processeur le mieux adapté pour mener à bien nos travaux. Ce choix à été guidé par l'analyse des intérêts et des limites de chacune des implantations étudiées. Parallèlement, une étude bibliographique préliminaire sur les performances de TP4 nous a permis d'observer que:

.les implémentations effectuées sur des systèmes généraux, tel UNIX ou VMS, sont généralement peu performantes [Lantz 84] [Vazquez 88].

.les mesures de performances effectuées sur machine à base de processeurs CISC (famille INTEL 80x86 ou Motorola 68000) sont pour le moins classiques et toujours loin des débits que nous visons [Heatley 89] [Lasker 84].

.[Colella 85] et [Watson 87] montrent que pour atteindre des débits élevés, le protocole doit exploiter au plus près les facilités de la machine, minimiser l'utilisation de tâches complexes pour la gestion mémoire.

.[Stokesberry 83] rajoute quant à lui que le traitement du protocole doit être parallélisé pour être plus efficace. Les tâches systèmes associées à la gestion de ce parallélisme doivent avoir un effet négligeable sur l'application, sinon les gains dûs au parallélisme s'effacent rapidement.

Les performances observées dans la littérature référencée se situent dans l'intervalle 250 à 500 TPDU de données par seconde. Rappelons que l'étude que nous allons entreprendre doit nous amener à

. atteindre 100 Méga-bits par seconde à partir d'un protocole standard (en l'occurrence OSI Transport, classe 4), ou entre 4000 et 5000 DT TPDU par seconde (le chiffre de 100 Mbps étant fixé pour être compatible avec les performances du réseau FDDI).

. analyser les problèmes d'architecture hautes performances liés à l'environnement d'implantation et à la nature du protocole.

L'environnement RISC semble donc prédisposé pour ce type d'étude. Un processeur RISC est en effet moins complexe qu'un CISC; ses instructions sont élémentaires et permettent de développer des primitives système de bas niveau. Les problèmes d'architecture sont plus simples à appréhender à partir de machines RISC car ces dernières sont conceptuellement plus proches des éléments d'un circuit de communication. De là à sélectionner définitivement le Transputer, il n'y a qu'un pas:

.Les microprocesseurs RISC disponibles dans le commerce (en 1987) sont pour le moins rares.

.Outre sa structure RISC, le Transputer offre des avantages certains par l'intégration d'une gestion du parallélisme, des communications et des temporisations câblées.

.Grâce à quatre liens séries (gérés par le processeur via un DMA câblé), la conception d'architecture multi-Transputers complexe est simple à réaliser, simple à modifier.

.La maquette de simulation de MC3 à été développée à base de Transputers; c'est encore le support retenu par [Kaiserswerth 89] pour leur architecture PPE.

L'implémentation à été écrite en OCCAM. Ce langage, qui suit le modèle CSP (défini pour la communication de processus parallèles par rendez-vous [Hoare 77]), a été conçu pour le Transputer, parallèlement à ce dernier. Le compilateur OCCAM exploite donc efficacement les services du Transputer. Malgré la pauvreté d'OCCAM en terme de structures de données et malgré le peu d'outils de mise au point disponibles dans l'environnement logiciel du Transputer, ce langage nous a permis de réaliser, dans de bonnes conditions, cette implémentation de TP4.

Présentation succincte du Transputer [INMOS 89a]

Le Transputer est un microprocesseur qui possède une mémoire interne et des liens séries pour connecter entre eux plusieurs circuits. L'architecture du circuit a été conçue pour simplifier la

conception de systèmes parallèles complexe par association de la notion de processus parallèle logiciel à la notion de processeur indépendant. Le Transputer est alors considéré lui-même comme le processus qu'il exécute. Construire un système revient à interconnecter une certaine quantité de Transputers. Cependant, plusieurs processus peuvent être implémentés en parallèle sur un unique Transputer. La gestion du parallélisme étant intégrée (câblée même) au circuit. Les capacités du Transputer pour simuler ou modéliser un circuit de communication complexe sont donc très intéressantes et simples à mettre en œuvre.

D'un point de vue architectural, il intègre, sur un seul circuit, un microprocesseur RISC 32 bits, 4 liens de communication série à 20 MHz (pour un débit efficace d'environ 12 Méga-bits par seconde), de 2 à 4 kilo octet de mémoire locale, les services classiques d'interfaçage mémoire et périphériques (figure 1.2.2).

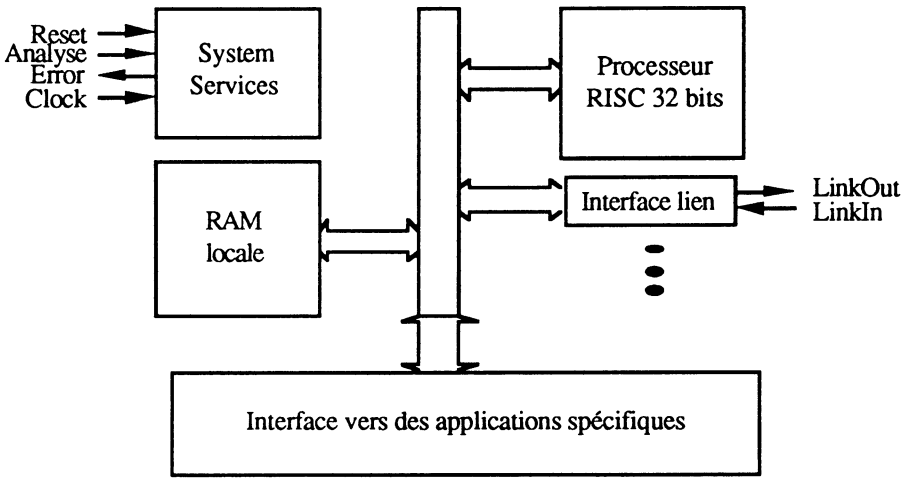
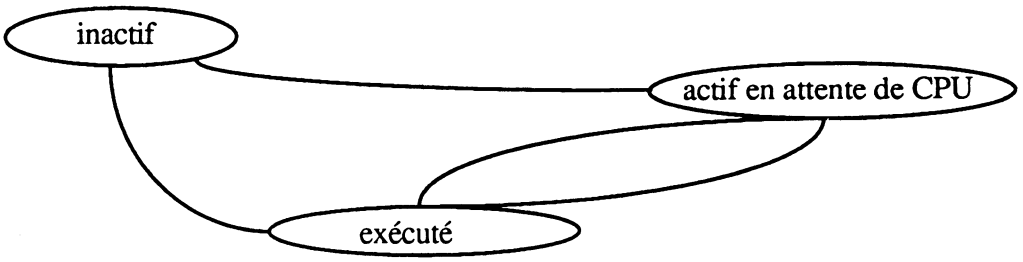


figure 1.2.2; architecture interne du Transputer

La gestion intégrée du parallélisme est un avantage certain pour le Transputer. Il est intéressant pour la suite de cette étude de détailler les mécanismes de gestion de la concurrence entre processus. Bien que le temps de commutation de processus soit inférieur à la micro-seconde, cette fonction va se révéler être un point névralgique dans la conception de l'application et des tâches système.

Le Transputer gère les processus parallèles actifs dans deux listes indépendantes associées chacune à une priorité différente. Un processus exécuté en priorité haute ne peut pas perdre la main au profit d'un processeur de priorité basse. Ainsi, les processus de priorité basse ne peuvent être exécutés que lorsque aucun des processus de priorité haute n'est actif.

Un processus peut prendre trois états :



Le passage d'un état à l'autre est effectué systématiquement, en priorité haute et basse :

. Lors des instructions de communication (émission ou réception d'une information sur un lien série). Dans ce cas, le processus qui arrive le premier au rendez-vous (puisque tout transfert par lien provoque un rendez-vous entre deux processus) passe en attente de rendez-vous, jusqu'à ce que le deuxième processus, alors inactif ou actif en attente de CPU soit exécuté par le Transputer. Il provoque, à son arrivée sur l'instruction de communication correspondante, le passage du processus précédent dans l'état "actif en attente de CPU".
Notons qu'une communication par lien physique est effectuée au travers d'un DMA câblé (soit 12 Mbps efficaces); une communication par lien virtuel (les 2 processus communicant sont installés sur le même Transputer) est réalisée par un simple transfert en mémoire (soit avec un débit de 80 Mbps).

. Sur les instructions d'armement de temporisation qui provoquent la mise en attente du processus. Dans les autres cas, les instructions de manipulation des timers ne perturbent pas l'ordre d'exécution.

En priorité basse seulement, les commutations de processus interviennent en temps partagé, toutes les milli-secondes, sur la prochaine instruction JUMP rencontrée dans le code. Il est impossible, en priorité basse, de contrôler l'avènement des tâches de commutation, si ce n'est en garantissant que les processus exécutés ne comportent pas de suites d'instructions (délimitées par des instructions de communication) de longueur supérieure à 1 ms.

Pour en finir avec ce survol des caractéristiques du Transputer, notons que ce processeur a été ***conçu pour communiquer avec l'environnement extérieur par lien***, et qu'à ce titre, il n'offre ***aucun moyen de faire de la communication par mémoire partagée***.

Une description approfondie du Transputer est effectuée dans [INMOS 88b] et [INMOS 89b]. Quant à OCCAM, [INMOS 88d], [INMOS 88c] et [Jeugt 88] en permettent une approche simple. OCCAM est, en effet, un langage très proche de Pascal côté syntaxe des constructions séquentielles. On doit cependant y rajouter les concepts de construction parallèles, rendez-vous sur lien série et commandes gardées [Dijkstra 65] ; autant de concepts familiers au modèle de référence : CSP.

L'application et son environnement ont été initialement développés et mis au point sur une carte à un Transputer [LL1T 89], hôte d'un micro-ordinateur de type PC-AT. Précisons enfin que pour des raisons de mise au point de l'application et aussi de disponibilité en taille de mémoire interne, les Transputer utilisés sont de modèle T425 [INMOS 89a] (ce dernier, contrairement au T414, permet le déboggage des processus exécutés en priorité haute).

Mais nous reviendrons plus loin dans ce chapitre, avec plus de détails, sur les limites imposées par l'environnement sur le développement de l'application.

1.2.1.3 Contraintes liées au Transputer

Nous ne voulons pas dans ce paragraphe présenter nos conclusions sur l'opportunité d'Occam et du Transputer dans l'étude d'implémentations performantes des protocoles de communication. Nous voulons plutôt montrer qu'un certain nombre de caractéristiques du microprocesseur nous ont obligé à reconsidérer des techniques d'implémentation utilisées habituellement pour ce type d'application, ou nous ont empêché d'approfondir certaines mesures comme nous l'aurions souhaité:

-Le Transputer n'offre aucune visibilité sur son comportement interne.

.Il ne fonctionne pas sur des registres prédéfinis (le Transputer alloue dynamiquement ses registres dans sa mémoire locale); il devient impossible de connaître, à un instant donné, le contenu des registres d'état du processeur.

.les outils de développement et de déverminage proposés par INMOS [TDS 87], rendent laborieuse la mise au point d'une application parallèle complexe.

.Nous n'avons pu effectuer aucune mesure de taux d'occupation du circuit par les processus exécutés, ni de proportion d'accès mémoire.

.S'il est possible de connaître le séquencement provoqué par le Transputer dans la liste des processus actifs, il est impossible de protéger un processus dynamiquement contre la perte d'activité.

.Le Transputer ne possède que deux priorités, avec lesquelles il faut gérer (1) les processus de l'application qui peuvent être interrompus (priorité basse), (2) ceux qui doivent être exécutés en priorité haute pour garantir qu'ils accéderont bien en exclusion mutuelle à la mémoire partagée, (3) les outils de mesure de performances et d'espionnage du protocole qui doivent eux aussi, pour des questions de hiérarchie entre les processus être exécutés en priorité haute (afin d'avoir une vue d'ensemble de l'application) et enfin (4) les tâches systèmes qui sont théoriquement indépendantes du reste de l'application.

-Le Transputer n'offre aucune facilité pour la communication non blocante par mémoire partagée. Toute exclusion mutuelle doit être effectuée par communication si l'on veut qu'elle soit efficace en environnement multi-processeurs. Ainsi le polling, technique usuelle sur des microprocesseur classiques est d'un usage délicat sur Transputer. Un processus en attente sur un événement, s'il n'effectue cette attente sur un lien série OCCAM, est actif et, n'effectuant pas de communication, ne pourra être désactivé au profit d'un autre processus de même priorité qu'après 1 ms d'exécution (durée d'un timeslice sur Transputer). Soit, sur un processus producteur/consommateur classique, le producteur ne disposera (dans le meilleur des cas) que de 50% du temps total pour s'exécuter, les autres 50% étant consommés par le processus consommateur en attente passive. Le polling est contraire à la philosophie du Transputer qui à été conçu pour que des attentes d'événement se fassent sur un canal, garantissant ainsi que les processus en attente sont inactifs, donc ne consomment pas de temps CPU, si ce n'est environ 40 cycles pour prendre la main après que le rendez-vous n'ait été réalisé.

1.2.2 Conditions d'implémentation hautes performances

Les évaluations de performances menées sur des implémentations logicielles de la couche Transport ont comme principal intérêt d'avoir permis l'étude précise des défauts et qualités du protocole et d'avoir établi un jeu de valeurs optimales pour les paramètres du protocole (tels la taille des TPDU, la taille de la fenêtre de contrôle de flux, le nombre de connexions,...) [Strayer 88] [Svodobova 89] [Stokesberry 83]. Toutes ces études de synthèse, bien que réalisées sur des systèmes et dans des langages complètement différents convergent quant aux conclusions obtenues.

L'implémentation que nous avons réalisée a été écrite:

. Pour développer et montrer l'efficacité des techniques (structures de données spécialisées et autres algorithmes) qui permettent d'atteindre des débits élevés

. Pour exploiter avec un maximum d'efficacité son environnement de développement; nous avons développé à cet effet des primitives systèmes de gestion de ressources.

. Pour étudier les problèmes d'architectures multi-processeurs et aider à la conception de circuits de communication à hautes performances.

Seuls les travaux relatés dans [Lasker 84] présentent quelques éléments de proximité avec notre démarche. Notons qu'en ce qui concerne les problèmes d'implémentations logicielles, on peut trouver des études théoriques chez [Bochman 83], [Bochman 88] et [Serre 86].

Cette étude diffère des implémentations habituelles par la démarche entreprise. Pour atteindre les objectifs que nous nous sommes fixés, nous redéfinissons complétement un système flexible d'implantation, en évitant toute hypothèse qui pourrait être génante pour les observations ultérieures. Ainsi, nous ne faisons pas d'hypothèses à priori sur le parallélisme potentiel du protocole implémenté, mais nous nous donnons les moyens d'étudier l'opportunité des différents aspects du parallélisme.

La culture de base de ces travaux s'appuie presque exclusivement sur les résultats obtenus dans l'étude comportementale du circuit MC3 [Diot 88] et sur un rapport de recherche du centre scientifique IBM de Zurich qui s'avère être une synthèse très complète des diverses implémentations logicielles effectuées et de leurs conclusions [Svodobova 88].

Nous avons commencé par définir les raisons pour lesquelles les performances des implémentations analysées étaient réduites, et cherché à localiser, dans la structure complète du protocole, les causes de ces mauvaises performances.

Les problèmes liés à la nature même du protocole, ainsi qu'à ses fonctionnalités ont été identifiés puis analysés. Les solutions ont été étudiées à la fois du point de vue de l'implémentation et de l'environnement d'implémentation qui, à nos yeux, représentent un tout indissociable. Nous avons été amenés à définir un ensemble de contraintes sans lesquelles des performances élevées auraient été impossibles à atteindre dans un environnement à base de Transputer:

. minimiser les accès aux listes de données décrivant les TPDU (ou **minimiser les indirections** sur les ressources);

. **minimisation des communications par canaux**;

. **modularité maximum** de l'application et paramétrisation maximum, de façon à obtenir une implémentation flexible du protocole (même si cela doit entraîner dans un premier temps quelques pertes au niveau des performances);

. entre les processus, exploitation maximale, quand elle est sûre, de la **communication par mémoire partagée**.

Contraintes que nous avons appliquées avec rigueur lors de l'implémentation des

-Primitives systèmes de base (dédiées à la **gestion de ressources** ou encore à la **gestion des temporisations**).
-**Processus d'interface aux bornes du système** (acquisition et émission des informations).
-**Fonctionnalités et services coûteux du protocole.**
-**Techniques de synchronisation (donc de la hiérarchie) entre les divers processus** de l'application .
-**"Transferts de données"** entre les divers processus parallèles (par listes partagées).

Nous décrirons donc dans la suite de ce chapitre chacun de ces points avec les alternatives qui se sont présentées et les choix définitifs réalisés pour l'implémentation.

1.2.2.1 Conception d'un modèle de structures de données

Mis au point dans le cadre du projet MC3 [Dang 88a], le modèle de structures de données définit un ensemble de descripteurs et de règles de chaînage qui permet de traiter un protocole ou une suite de protocoles sans effectuer aucun mouvement de données. Deux transferts suffisent pour traiter une suite de protocoles: un en réception, puis un ultime après traitement par toutes les couches en vue de l'émission vers l'environnement extérieur.

En réception, les informations entrantes sont prises en charge par un processus qui les transfère dans une liste de demandes. Chaque SDU ou PDU y est décrit par un unique descripteur de demande (figure 1.2.3) qui contient, outre des informations de chaînage et de contrôle, une zone de taille fixe dans laquelle peuvent être stockées tout ou partie des données entrantes. La

suite de ces données utilisateurs est alors mémorisée dans un ou plusieurs tampons de données que le descripteur de demande accède via des descripteurs de tampon de données (figure 1.2.4).

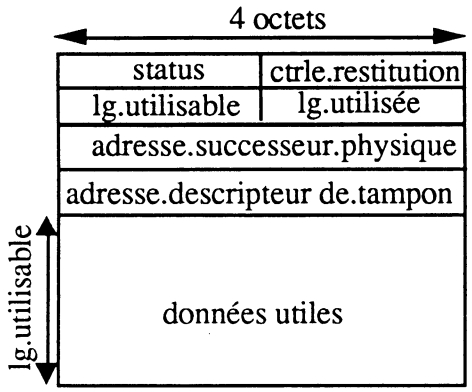


figure 1.2.3; descripteur de demandes

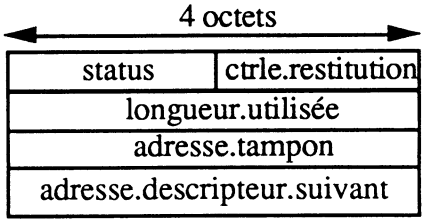


figure 1.2.4; descripteur de tampon de données

La structure d'une liste de demande ainsi construite est présentée figure 1.2.5.

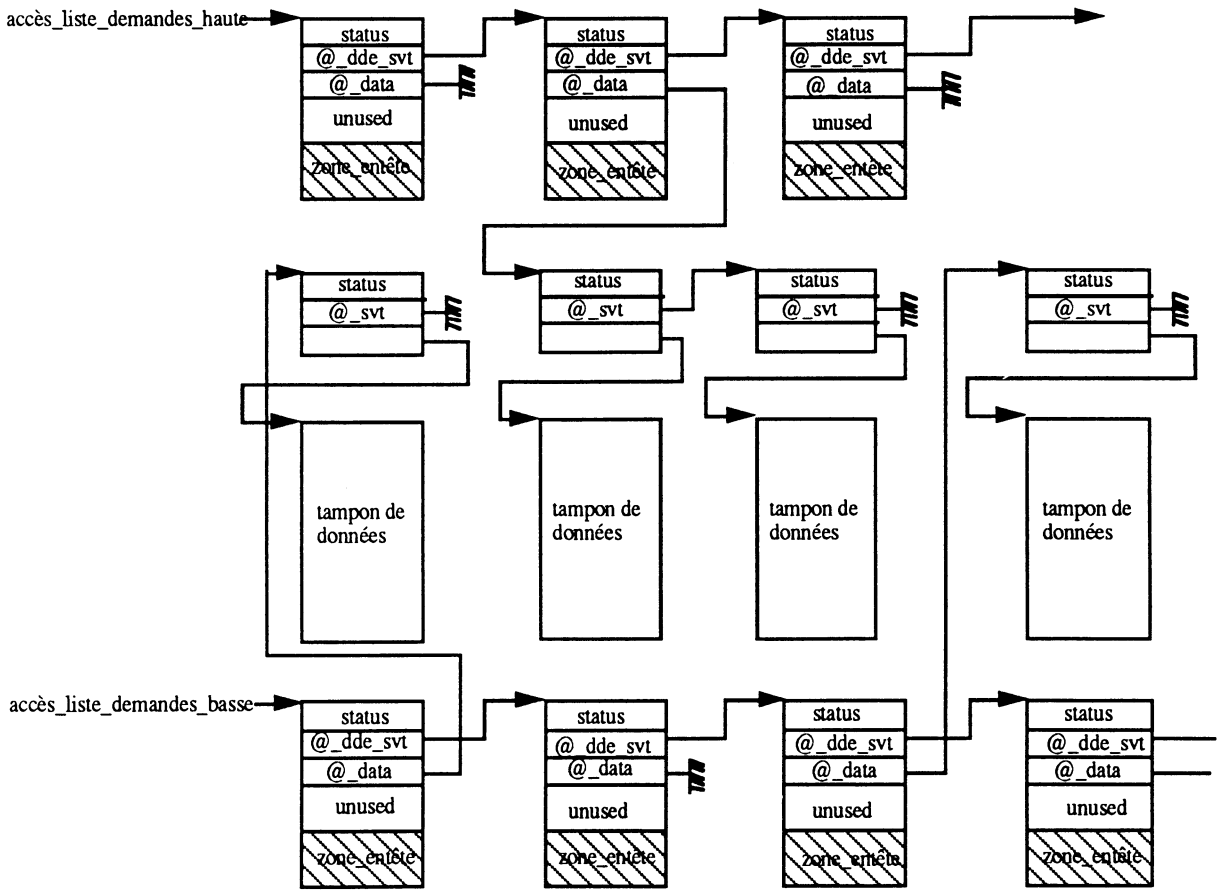


figure 1.2.5; listes de demandes en réception haute et basse

Une fois insérées dans une liste de demandes, les données entrantes sont traitées par les protocoles successifs sans jamais être déplacées. La structure de PnPDU (Process n Protocol Data Unit), présentée figure 1.2.6, permet à un protocole de décrire les données de l'utilisateur mémorisées dans une liste de demande; et de construire les PDU à transmettre au protocole adjacent par chaînage d'une séquence de PnPDU.

type.pnpdu	status
numero	type.ressource.décrite
longueur pnpdu	
adresse.pnpdu.sucesseur.physique	
adresse.pnpdu.sucesseur.logique	
adresse.ressource.décrite	
adresse.pere.pnpdu	
adresse.fin.pnpdu	
adresse.debut.pnpdu	

figure 1.2.6; modèle d'un PnPDU

Tout protocole qui utilise ces structures de données reçoit en entrée une arborescence de PnPDU qui décrivent une liste de demande (sauf le premier protocole qui travaille directement sur la liste de demande). Il construit alors son propre niveau de PnPDU sur cette arborescence (figure 1.2.7).

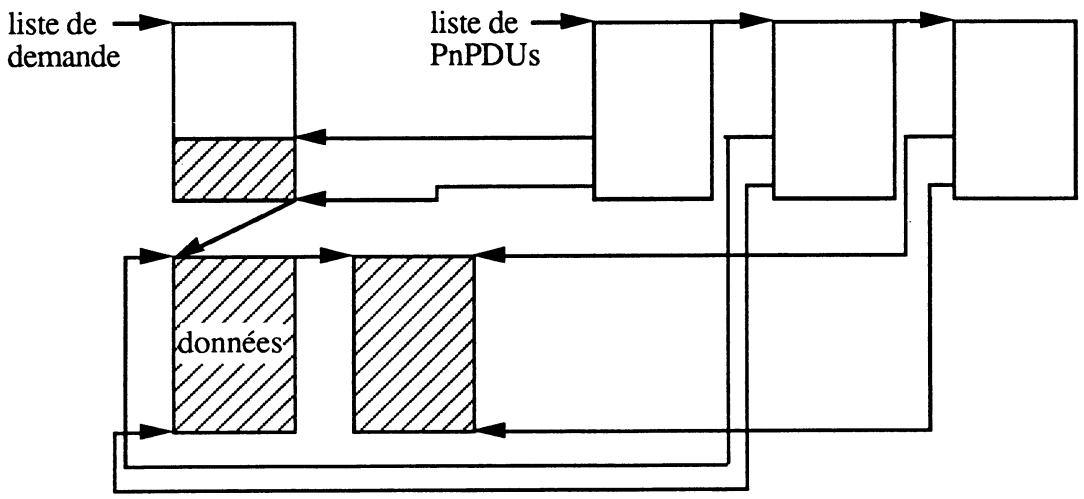


figure 1.2.7; relation liant les PnPDU aux listes de demandes

Une arborescence de PnPDU est donc partagée par les n protocoles du système; ainsi que par un processus d'émission situé aux bornes du systèmes. Le rôle du processus d'émission est de parcourir l'arborescence, et d'en extraire les informations à émettre (vers l'environnement externe du système). L'utilisation des PnPDU présente de multiples intérêts:

. suppression des transferts de données qui peuvent représenter, dans une implémentation classique, jusqu'à 80 % du temps de traitement d'une PDU;

. suppression des transferts de données entre les couches; les transfert de données ont lieu aux bornes du système exclusivement;

. l'émission aux bornes du système est obtenue par un parcours récursif de l'arborescence de PnPDU, indépendant du nombre de protocoles traités; mais aussi indépendant du sens de traitement (réseau vers hôte ou hôte vers réseau);

. décrire les données utilisateur permet de résoudre simplement les problèmes liés aux fonctions complexes définies pour ce type de protocole (segmentation ou multiplexage);

. associés à la structure de liste de demandes, ils rendent le **débit du système "quasiment indépendant" de la taille des TSDU traités**.

Le fonctionnement de ces PnPDU sur les listes de demandes est décrit pour les couches Réseau et Transport en annexe A (réception de NPDU et réassemblage de TPDU pour l'utilisateur du service Transport; puis segmentation des TSDU issu du service transport et construction de NPDUs).

Mais, le format de ces descripteurs ne fait que définir un modèle de structures de données qui,

. peut s'avérer d'un usage dangereux (et pourquoi pas nuisible aux performances de l'application) quand le modèle est mal utilisé;

. reste améliorable compte tenu des caractéristiques système de l'environnement et des propriétés des protocoles implémentés.

Il est par conséquent important, avant d'envisager la mise au point du modèle, de respecter certaines règles indépendantes des conditions d'utilisation. Pour s'affranchir des mouvements de données, nous avons adopté la notion de description. Le traitement des données utilisateur

s'opère dans ces conditions par indirections successives sur des listes de descripteurs; ces listes (ou arborescences) étant partagées entre deux ou plusieurs processus. Les problèmes à résoudre deviennent:

. Garantir l'exclusion mutuelle pour l'accès aux descripteurs.
. Minimiser, pour chaque processus les accès aux listes partagées, soit minimiser le nombre des parcours et la durée des recherches.

. Réduire la complexité des arborescences de PnPDU et des listes de demandes en restituant dès que possible les ressources occupées par les informations devenues obsolètes; mais aussi en minimisant le nombre de PnPDU nécessaires à la description d'une TPDU.

. Minimiser le nombre d'indirections nécessaires à la recherche d'informations. Pour obtenir un octet dans un tampon de données par exemple, on doit accéder à la liste de demandes (1 indirection), au descripteur de demande concerné (n indirections), puis enfin au tampon de données (1 indirection) via les descripteurs de tampon de données (p indirections).

L'application de ces contraintes, compte tenu des caractéristiques des protocoles implémentés, passe par l'application, quand elle est possible, des règles suivantes:

. La seule partie d'une TPDU que le protocole doit étudier est l'en-tête. Il est donc primordial d'accélérer l'accès à cette en-tête. La zone données du descripteur de demande à été créée à cet effet, et ne doit contenir que cette en-tête. Tâche impossible à réaliser pour tous les types de TPDU dans TP4 puisque les en-têtes sont de format variables, mais qui est triviale avec XTP ou même TCP.

. Minimiser le nombre de tampons de données (de manière à réduire les tâches d'allocation et de restitution de ressources) et, dans tous les cas, faire coïncider les limites physiques des tampons sur la taille utile des DT TPDU issus de la segmentation; ceci de manière à minimiser le nombre de PnPDU pour décrire les données utilisateurs.

. Disposer de primitives systèmes d'allocation et de restitution de ressources performantes et adaptées au type d'application (on a seulement quatre types de ressources de taille constante).

. Mettre au point des mécanismes de communication par ressources partagées qui garantissent à la fois rapidité d'exécution, absence de transfert de données et exclusion mutuelle.

Ainsi devra-t-on attacher une importance toute particulière aux algorithmes de restitution de ces ressources et aux primitives systèmes qui assistent ces algorithmes.

De ces règles peuvent découler un certain nombre de modifications sur le modèle de structures de données, destinées à maximiser l'efficacité de l'implémentation pour un protocole particulier ou un environnement hôte différent. Ainsi, selon le protocole implémenté, on peut envisager

. De modifier la structure de descripteur de demande pour introduire, dans la zone utilisateur, l'en-tête et la queue.

. De faciliter les travaux de segmentation en prévoyant en début des tampons de données, une zone réservée à l'en-tête de la future TSDU; ou en créant une zone en fin de PnPDU pour y rajouter l'en-tête de DT TPDU construite à la segmentation.

. D'associer le descripteur de tampon de données au tampon de donnée de manière à gagner une indirection dans l'accès aux données et supprimer ainsi un type de ressource.

. On peut encore décrire directement, en mémoire hôte, les TSDU par PnPDU et supprimer l'existence des listes de demandes. Une telle modification est particulièrement intéressante puisque:

.le PnPDU reste le seul type de ressource utilisé (la gestion mémoire s'en trouve simplifiée);

.on n'a plus a transférer les données en réception.

1.2.2.2.2 Fonctionnalités d'un noyau système élémentaire

Nous ne voulons pas redéfinir complètement un système d'exploitation pour l'application implémentée, ni modifier un système existant afin de l'adapter à nos besoins (C'est la démarche entreprise par Protocol Engines qui propose avec l'implémentation en C de XTP un noyau système dédié adapté d'UNIX [Whaley 89]). Les nécessités en terme de système d'exploitation

d'une application de ce type sont loin d'un véritable système d'exploitation et n'en reprennent que les principes de base. Nous nous contenterons donc de définir un ensemble de procédures élémentaires qui permette d'adapter l'application aux propriétés de l'environnement hôte [Murray 88].

Pour les primitives intégrées au Transputer (gestion du parallélisme et des temporisations), nous étudierons comment les aménager pour les exploiter avec le maximum d'efficacité. Les primitives de gestion de ressources seront définies (dans le paragraphe suivant) pour simplifier et optimiser l'utilisation du modèle de structures de données dédiées. L'écriture de ce noyau système à fait l'objet d'une analyse théorique initiale ainsi que de l'étude des techniques d'implémentation existantes sur [Krakowiak 85], [Cornafion 81] et [Crocus 75].

Cependant, le type d'application sur lequel nous travaillons introduit des contraintes qui font que les algorithmes proposés dans la littérature étudiée ne peuvent être réutilisés directement. Justifions le choix de ces algorithmes pour la gestion mémoire et le traitement des temporisations.

Le gestionnaire mémoire

Suite à l'étude des méthodes existantes (abordé dans [Krakowiak 85], chapitre 9 et [Woodside 89]), et aux conclusions de [Cabrera], nous avons choisi d'analyser les caractéristiques des structures de données et de l'application pour en déduire un gestionnaire de mémoire efficace.

Le modèle de structures de données défini au paragraphe précédent fonctionne sur quatre types de ressources différentes, qui ont chacune une taille fixe dans toute l'application. Plutôt que de choisir un algorithme trop général, trop complexe pour ce problème, nous avons sélectionné une variante du buddy [Margolin 71]:

On divise la mémoire en n zones, chaque zone étant affectée à un type de descripteur particulier. Le gestionnaire possède alors n listes de descripteurs libres et se comporte comme n allocateurs indépendants (l'initialisation devra garantir que chaque sous-allocateur s'est vu affecter une zone mémoire disjointe). Les avantages d'une telle méthode sont nombreux:

.allocation et restitution sont rapides (voisines de 6μs) ;

.risques de conflit minimisés (on alloue en retirant en tête, on restitue en chaînant en queue);

.cette partition de la mémoire nous protège contre un blocage du système pour manque de ressources. On peut en effet subdiviser, à l'initialisation, une même zone en pages réservées pour des types d'utilisation différents;

.la technique est directement transportable sur un environnement parallèle où les divers allocateurs seraient alors réellement implantés sur des processeurs différents.

Il restait à résoudre le problème d'exclusion mutuelle pour l'accès à l'allocateur. La solution immédiate sur le Transputer consistait à affecter un canal virtuel à chaque processus susceptible d'utiliser l'allocateur et de multiplexer ensuite ces canaux sur un canal unique par lequel on accède enfin à l'allocateur. Or le ***multiplexage de n canaux coûte n * 1,5 µs*** (sans compter les n commutations introduites par chaque instruction de communication).

Alors que pour garantir l'exclusion mutuelle, il suffit de protéger le gestionnaire mémoire contre les risques de commutation de processus, soit de l'exécuter en priorité haute. Le ***passage en priorité haute coûte 3 µs*** seulement. Malheureusement, cette technique devient partiellement inefficace sur une architecture multi-processeurs. On pourra alors la remplacer par l'appel d'une instruction de Test&Set câblé sur l'architecture cible, et par une implantation astucieuse des processus sur les différents Transputers.

La gestion des temporisations

Forts de l'expérience acquise pour la conception de l'allocateur de ressources, nous avons voulu développer un contrôleur de temporisations ayant les mêmes caractéristiques: rapidité, sûreté, communications par canaux minimisées.

[Varghese 87] montre la nécessité d'un gestionnaire de temporisations efficace et propose à cet effet des solutions moins coûteuses. La présence sur le Transputer d'un noyau de gestion des temporisations câblé est, à cet effet, une base intéressante, mais mal adaptée aux besoins spécifiques d'un tel protocole.

Le nombre de temporisations à gérer par l'application est dynamique parce que dépendant du nombre de connexions ouvertes; une temporisation donnée a donc la durée de vie de la connexion à laquelle elle est attachée.

Les procédures utilisées sont classiques: start, stop ou reset. ***4 temporisations sont systématiquement créées à l'ouverture de chaque connexion*** . Il est donc primordial de

minimiser le temps d'armement et d'arrêt d'une temporisation. Le coût minimum qu'il est possible d'atteindre est d'une instruction de communication si l'on utilise la gestion intégrée du Transputer. Mais cette gestion n'est pas adaptée. L'événement qui a provoqué l'armement d'une temporisation et l'événement qui annonce l'arrivée à échéance de cette même temporisation sont totalement indépendants. Le protocole de Transport ne peut donc pas armer directement une temporisation et se bloquer en attente de l'échéance; ce comportement lui interdirait la possibilité de traiter tout autre événement. On doit donc nécessairement passer par l'intermédiaire d'un processus de gestion de temporisations indépendant du protocole.

De plus, armer une temporisation Occam à chaque procédure Start serait trop coûteux et la nécessité d'un processus intermédiaire qui gère lui même des "temporisations virtuelles" à partir d'un unique "timer Occam" est indispensable. Nous avons donc conçu un gestionnaire de temporisations qui associe, à chaque temporisation de TP4, un descripteur qui identifie sa durée, sa signification et la connexion concernée.
Armer une temporisation consiste à l'insérer dans une liste des temporisations actives et, éventuellement, à mettre à jour l'unique "timer Occam" utilisé. Afin de simplifier cette insertion, nous gérons une liste par type de temporisation de sorte que toutes les temporisations d'une même liste aient la même durée. L'insertion d'un descripteur dans la liste est ainsi effectuée en queue pour un coût réduit de 5 µs. Pour désarmer une temporisation, on la retire simplement de la liste et on met éventuellement à jour l'heure de réveil du "timer Occam"; tâche toute aussi rapide puisque c'est à partir du contexte de la connexion concernée que l'on accède au descripteur dans la liste. 5 µs sont nécessaires pour cette tâche.

On utilise donc un unique "timer Occam" par type de temporisation, remis régulièrement à jour à l'armement et au désarmement des temporisations (par l'intermédiaire des instruction PLUS ou MINUS qui permettent de prolonger ou réduire la durée d'une attente). C'est par ce même "timer" (ou canal) qu'est prévenu, en cas d'échéance, le protocole de Transport. Le principe d'exclusion mutuelle par passage en priorité haute est dans ces conditions conservé.

Si le principe retenu pour la gestion des temporisations est conçu pour être efficace dans un environnement à base de Transputers, il n'en reste pas moins coûteux par le temps qu'il consomme.

L'implantation sur unique Transputer de l'application ne nous permet pas de réduire encore ce temps de traitement. Par contre, sur une architecture parallèle, on peut envisager d'associer au Transputer qui traite le protocole un autre Transputer chargé uniquement de prendre en charge la gestion mémoire. Le Transputer annexe travaille directement sur de la mémoire propre. Il reçoit

du protocole les instructions d'armement et d'arrêt de temporisation par canal (deux octets suffisent pour ce faire) et rend les échéances par le même moyen.

Vu du côté du protocole, armer et désarmer une temporisation devient alors immédiat:

.une seule instruction est nécessaire
.les conditions définies pour une synchronisation rapide sont réunies.

Les primitives développées pour le gestionnaire mémoire et le gestionnaire de temporisations sont détaillées respectivement en annexes D et E.

1.2.2.3 Gestion du parallélisme

La gestion du parallélisme regroupe autour de la notion de processus un ensemble d'activités complexes, généralement coûteuses à implémenter par logiciel, et parmi lesquelles on retrouve les notions de communication, synchronisation ou encore exclusion mutuelle.

L'ouvrage de Sacha Krakowiak (déjà cité) explique en détail la problématique de ces mécanismes et présente, pour chacun d'eux, les techniques classiques d'implantation. Ayant choisi d'évoluer dans un environnement à base de Transputer, nous nous imposions d'utiliser, pour la gestion du parallélisme, les facilités offertes par le processeur.
Le Transputer intègre en effet des primitives de gestion du parallélisme câblées, à la mode "rendez-vous" [Hoare 79], selon le modèle CSP. Les processus utilisent pour se synchroniser, puis pour communiquer, un canal de communication sur lequel ils se bloquent pour attendre le rendez-vous. Par contre, le Transputer n'offre aucun moyen de faire de la communication non blocante par ressource commune. Mais un processus en attente ne paralyse pas le processeur.

Vu la problématique des protocoles de communication, qui consiste en particulier à minimiser les transferts de données, l'utilisation de ressources ou listes partagées reste le meilleur moyen de faire de la communication. D'autant plus que, la vitesse de communication sur lien (12 Méga-bits par seconde) et le mécanisme de gestion de la concurrence entre processus (voir [INMOS 88b]) font qu'il est préférable d'éviter de communiquer par canaux de communication.

La communication par canal n'est pas coûteuse pour le temps de commutation de processus (inférieur à la micro-seconde), mais pour les effets de bord qu'elle provoque sur l'application:

."hachage" du temps CPU affecté à chaque processus et réduction du rendement du Transputer
.étalement du temps d'exécution des processus
.introduction d'attentes supplémentaires lors des rendez-vous.

Mais le rendez-vous reste, sur le Transputer, le meilleur moyen de synchroniser deux processus dans les conditions suivantes:

. en environnement multi-processeurs, les deux processus communiquant sont seuls implémentés sur deux processeurs différents; sinon ils sont les deux seuls processus actifs au moment du rendez-vous

. le consommateur est déjà au rendez-vous quand le producteur exécute la communication; ceci évite au consommateur d'être commuté

. le producteur n'attend pas d'acquittement pour poursuivre le traitement; il redevient donc immédiatement actif.

Une synchronisation par ressource partagée uniquement est donc dangereuse sur le Transputer pour plusieurs raisons. Qui dit ressource partagée dit accès simultané sur cette ressource des processus autorisés et nécessité de garantir une exclusion mutuelle. Exclusion qui peut être obtenue au niveau de la mémoire, de la ressource, ou d'un descripteur (si cette ressource est une liste par exemple). ***L'attente sur une liste ou sur un descripteur partagé est blocante*** pour le processus qui l'exécute et donc pénalisante si d'autres tâches sont à effectuer par le Transputer concerné. Le blocage intervient à deux niveaux:

.impossibilité à un autre processus de traiter des données pendant au plus un time-slice (soit 1 ms sur le même processeur);

.selon le mode d'exclusion mutuelle, risque de blocage temporaire du processus qui range les descripteurs dans la liste avec tous les risques que cela peut comporter.

Nous avons donc décidé de synchroniser les processus de l'application par rendez-vous, puis de les faire communiquer par mémoire partagée. Ce qui nécessite de définir des algorithmes supplémentaires pour protéger ces ressources contres les modification simultanées.

1.2.3 Structuration de l'implémentation

La première partie de ce chapitre nous à permis de définir un environnement d'étude, et de définir sur cet environnement des techniques d'implémentation et autres contraintes destinées à atteindre des performances élevées. En fait, la démarche entreprise vise à adapter, avec le maximum de précision, l'application à son environnement hôte.

Nous allons donc, dans cette seconde partie, montrer comment, à partir de l'analyse de la structure du protocole, nous avons utilisé les techniques mise au point, et appliqué les contraintes, pour obtenir une application performante qui réponde en plus à des critères de

modularité, flexibilité, adaptabilité

modularité et adaptabilité se rejoignent pour permettre l'analyse de diverses configurations, et aussi de diverses implantations sur architectures cibles. La modularité nous assure en plus des simplifications à la mise au point, dans la lisibilité de l'application, et lors des mesures de performances de l'application.
La flexibilité enfin nous permettra de faire évoluer l'implémentation (par mises au point diverses sur les techniques d'implémentation) sans avoir à la reprogrammer complètement.

La structure en OCCAM de l'implémentation produite est proposée en Annexe C.

1.2.3.1 Structure de l'implémentation du protocole

L'importance de la phase d'implémentation avait été évoquée dans [Diot 88]. Une implémentation performante résulte d'un compromis entre les spécifications et services du protocole et les caractéristiques de l'environnement hôte.

Les moyens que nous avons définis pour atteindre des performances élevées tendent tous à réaliser une ***adaptation mutuelle*** entre l'environnement de développement et les contraintes de l'application. Et il faudra faire en sorte d'adapter les facilités offertes par l'environnement hôte pour accroître les performances de l'implémentation.

Les documents de base utilisés pour l'implémentation de TP4 sont les normes ISO 8072 et 8073 qui définissent avec précision (mais de manière informelle) toutes les fonctions et services que doit assurer le protocole.

L'analyse d'une représentation synthétique de l'ensemble des tâches que doit effectuer le protocole sur chaque événement entrant figure 1.2.8 nous conduit à une série d'observations qui nous guideront dans la conception de la structure de l'implémentation.

-l'***information de base*** traitée par le protocole est en réception une ***TSDU*** , ou une ***TPDU*** (mémorisée pour les besoins de l'implémentation dans une liste de descripteurs) dont il extrait un couple ***événement, connexion*** qui détermine enfin un ensemble d'***actions*** à effectuer, selon l'état courant de la connexion. L'échéance d'une temporisation est aussi considérée comme un événement et suit donc le même traitement.

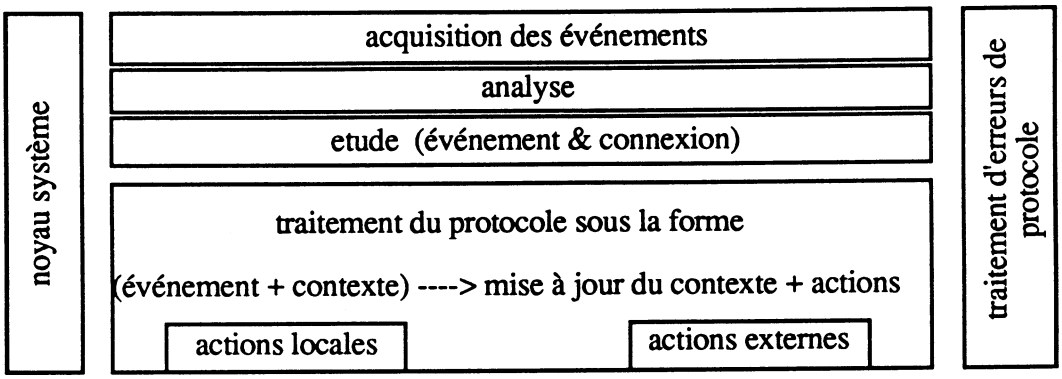


figure 1.2.8; structure du protocole de Transport OSI TP4

-Les ***actions*** peuvent être de deux type différents;

> ***.locales***, elles affectent exclusivement le contexte de la connexion traitée
> ***.globales***, quand elles provoquent l'émission de trames vers les couches adjacentes.

-le ***traitement d'un événement est systématiquement lié à une connexion*** .
Ainsi toute l'histoire d'une connexion (identifiée de manière unique par une **référence de connexion**) est retenue dans un **contexte de connexion** (dont le modèle est présenté figure 1.2.9); contexte mis à jour après traitement de tout événement survenant sur la connexion. Ce contexte qui, pour une connexion donnée, ne doit donc pas être modifié par plusieurs processus parallèles, ni même accessible depuis une autre connexion, a la durée de vie de la connexion qu'il décrit (la vie d'une connexion comporte les deux phases ouverte et gelée; ouverte durant le transfert de données et gelée afin de s'assurer après fermeture que les références associées ne soient pas réutilisées pour une autre connexion).

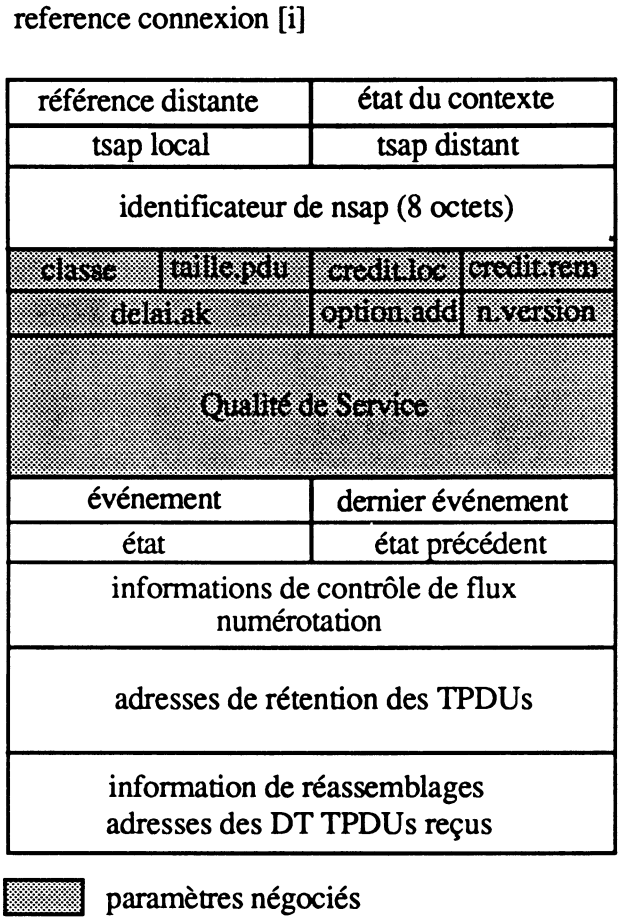


figure 1.2.9; contexte de connexion accessible à partir de la référence locale

-Traiter un événement entrant, pour un protocole, c'est lui faire subir une série de procédures, et ce en séquence; comme le met en évidence la structure verticale de la figure 1.2.8.

Cependant, certains parallélismes sont possibles dans un protocole, mais pas nécessairement opportuns:

. L'exécution en pipe-line des étapes de traitement d'un événement est possible; le débit de l'application sera dans ce cas le débit du plus lent des éléments du pipe-line.

. Si l'on dispose de moyens de synchronisation sur des ressources partagées, il est possible de paralléliser des parties du traitement d'un même événement. Par exemple, on peut paralléliser le protocole en deux processus, l'un traitant les événements reçus de l'utilisateur du service transport, l'autre les TPDU issus du réseau sous-jacent. Ce parallélisme est difficile à implémenter sur TP4; sa complexité rendant les synchronisations pour l'accès au contexte de connexion fréquentes et les attentes

actives nombreuses. XTP quant à lui définit deux machines d'états finis indépendantes pour le protocole en émission et réception. La parallélisation est dans ce cas sans danger.

. Il est enfin possible de traiter en parallèle plusieurs événements, une fois qu'ils ont été affectés à une connexion particulière (soit à partir de la tâche "traitement du protocole" sur la figure 1.2.8) et à condition que deux de ces événements n'affectent pas la même connexion.

L'implémentation que nous avons réalisée ne priviliégie aucun de ces parallélismes et les préserve tous. Le protocole de Transport ainsi implémenté comporte donc deux parties principales comme le montre la figure 1.2.10:

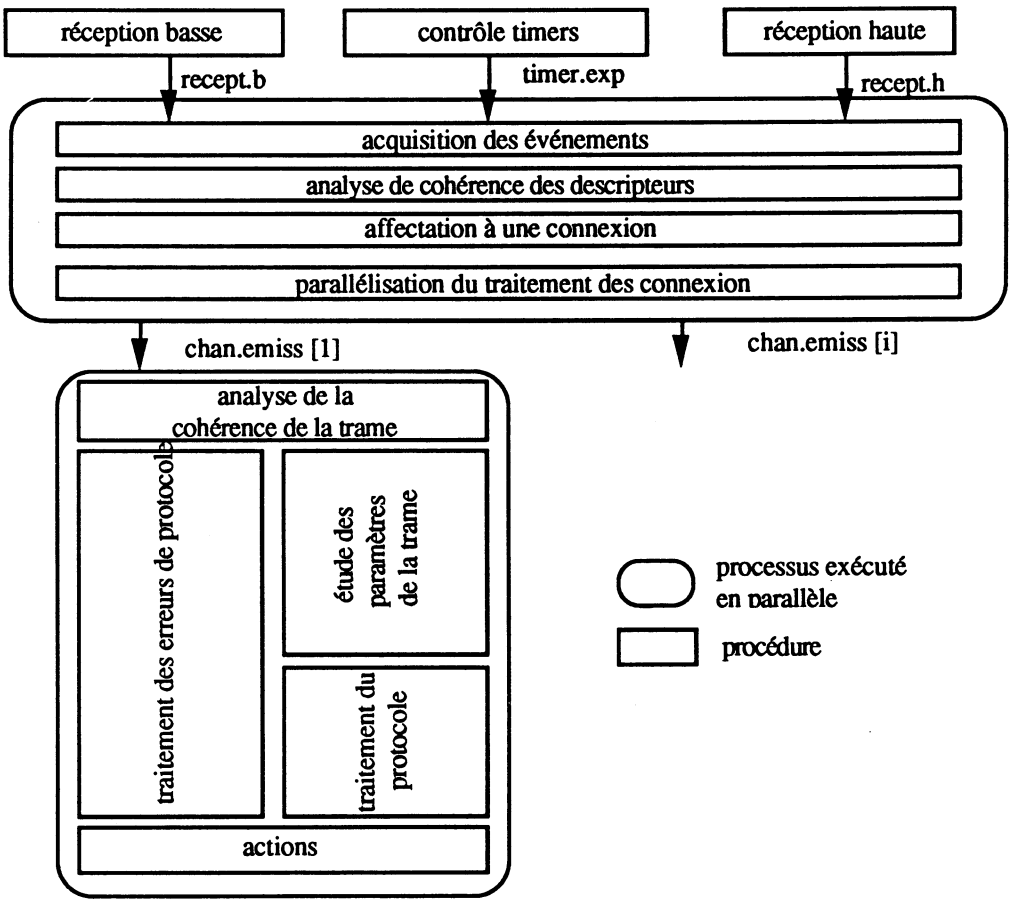


figure 1.2.10; structure de l'implémentation réalisée

Toutes les demandes entrantes sont traitées séquentiellement par le processus d'acquisition des événements. Cette sérialisation est nécessaire jusqu'à ce que le couple (événement, connexion)

ait put être déterminé; suite à quoi plusieurs événements affectant des connexions différentes pourront être traités parallèlement.

Des points de synchronisation (qui correspondent à des rendez-vous entre processus concurrents) garantissent l'indépendance des différents processus parallèles tout en leur permettant de communiquer: par ressources partagées:

. Entre les processus de réception et le protocole de Transport, on doit s'assurer de ne perdre aucune requête entrante, ni de bloquer un processus sur la synchronisation. La communication est alors réalisée par liste partagée.

. Entre la partie d'analyse d'un événement entrant et de parallélisation du traitement des connexions, même contraintes d'efficacité. On doit de plus avoir la certitude qu'une connexion ne puisse se voir affecter deux processus simultanément.

. Enfin dans les mêmes conditions qu'en réception, entre le protocole et les processus d'émission.

Ces points de synchronisation sont représentés sur la figure 1.2.10 par les flèches verticales.

1.2.3.2 Communication et synchronisation

Nous allons maintenant étudier les contraintes liées aux points de synchronisation (symbolisés figure 1.2.10 par des flèches verticales), et décrire pour chacun le mécanisme d'implémentation retenu. Nous ne nous intéresserons pas à tous les points de synchronisation de l'application. En effet, la synchronisation entre l'utilisateur du service Transport et le service est de moindre intérêt puisqu'il n'y a pas, entre ces deux processus, manipulation du modèle de structures de données dédiées.

Le problème du point de synchronisation interne au protocole est simple à traiter. On doit s'y assurer que deux événements associés à la même connexion ne pourront être exécutés en parallèle, affecter un processus à chaque événement traité, minimiser les attentes pour l'obtention des processus et maximiser le rendements des processus actifs.
La répartition du travail entre plusieurs processus identiques et indépendants impose:

. la présence d'un processus superviseur qui contrôle l'état de tous les processus de traitement de connexions;

. la gestion d'une table partagée pour contrôler l'état des processus;
. la mise au point d'un algorithme de répartition des événements entre les divers processus;
. enfin le multiplexage des processus qui voudraient, à un instant donné, insérer simultanément une séquence de PnPDU dans l'arborescence d'émission.

Par contre les points de synchronisations en réception, ainsi que la synchronisation entre deux couches adjacentes vont faire l'objet d'études précises. Les mécanismes retenus doivent être en mesure de:
-Minimiser les temps d'attentes aux points de rendez-vous (ou encore réduire le parcellement du temps CPU nécessaire à l'exécution de ce processus).

-Eliminer les goulots d'étranglement habituellement rencontrés aux points de rendez-vous entre les processus; goulots d'étranglement principalement dus aux actions d'allocation et de restitution des ressources, ou encore au transfert de données.

-Permettre un fonctionnement indépendant, sans interruption, des processus synchronisés.

Nous commencerons par étudier la communication entre le protocole de transport et les processus d'interfaces. En réception, le processus de traitement du protocole se retrouve en position de consommateur sur les informations entrantes alors qu'en émission il se retrouve dans le rôle du producteur. Dans tous les cas, il est prioritaire et son rendement doit être maximisé. A l'interface entre deux couches, les deux protocoles sont à la fois producteurs et consommateurs l'un de l'autre; la création de deux listes distinctes à l'interface ramène le problème étudié à deux producteurs- consommateurs classiques.

Notons qu'aux interfaces entre la couches Transport et son environnement, la notion de canal de communication coïncide avec celle de point d'accès. A chaque point d'accès correspond un canal et une liste de demandes.

Synchronisation en réception

C'est le problème de la restitution ultérieure des ressources qui à guidé la sélection des deux techniques implémentées:

Les ressources allouées par le processus de réception ne sont pas restituées par ce dernier. C'est le protocole de Transport (consommateur) qui les utilise et les gère après que le processus de

réception les ait préparées. Les deux processus travaillent donc sur deux listes indépendantes.

Le processus producteur (celui qui prépare les événements entrant au protocole) reçoit les informations au rythme de l'environnement externe, et les mémorise dans une liste de demande;s parallèlement, et ceci dès qu'il connaît l'adresse de la liste de demandes, il envoie vers le protocole de transport (consommateur) cette adresse et continue de chaîner les informations à la dite liste tant que le protocole ne s'est pas manifesté. Ainsi le protocole est attendu au rendez-vous par le processus de réception;

Tant que le protocole est occupé (la racine reste en attente sur le canal de synchronisation et le protocole n'est pas interrompu) le processus récepteur continue de chaîner des requêtes à cette même liste. Dès que le protocole est prêt, il prend en charge la nouvelle liste (et ce immédiatement après avoir fini le traitement de la précédente puisque la racine de cette nouvelle liste l'attend sur un canal). Le processus de réception alloue alors une nouvelle racine derrière laquelle il recommence la construction d'une nouvelle liste après avoir, comme précédemment, envoyé au protocole de Transport la nouvelle adresse racine par canal. Cet algorithme garantit au producteur de toujours pouvoir insérer ses demandes dans la liste et au consommateur un rendement maximum soit:

.consommer directement une demande dès son arrivée si le processus est inactif et s'il n'y a aucune demande en attente.

.être en mesure de continuer de traiter des demandes immédiatement après avoir fini le traitement de la demande précédente.

.minimiser le nombre de communications (soit le besoin en tâches systèmes) pour contrôler cette synchronisation.

.ne pas faire d'attente active, ni perdre de temps pour démarrer le traitement d'une sous-liste.

.préserver l'indépendance des deux processus communicant

L'unique difficulté concerne les ressources allouées par l'utilisateur du service Transport qui ne peuvent en aucun cas être restituées par le protocole de Transport. Le processus de réception n'étant pas plus autorisé à faire cette restitution, le problème pourra être résolu de deux manières:

. L'utilisateur recopie dans un tampon partagé les données à transmettre (comme nous le faisons en émission); il dispose donc d'une copie des requêtes et le protocole de Transport peut détruire les ressources devenues obsolètes.

. A chaque ressource en mémoire partagée est associée une variable d'état qui permet au protocole de rendre la dite ressource restituable.

Synchronisation en émission

Entre le protocole de Transport et le processus d'émission (ou de manière équivalente entre deux protocoles), les ressources partagées ne peuvent être restituées par le consommateur (à moins qu'il n'en reçoive l'ordre explicite du producteur). La décision de restituer une ressource est généralement prise par le processus qui l'a allouée, suite à la réception d'un événement sur la connexion concernée.

Vu les structures de données utilisées, l'émission est réalisée par construction d'une arborescence de PnPDU qui sera ultérieurement traitée par le processus consommateur (qui en extrait les informations nécessaires à l'exécution de sa tâche).

Nous avons donc été contraints de conserver une communication par liste complètement partagée, de sorte que la solution proposée en réception n'était pas utilisable.

Nous avons pu le constater lors des premières évaluations de performances, **la procédure de restitution de données décrites par PnPDU** est très coûteuse et peut **diviser par deux les performances de l'application** (suite à la réception d'un AK TPDU, le processus de restitution des ressources doit libérer les descripteurs de demandes et autres tampons de données alloués en réception, et ce quasi indépendamment de la fréquence des acquittements).

La restitution d'une suite de PnPDU nécessite en effet le parcours de la liste à restituer, l'accès aux ressources décrites, leur restitution et éventuellement la reconstruction du chaînage si l'on a restitué un élément en cours de liste. Cet algorithme est concurrent aux mécanismes de production et de consommation sur la liste partagée.

On cherchera donc, afin d'optimiser le rendement des processus communiquant, à ne pas faire de parcours de liste inutile (introduisant un délai dans la prise en compte des informations à émettre).

Nous avons donc envisagé deux algorithmes (dont on peut trouver encore quelques variantes) pour implémenter le partage de la liste d'émission et permettre à l'émetteur d'envoyer ou de restituer les ressources situées à un endroit quelconque de l'arborescence; ces trois implémentations excluant toute attente active chez chacun des deux processus en conflit sur la liste.

algorithme 1: le producteur protège le PnPDU qui précède la suite logique de PnPDU à traiter (ou qui précède l'adresse d'insertion); y effectue son traitement et signale par canal qu'une modification a eu lieu après avoir libéré le PnPDU préalablement protégé.
Le consommateur peut alors accéder à la liste, la parcourir et y effectuer les traitement détectés.

algorithme 2: le producteur suit le même comportement à ceci prêt qu'il envoie l'adresse de la ressource précédent celle qu'il vient de traiter dans la liste (rappelons qu'il n'existe pas de chaînage arrière sur les PnPDU). Le consommateur peut alors y accéder directement, sans parcours préalable de liste et la traiter sans délai.

Dans les deux cas, chaque processus qui consulte l'état d'un PnPDU doit être assuré qu'il ne sera pas interrompu par un autre processus avant d'en avoir modifié l'état; On a donc à simuler une instruction de type **"test&set"** pour éviter que deux processus ne prennent la main sur une même ressource. Encore une fois le **passage en priorité haute** nous sauve d'un problème bien délicat puisque le Transputer ne possède pas à son niveau le plus bas une telle instruction et qu'elle reste donc à implémenter sur la plate-forme de test au niveau du hardware (notons que le passage en priorité haute ne résout pas le problème si les processus en conflit sont implémentés sur des processeurs différents).

L'implémentation actuelle est réalisée selon l'algorithme 1 (avec prise en charge de la restitution par le producteur) mais nous comptons implémenter prochainement l'algorithme 2 afin d'en comparer les performances sur plate-forme de test et dans diverses configurations d'architectures (par exemple, émission et réception d'une même interface sur le même site; ou émission haute et basse ensemble).

Dernière remarque d'ordre général qui tend à justifier la communication par ressource partagée (telle que nous venons de la décrire) plutôt que par rendez-vous avec acquittement:

Une **liste partagée nous permet** d'effectuer directement la **rétention des TPDU dans la liste d'émission.**
On maintient ainsi une seule liste de PnPDU dont l'état de chacun permet aux processus qui

sont autorisés à y accéder de savoir quels traitements ils peuvent leurs faire subir. On évite alors les listes intermédiaires telles qu'elles existent dans l'implémentation actuelle pour la rétention des DT et ED TPDU. C'est d'ailleurs ce mode de rétention qui a été retenu en émission haute puisque toutes les primitives services à acheminer vers l'utilisateur du service transport sont directement insérées dans la liste d'émission, et non pas retenues au niveau des contextes de connexion.

1.2.3.3 Segmentation et réassemblage

Dans ce paragraphe nous allons aborder les problèmes d'implémentation liés aux fonctionnalités et services du protocole dont l'efficacité influence directement les performances de l'application (sans considérer les problèmes de parallélisation des processus qui déborderaient sur l'étude d'architectures ultérieurement prévue), et en particulier de la procédure de segmentation / réassemblage. Lors de l'implémentation de la fonction de segmentation / réassemblage, nous avons été confrontés à plusieurs choix d'algorithmes. Grâce à la structure chaînée du modèle de structures de données défini, la segmentation des TSDU était réalisée au fur et à mesure de l'étude de cette TSDU, sans introduire de parcours de liste, ni de délais supplémentaires.

Si la structure de liste était appréciable à la segmentation, elle s'avérait plus gênante au réassemblage puisqu'en effet, chaque DT TPDU est reçu indépendamment des autres, dans un ordre qui n'est pas nécessairement celui d'émission, et provoque systématiquement un traitement plus lourd qui peut se résumer en trois étapes:

(1) analyse de l'en-tête de TPDU et affectation à une connexion (non directement lié à la concaténation);
(2) rétention du DT TPDU reçu, dans l'ordre des numéros croissant et dans les limites de la fenêtre;
(3) recherche des conditions permettant l'émission vers l'utilisateur d'une TSDU complète, l'acquittement et la restitution des ressources.

Notre effort devra se porter sur les deuxième et troisième étapes puisqu'elles sont particulières à la segmentation. Elles nécessitent toutes deux un parcours de liste et multiplient donc le nombre des accès mémoire. Que de surcroît elles sont inutiles ou triviales quand les DT TPDU sont reçus en séquence.

Nous avons choisit d'analyser trois algorithmes d'implémentation fonctionnellement différents.

i) Pour commencer, la rétention aveugle des DT TPDU entrants par chaînage au précédent (et éventuellement au suivant). On effectue un premier parcours pour retrouver la place du TPDU entrant dans la liste au risque d'effectuer le travail pour rien s'il s'agit d'un TPDU dupliqué!; puis un second parcours pour détecter la présence d'un TSDU complet à acheminer vers l'utilisateur de la couche transport.

Notons que si les DT TPDU peuvent être acheminés dans un ordre quelconque, il n'en est pas de même des TSDU qui doivent être remis à l'utilisateur de la couche transport dans l'ordre dans lequel il ont été communiqués à l'entité éloignée.

Si cette solution reste améliorable par quelques finesses d'implémentation sur les parcours de liste, elle n'en reste pas moins lourde à implémenter et coûteuse en terme d'accès mémoire. Cependant elle ne nécessite, au sein du contexte, que la mémorisation de l'adresse début (et éventuellement fin) de la liste des segments retenus.

ii) De l'analyse des défauts de cet algorithme (le plus gros coût est imputable en effet aux parcours systématiques de la liste des TPDU déjà reçus) nous est venue l'idée de conserver deux mots de contrôle (initialement positionnés a zéro) dont chacun des bits représente un segment de la fenêtre de réception de données; le ième bit étant positionné à 1 pour signifier (dans le premier mot) la présence du DT TPDU numéro i dans la liste des segments reçus et (dans le deuxième mot), que le DT TPDU numéro i contient une indication de fin de TSDU.

Ainsi le repérage des TPDU manquant s'en retrouve simplifié (le parcours de liste est remplacé par un masquage de bit). Reste que l'accès dans la liste pour la rétention est inévitable.

On a donc pensé retenir, et ce de manière indépendante, l'adresse de chaque DT TPDU dans le contexte de la connexion. On s'épargne ainsi un parcours de liste que l'on remplace avantageusement par une affectation dans un tableau; et qui du même coup, rend le temps de traitement indépendant du nombre de DT TPDU déjà présents dans la fenêtre. La détection des indications de fin de TSDU se fait simplement par masquages successifs sur les deux mots de contrôle préalablement décrits.

L'inconvénient majeur de cet algorithme réside alors dans les mises à jours consécutives à l'émission d'un TSDU, des informations liées à la fenêtre qui nous oblige à décaler les DT TPDU encore retenus, ainsi que les deux mots d'états.

Malgré cet inconvénient, et après chiffrage, cette méthode reste beaucoup plus rapide (ce gain

pouvant aller jusqu'à 100 fois en ce qui concerne la recherche des TSDU à émettre); de même que le code écrit est trois fois plus faible pour cette seconde implémentation.

iii) Nous avons enfin envisagé un troisième algorithme basé non plus sur le parcours des segments reçus mais sur le parcours des segments de fenêtre vide (dits aussi "trous") . Ainsi un TROU est défini, dans la fenêtre de réception comme une suite de segments non reçus; chacun d'entre eux pouvant être décrit par le numéro de début (premier segment manquant), le numéro de fin (dernier numéro d'une suite de segments manquants) et l'adresse des TPDU (et plus exactement des PnPDU qui les décrivent) bornant le trou; chacun de ces descripteurs étant chaînés entre eux pour obtenir une liste de trous.

Cet algorithme nécessite la rétention des TPDU dans le contexte de connexion et l'utilisation des mots d'états de la seconde méthode, associés à une gestion de trous par chaînage. Il sera sans doute moins performant que le précédent à la rétention des DT TPDU puisqu'il aura de plus à mettre à jour la liste des trous, mais plus rapide à la recherche des TSDU à émettre (avec uniquement la consultation de la position du premier trou). Nous envisageons d'implémenter cette solution ultérieurement. Elle se montrera particulièrement adaptée à des protocoles tels qu'XTP qui propose un système de retransmission sélective des DT TPDU non reçus par contrôle des "trous" dans la fenêtre de réception.

1.2.3.4 TP4 et la gestion de ressources

Le problème de la gestion de ressources est complètement indépendant des algorithmes de gestion mémoire discutés au paragraphe 1.2.2.1. Il s'agit, dans ce paragraphe, de discuter comment les ressources, une fois allouées, doivent être gérées par le protocole. Analysons les besoins en ressources du protocole de Transport:

-la ***gestion statique*** provoque l'allocation des ressources nécessaires à une connexion dès son ouverture, et ce pour la durée de vie de la dite connexion (selon la taille de la fenêtre de contrôle de flux). Ainsi ***pas de tâche d'allocation ni de restitution au cours du traitement*** des DT TPDU, et garantie que la connexion ne manquera jamais de ressources tant qu'elle sera active. Les travaux de restitution seront effectués de manière automatique à la fermeture de la connexion. Mais ce type de gestion nécessite de pouvoir anticiper les besoins d'une connexion en ressource, et ce dès son ouverture.

-La ***gestion mémoire dynamique*** est quant à elle beaucoup plus souple à l'utilisation; à chaque réception d'une DT TPDU ou d'une TSDU, on alloue les ressources nécessaires, puis on restitue ces dernières suite à leur acquittement par l'entité éloignée.

Le choix de la stratégie doit être à la fois guidé par

. Le protocole; selon la taille de la fenêtre de contrôle de flux, la taille des DT TPDU, les caractéristiques de segmentation et de réassemblage, fonctionnement du contrôle de flux.
. Le domaine d'application; la gestion dynamique permettra une gestion plus souple pour des applications temps réel, et la gestion statique sera mieux adaptée au transfert de fichiers.
. La gestion mémoire, et la capacité mémoire; il est en effet exclus de faire de la gestion dynamique si l'allocation et la restitution sont très coûteuses, de même qu'il est hors de question de faire de la gestion dynamique avec une mémoire de taille réduite.
. Le modèle de structures de données.

Pour des questions de souplesse, de flexibilité de l'application (à la mise au point ou lors des évaluations), nous avons choisi de gérer les ressources dynamiquement.

Chapitre 2: EVALUATION DE PERFORMANCES ET PROPOSITION D'ARCHITECTURES

2 EVALUATION DE PERFORMANCES ET PROPOSITION D'ARCHITECTURES

2.1 Evaluation de Performances

2.1.1 Introduction à l'évaluation de performances et mesures globales

Les mesures de performances ont été menées en environnement mono-Transputer. Nous avons cherché à obtenir le plus grand nombre d'informations sur le comportement de l'environnement hôte et de l'implémentation. Les observations ont été menées en deux étapes pour nous permettre de:

. connaître les performances de l'application quand elle partage avec son environnement réseau un unique processeur, soit d'en établir les performances dans une implantation défavorable;
. étudier localement les différents services et fonctionnalités du protocole;
. analyser le comportement du modèle de structures de données;
. prédire le comportement de l'application en environnement multi-processeurs, de sorte à évaluer les performances de l'application dans un environnement parallèle;
. analyser le Transputer dans une application qui lui est typiquement destinée: un protocole de communication, pour mieux appréhender les problèmes de parallélisation et d'architectures cibles.

- La première étape nous donne une vision globale du comportement de l'application et une évaluation pessimiste des performances. Les résultats en seront étudiés dans cette introduction. Les mesures ont été effectuées entre deux entités distantes, hôte d'une carte mono-Transputer (voir figure 2.1.1), sur le temps de traitement complet de suites d'au plus 360 TSDU transportant de 1 octet à 8 kilo-octets de données utiles (la taille d'un DT TPDU ayant été fixée à 1024 octets par le profil CNMA), que l'on a répartis sur 255 connexions transport, dans les conditions suivantes:

.chacune des entités est implémentée, sur un unique Transputer;
.les deux entités sont reliées par un canal physique du Transputer
.Les cartes Transputer sont hôtes de deux micro-ordinateurs de type PC AT.

Le temps de traitement d'une suite de TSDU a été mesuré à partir de l'entité A du protocole de Transport, depuis la réception par le protocole de la première requête de Transfert de données issue de l'utilisateur de son service jusqu'à ce que toutes les DT TPDU transmises aient été

acquittées par l'entité distante (et les ressources associées restituées localement).
L'intervalle de temps mesuré comprend donc, outre le temps de traitement des TSDU par le protocole de Transport et son Service, les temps d'exécution de tous les autres processus implantés sur le même processeur et surtout le temps de réponse de l'entité distante.

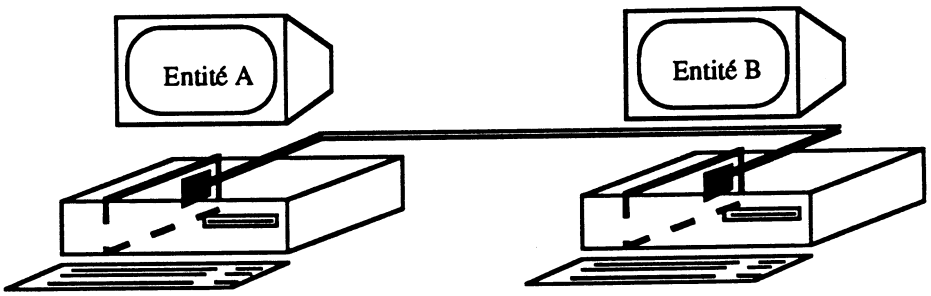


figure 2.1.1; environnement d'étude

-La seconde section de ce chapitre expliquera comment nous avons décortiqué et analysé le protocole de manière à évaluer précisément les performances de l'application selon le degré de parallélisation de l'environnement cible. Ces mesures nous permettront d'analyser plus précisément les caractéristiques et fonctionnalités du protocole; ou encore les techniques d'implémentation utilisées et autres structures de données dédiées.
La décomposition des mesures en deux phases nous permet de faire évoluer, dans un premier temps, l'application vers des conditions d'implémentation plus performantes (où l'on a cherché à rendre les performances de l'implémentation indépendantes des mécanismes de synchronisation cablés du Transputer), avant d'étudier le comportement précis du protocole et des structures de données dédiées.

2.1.1.1 Les mesures globales

Outre les conditions d'implantation (l'unique processeur est surchargé puisqu'il implante, en plus de l'application, un ensemble de processus indépendants, à exécuter parallèlement), les paramètres fonctionnels du protocole, ainsi que le format des structures de données, n'ont pas été choisis pour atteindre les performances les plus élevées.

Les temps et débits obtenus ne sont donc pas interprétables au sens de l'efficacité de l'implémentation, mais uniquement utilisables dans l'analyse d'un certain nombre de phénomènes (aussi bien liés au Transputer qu'au protocole lui même) que seule une étude sur site peut nous permettre d'observer. D'un autre côté, elle ne permet pas d'étudier séparément le comportement des diverses parties de l'application puisque l'exécution sur un unique Transputer confond (en partageant son temps CPU) l'exécution dite parallèle de tous les

processus actifs.
Malgré les conditions non optimales de test, nous avons extrait de ces mesures un ensemble de résultats généraux intéressants à plusieurs titres:

. étude du coût du parallélisme en environnement mono-Transputer et des conséquence de la charge en processus parallèles sur le débit de l'application;
. obtention d'une première évaluation des performances du protocole implémenté dans des conditions défavorables; nous pourrons comparer ces chiffres aux mesures trouvées dans la recherche bibliographique;

. analyse du comportement général de l'application, par rapport à son environnement (influence de la structure interne de l'application sur son débit, vérification des propriétés des structures de données dédiées) ou par rapport aux paramètres fonctionnels du protocole (taille de la fenêtre de contrôle de flux, taille des TSDU et taille des TPDU);

. évaluation des mécanismes de synchronisation; évaluation de l'influence du parallélisme intégré sur le comportement de l'application.

L'ensemble des courbes commentées traite, en deux parties distinctes, de l'influence du noyau système câblé du Transputer sur les performances de l'application, puis de l'analyse de résultats d'ordre généraux sur le comportement du protocole (pour utilisation ultérieure à des fins de comparaison avec les résultats des mesures atomiques).

2.1.1.2 Influence du Transputer sur les performances de l'application

La structure de l'application telle qu'analysée dans ce paragraphe reste celle présentée au chapitre précédent (paragraphe 1.2.3.2, structure de l'implémentation du protocole) et synthétisé figure 1.2.10. Nous allons étudier, en deux étapes:

. la parallélisation du traitement des événements sur n processus indépendants,
. les mécanismes de synchronisations et de comunication entre processus.

Coût du pseudo-parallélisme

Les premières courbes étudiées, (figure 2.1.2 qui établissent le temps de traitement d'une séquence de 120 TSDU de 1024 octets répartis sur 1 à 15 connexions, en fonction du nombre de processus traitant simultanément ces TSDU) nous montre combien le pseudo-parallélisme est coûteux, et surtout nous permet de ne pas retenir cet aspect du parallélismepour l'implantation

ultérieure du protocole.
La synchronisation entre le protocole de Transport et les processus d'interface est, pour les besoins de cette étude, réalisée par deux rendez-vous: un premier par lequel le processus producteur signale au consommateur qu'il a des informations à transférer, puis un second après transfert (par lequel le consommateur acquitte la réception des informations).

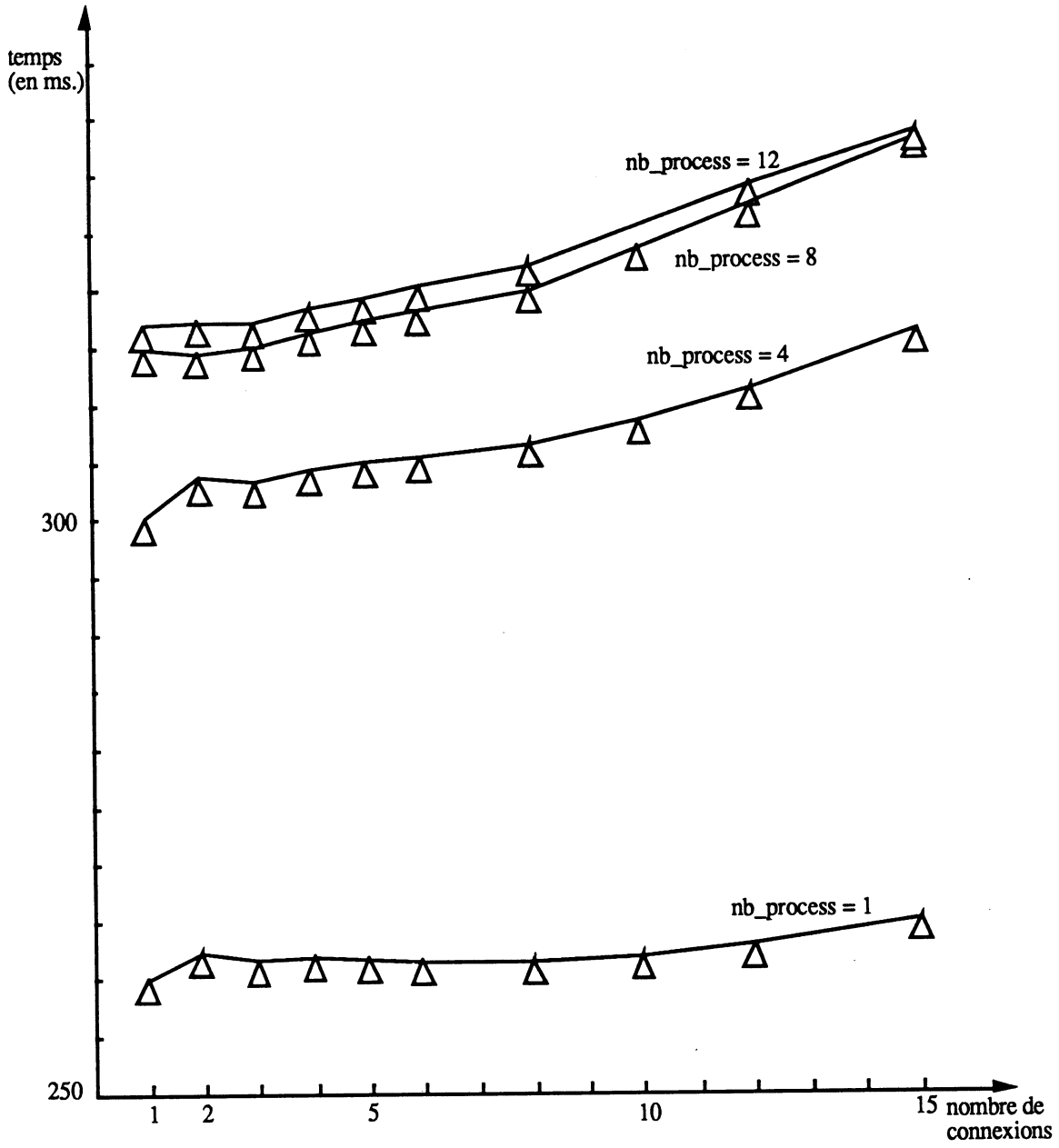


figure 2.1.2; étude du coût du pseudo-parallélisme sur la mesure du temps de traitement d'une séquence de 120 TSDU de 1 kilo-octets

On y observe deux comportements intéressants:

- La pente de chaque courbe est croissante alors qu'elles aurait dû être croissante jusqu'à ce que le nombre de processus parallèles soit égal au nombre de connexions actives, puis nulle ensuite puisque:

. le temps de traitement d'un événement est, par définition du protocole, indépendant de la connexion à laquelle il est affecté;

. tant que le nombre de connexions actives est inférieur au nombre de processus parallèles dédiés au traitement des événements, chaque connexion dispose de son propre processus; dans ces conditions, l'arrivée d'un événement sur une nouvelle connexion provoque l'activation d'un nouveau processus jusqu'alors inactif;

. dès que le nombre de connexions devient supérieur (au nombre de processus disponibles), tous les processus disponibles sont théoriquement actifs et quelque soit le nombre de connexions et d'événements par connexion, la charge de travail du processeur reste constante.

- L'introduction d'un processus parallèle supplémentaire est très coûteux, et ce malgré l'efficacité du Transputer dans les tâches de commutation de processus.

L'analyse de l'écart qui sépare ces trois courbes montre un accroissement de 17% du temps de traitement d'une TSDU si l'on passe de 1 à 4 processus parallèles, accroissement qui suit une progression logarithmique (cet accroissement n'est plus que de 5% entre 4 et 8 processus et devient inférieur à 1% au passage à 12). La figure 2.1.3 met en évidence cette progression en présentant, sous un aspect différent, les résultats de la figure 2.1.2. On y observe de surcroît que la pente des courbes augmente de manière logarithmique avec le nombre de processus actifs. Tous ces comportements montrent de manière irréfutable que:

-Le pseudo-parallélisme coûte cher; on a donc intérêt à minimiser le nombre de processus actifs simultanément sur un Transputer, quelles que soient leurs fonctionnalités.

-Introduire du parallélisme pour traiter les événements sur leur connexion n'est pas intéressant. Il impose:

. un processeur par processus pour rester dans des conditions de performances élevées;
. de gérer la répartition *sûre* des événements sur les processus disponibles;
. de multiplexer ensuite ces processus en émission sur les arborescences de PnPDU, en garantissant l'exclusion mutuelle.

-Du point de vue du coût, disposer d'autant de processeurs parallèles que de connexions que le protocole peut théoriquement ouvrir (soit 65535 connexions avec des références codées sur deux octets) n'est pas crédible.

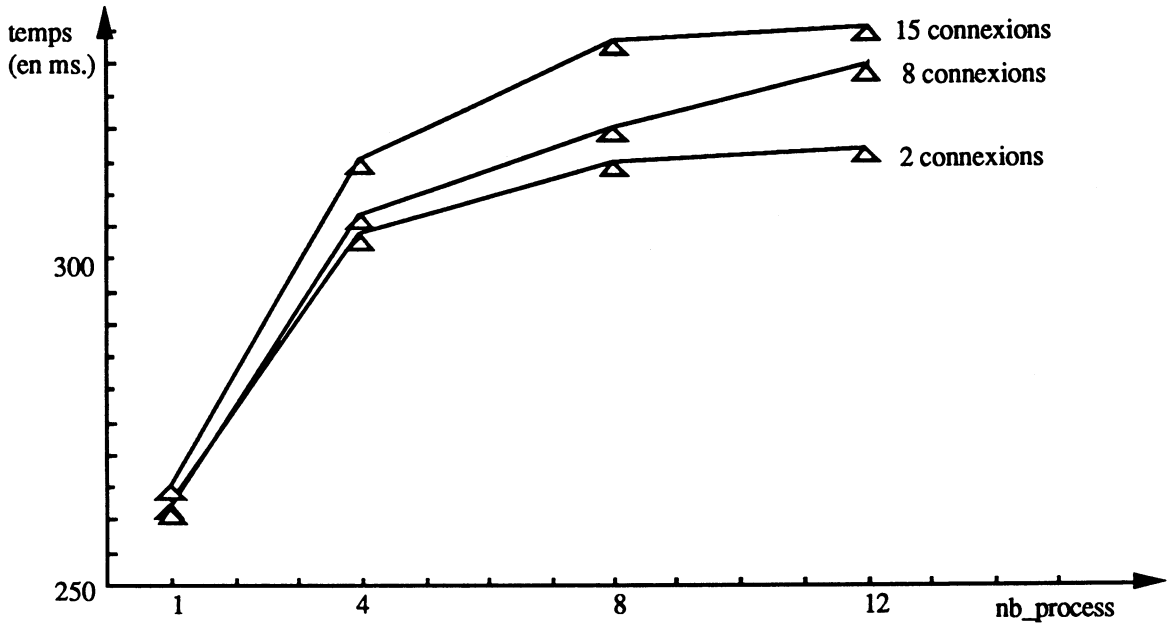


figure 2.1.3; temps de traitement d'une séquence de 120 TSDU de 1 kilo-octets

Après avoir éliminé le parallélisme entre les connexions (et montré qu'un processus unique était à la fois la solution la plus économique et la plus performante), nous allons montrer l'opportunité du parallélisme de type "pipe-line" à partir des courbes de la figure 2.1.4. On y compare le temps de traitement, sur un unique processus, de 120 TSDU de données:

. lorsque ce processus est lancé par canal (le processus d'étude de l'événement entrant et le processus de traitement sur connexion fonctionnent alors en pipe-line),
. lorsqu'il est simplement appelé comme une procédure du processus d'affectation à une connexion (le traitement du protocole est alors complètement séquentiel).

Le gain substantiel réalisé (de valeur moyenne 5%). Ces deux implantations étant sensiblement équivalentes en environnement mono-processeur, le gain observé sur deux processeurs sera très important (jusqu'à 100 % si les deux processus ont un temps d'exécution égal).

Optimisation du mécanisme de synchronisation des processus

L'aspect des courbes de la figure 2.1.4 n'étant toujours pas linéaire et horizontal, nous avons

modifié les conditions de synchronisation aux interfaces pour supprimer un rendez-vous et se rapprocher ainsi des conditions de synchronisation à un unique rendez-vous définies dans le paragraphe 1.2.3.2, communication et synchronisation.

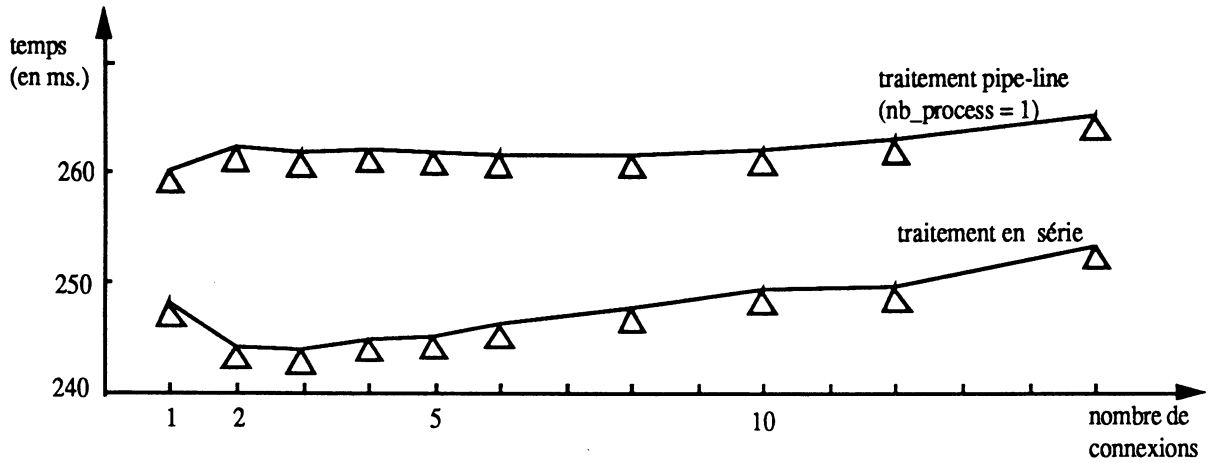


figure 2.1.4; évaluation du coût du parallélisme de type "pipe-line" à partir de la mesure du temps de traitement d'une séquence de 120 TSDU de 1 kilo-octets

Sur la figure 2.1.5, on observe le débit de l'application dans les conditions de la figure précédente (les processus sont alors traités séquentiellement):

. lorsque la synchronisation avec les processus d'interface est réalisée par un double rendez-vous;
. lorsqu'en ces mêmes point de synchronisation, le mécanisme retenu permet de ne pas interrompre le protocole de Transport, et n'utilise alors qu'un unique rendez-vous.

Le débit observé est visiblement indépendant du nombre de connexions actives. Sur 180 TSDU de 1024 octets, la courbe est quasiment plane. Ce qui tend à montrer l'efficacité de règles établies sur les conditions d'une synchronisation rapide entre deux processus, et surtout confirme que la communication par mémoires partagées, associée au mécanisme de rendez-vous pour la synchronisation, permet de rendre le débit de l'application indépendant des vicissitudes du noyau système câblé, et des effets de bords de la gestion de la concurrence entre processus parallèles (sur les instructions de rendez-vous).

Depuis le début de ce paragraphe, nous avons donc réduit l'influence du noyau système intégré du Transputer sur les performances de l'application en minimisant le nombre de processus simultanément actifs, puis en utilisant un mécanisme de synchronisation qui augmente le rendement utile du Transputer.

Tentative d'évaluation du rendement du Transputer

Pour terminer ces mesures sur le comportement du Transputer, nous avons essayé de contourner l'impossibilité technique de mesurer le taux d'occupation du Transputer (ou encore son efficacité) par la mesure du débit de saturation de l'application en environnement mono-processeur.

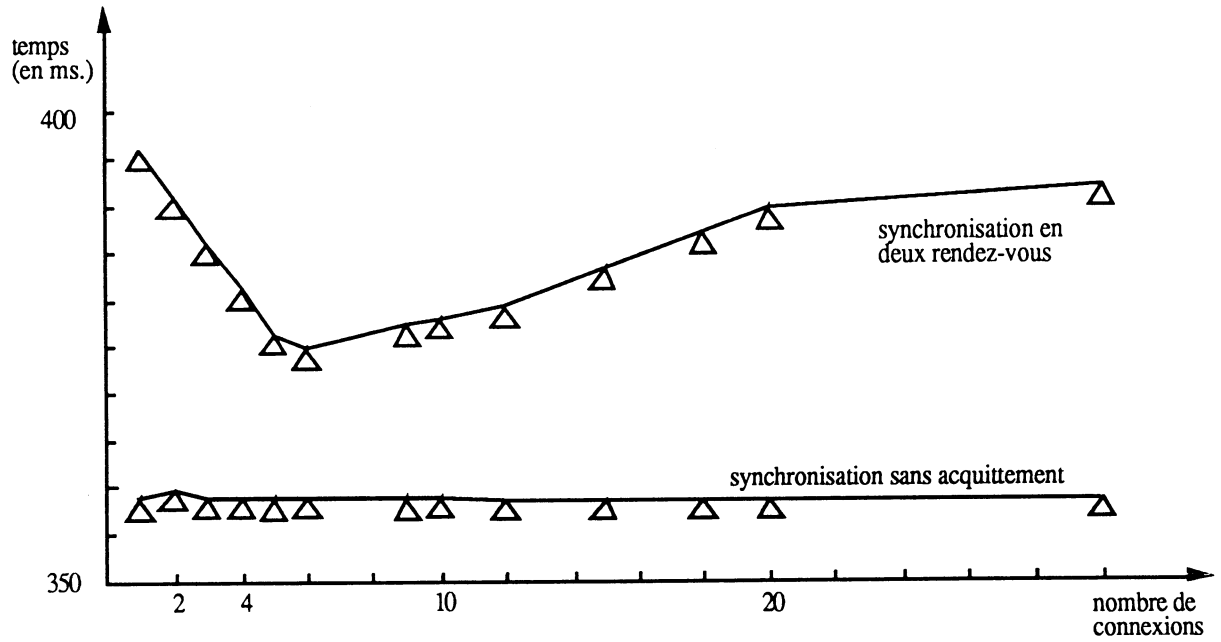


figure 2.1.5; effet du mécanisme de synchronisation aux interfaces sur le débit de l'application évalué sur le temps de traitement de 180 TSDU de 1024 octets

Nous avons mesuré, figure 2.1.6, le débit de l'application selon la taille des séquences de TSDU qui lui étaient soumises (dans chaque séquence, la taille des TSDU est constante):

-Pour des TSDU de 1 kilo-octets, le plafond est atteint aux environs de 600 TSDU par seconde; ce plafond est de 310 TSDU par seconde pour des TSDU de 2 kilos et enfin de 155 TSDU par seconde pour des TSDU de 4 kilos. Ce qui laisse apparaître une progression linéaire dans les débits de saturation en fonction de la taille des TSDU, ou un débit constant en terme de DT TPDU voisin de 600.

-Le débit plafond est atteint, selon la taille des TSDU, pour des séries de 40 TSDU (pour des TSDU de 1 kilo-octets), 20 TSDU (pour des TSDU de 2 kilo-octets) et 10 TSDUs environ (pour des TSDU de 4 kilo-octets); ou pour un débit moyen de 40 DT TPDUs quelque soit la taille des TSDU.

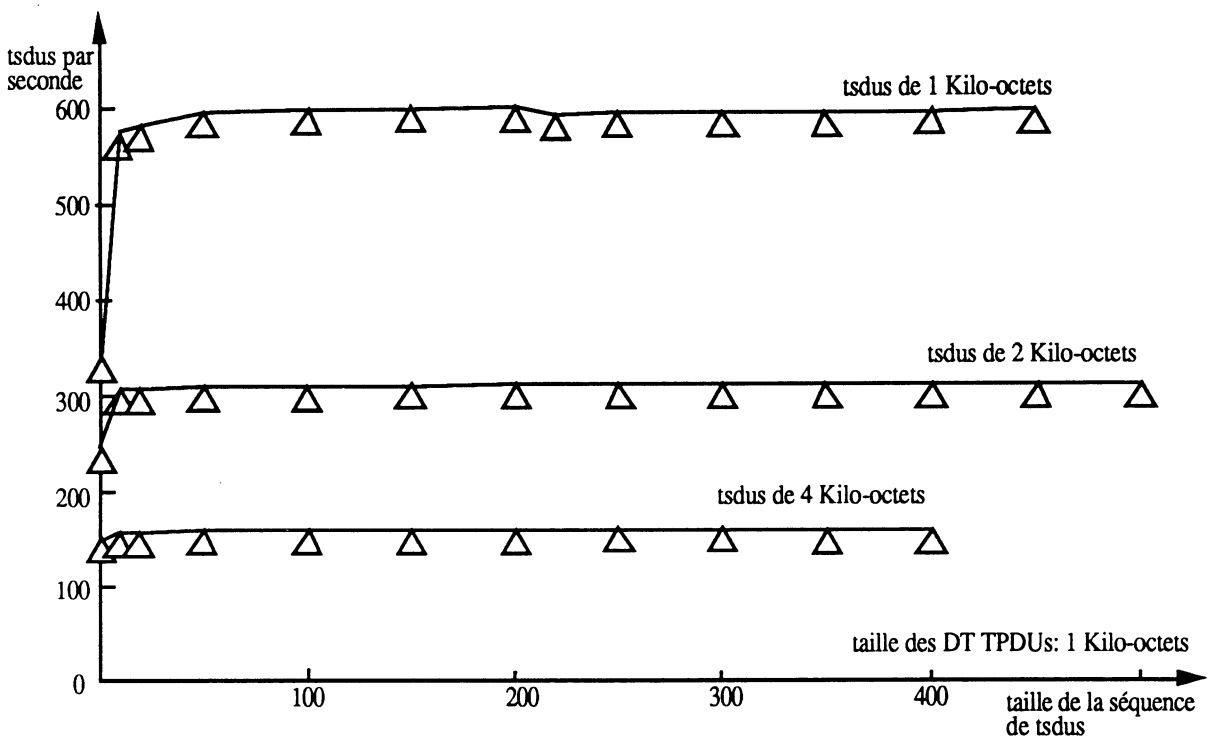


figure 2.1.6; évaluation du débit de saturation sur un Transputer unique par envoi au protocole des séquences de n TSDU de même taille

Ces résultats tendent à prouver que la taille des données transférées est le facteur prédominant dans le débit de saturation du Transputer (le débit de saturation moyen peut être estimé à 5 Méga-bits par seconde). Le temps consacré au protocole est, dans cette implantation, négligeable devant les transferts de données. D'où l'intérêt de chercher à minimiser ces transferts en utilisant des structures de données spécialisées.

On est donc encore bien loin des 100 Mbps à atteindre pour s'approcher de la catégorie des protocoles temps réels à haute performances (et surtout des 86 Mbps atteints actuellement par la famille de circuits FDDI commercialisés par AMD). Mais que ces résultats n'ont rien d'inquiétant ni d'alarmant puisque:

-Nous avons montré que l'environnement mono-Transputer ne permettait pas d'atteindre de très haut débits (il lui reste à gérer 12 canaux internes, 2 canaux physiques et 12 processus parallèles).

-Les conditions de test ne sont pas optimales; outre l'application elle même qui regroupe protocole de transport, noyau réseau et processus d'entrée-sortie, des processus d'émulation de l'application et de gestion de l'écran sont encore hôte du processeur. Le débit est surtout contraint par la vitesse de la transmission sur lien.

-Pour des TSDU de 1 kilo-octets non segmentés nous atteignons un débit voisin de 600 TSDU par seconde, chiffre tout à fait honorable comparé aux débits atteints par des implémentations classiques du protocole de Transport classe 4 (qui eux n'intègrent pas dans les mesures les processus d'émulation de l'utilisateur et du noyau réseau).

2.1.1.3 Conséquences sur les performances de l'application

L'étape précédente nous ayant permis d'étudier plus particulièrement le comportement du Transputer, nous allons analyser le comportement du protocole implémenté. Les conditions de mesure sont toujours celles définies en début de chapitre, à savoir sur la base de séquences de 360 TSDU de taille désormais variable. Les caractéristiques fonctionnelles du protocole sont:

. l'acquittement des DT TPDU reçues est réalisé par l'entité destinataire pour chaque TSDU complet remis à l'utilisateur distant;
. la taille des DT TPDU est de 1024 octets, soit 1019 octets utiles;
. les descripteurs de demandes peuvent contenir les 6 premiers octets d'un TSDU (soit la totalité de l'entête des TSDU de données);
. la taille utile d'un tampon de données est de 1019 octets (soit la taille d'un segment);
. la fenêtre de contrôle de flux du côté émetteur et récepteur a une largeur de 8.

Performances de bout en bout, en environnement mono-Transputer

La figure 2.1.7 établit, dans les conditions ci-dessus citées, le temps de traitement d'un TSDU en milli- secondes en fonction de la taille utile de ce dernier:

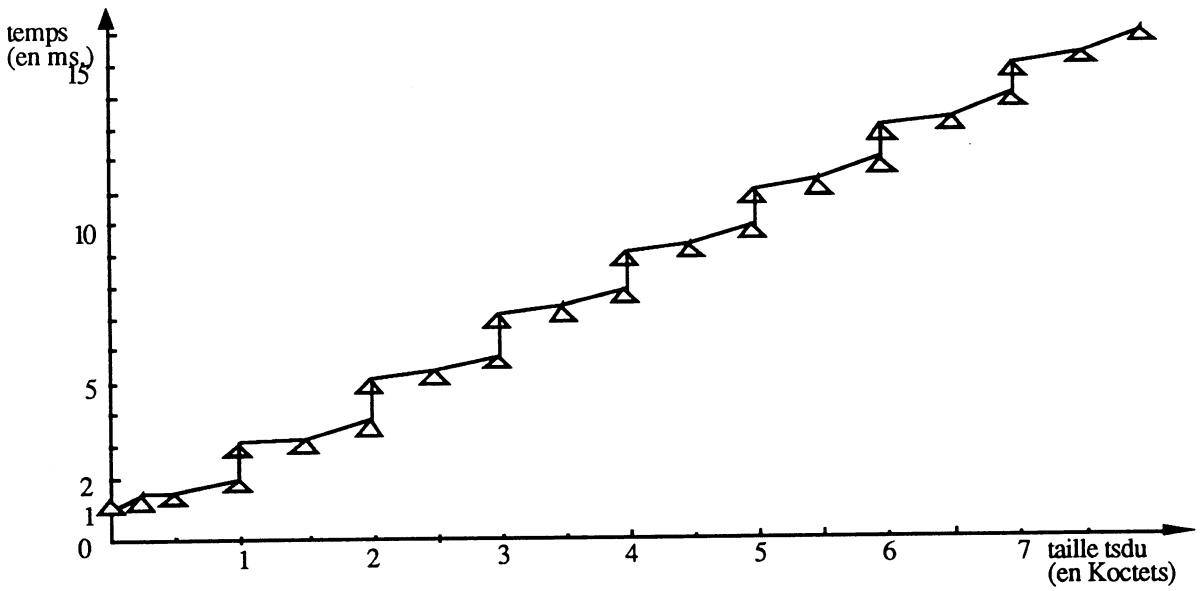


figure 2.1.7; temps de traitement d'une TSDU

-A chaque multiple de 1019 octets utiles, le service doit affecter à la TSDU entrante un nouveau tampon de données et initialiser ce dernier; le protocole quant à lui doit, tous les 1019 octets, segmenter la TSDU entrante. Ces deux activités justifient la cassure de hauteur constante dans la courbe.

-Le traitement effectué par le protocole est indépendant de la longueur des TSDU traitées (cette propriété est garantie par les structures de données dédiées utilisées). Chaque morceau devrait théoriquement être de pente nulle. Cependant, les processus d'interface doivent, pour transférer les TSDU entrantes et les TPDU sortantes, les recopier partiellement. Il est donc normal que la pente des morceaux de droite soit plus ou moins linéaire et surtout croissante.

Les figure 2.1.8 et 2.1.9 proposent les mêmes résultats sous deux aspects différents:

. débit de l'application en TSDU par seconde figure 2.1.8
. débit de l'application en Méga-bits par seconde figure 2.1.9

Ces deux courbes mettent en évidence un comportement invisible sur la précédente: la tendance asymptotique pour des TSDU de très grande taille. La valeur de la limite correspondant à un minima pour le débit en terme de TSDU par seconde et à une valeur moyenne pour le débit en Méga-bits par secondes (ou Mbps dans la terminologie anglo-saxonne).

Notons que la valeur de l'asymptote sur la figure 2.1.8 (théoriquement située en zéro), ne peut être atteinte puisque le cas échéant, le débit en Mbps reprendrait sa croissance (le nombre de TSDU par seconde devenant constant et la taille de ces mêmes TSDU continuant d'augmenter, le débit en Méga-bits par seconde augmente lui aussi). Sur la figure 2.1.9, le débit moyen atteint pour des TSDU de très grande taille se situe vers 4 Mbps.

L'interprétation de ces tendances asymptotiques est un exercice délicat; il est logique que le débit soit borné puisqu'une fois la capacité de traitement maximum du Transputer atteinte, les performances ne progressent plus, et le débit devient constant. Le facteur limitatif est bien entendu différent selon le processus considéré:

-les processus dont le débit est indépendant de la taille des informations traitées sera limité en terme de TSDU par seconde; ce qui signifie que le débit en TPDU par seconde du protocole de Transport va finir par se stabiliser (et le débit en terme de TSDU par seconde diminuer).

-les processus qui effectuent des transferts de données tels le processus d'émission seront quant à eux directement limités par le débit des transferts de données.

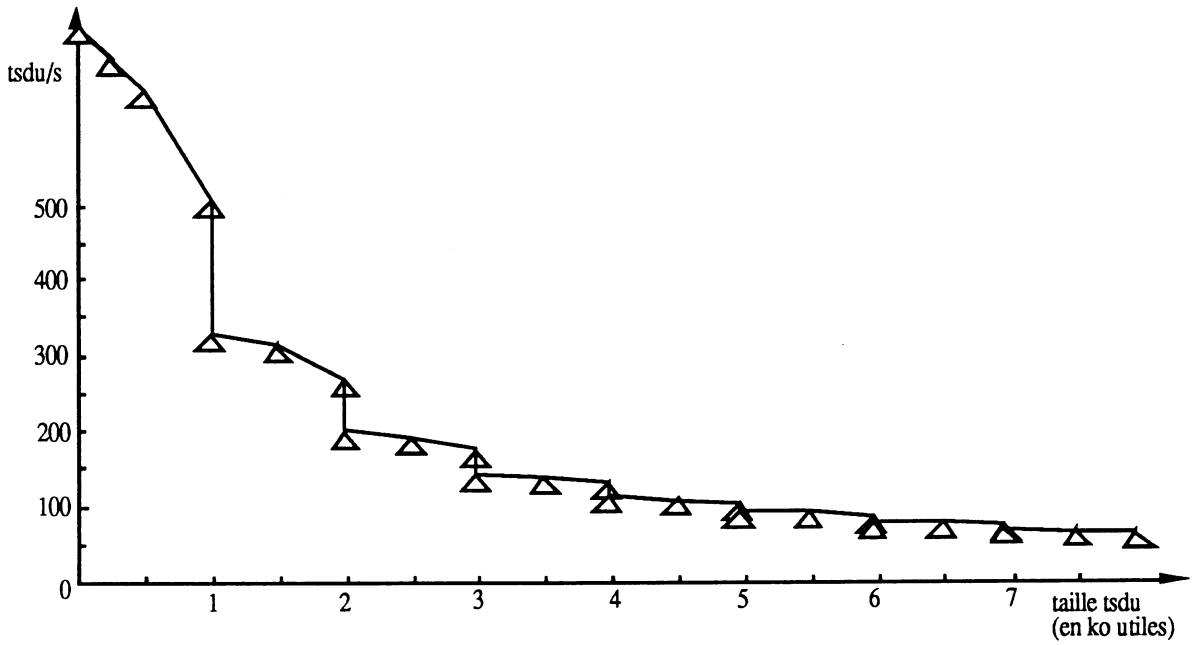


figure 2.1.8; débit de l'application en TSDU par seconde

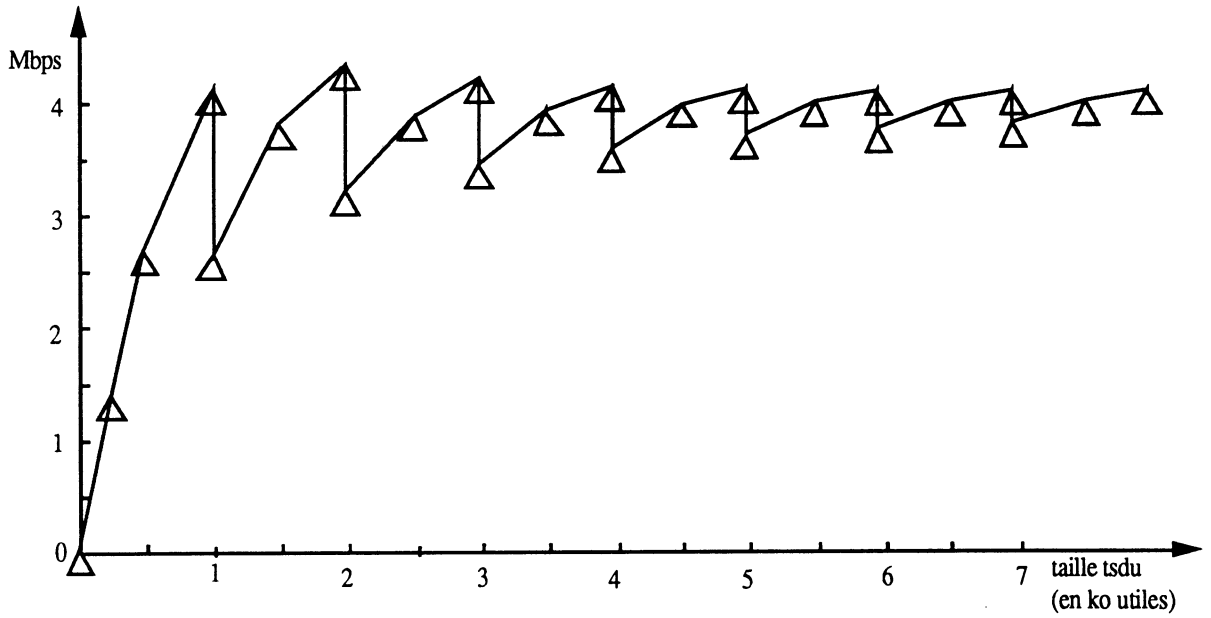


figure 2.1.9; débit de l'application en Méga-bits par seconde

Elimination des transfert de données superflus

Le débit de l'application est quoi qu'il arrive limité:

.par la vitesse des liens physiques du Transputer (liens utilisés comme médium de comunication entre deux entités Transport, et éventuellement aux interfaces)

.par la vitesse de transfert des données en mémoires.

Après avoir supprimé, au sein d'une entité Transport, tous les transferts de données par lien série, nous nous sommes penché, dans cette dernière étape, sur les moyens de minimiser encore le nombre de transferts de données.

En effet, le transfert des primitives services et des données inutiles qui y sont associées peut être supprimé; à condition bien sûr que le protocole de Transport et l'utilisateur du Service disposent d'une mémoire partagée.

La solution retenue pour supprimer ce transfert consiste à décrire directement les données utiles dans la mémoire de l'utilisateur du service, et de communiquer le PnPDU construit au protocole de transport. On s'affranchit ainsi des descripteurs de demande et autres tampons de données. L'acquisition d'une TSDU de longueur quelconque devient constante (mesurée à 45 µs). La tâche de l'application est ainsi simplifiée puisque

-Elle n'a plus à traiter qu'un seul type de ressource: le PnPDU; ce qui simplifie considérablement la gestion de la mémoire.

-On évite un transfert des informations utiles; les informations utiles ne sont plus recopiées qu'une seule fois lors de leur émission.

-La charge en terme d'allocation et de restitution se trouve considérablement réduite; une TSDU entrante découpée en n segments nécessite en réception par liste de demande l'allocation de 2 à n + 1 ressources et la restitution en fin de traitement de ces ressources. La solution par liste de PnPDU requiert l'allocation d'un unique PnPDU et sa restitution (et éventuellement la restitution de la zone décrite en mémoire partagée).

Sur la figure 2.1.10, on compare le débit de l'application en Méga-bits par seconde:

. tel que le montre la figure 2.1.9. Les TSDU sont alors transférées dans des listes de demandes à base de descripteur de demande avant d'être traitées par le protocole;

. lorsque les TSDU entrantes sont directement décrites (par l'intermédiaire d'un PNPDU) dans la mémoire de l'utilisateur du service transport.

L'acquisition des TSDU sans transfert de données provoque un regroupement des optimum locaux, sans provoquer, pour des TSDU inférieurs à 5 kilo-octets, d'augmentation des maximum. Au dessus de cette taille, la tendance s'inverse radicalement. Le débit, qui avait tendance à tendre vers une valeur moyenne un peu plus élevée qu'en réception par liste de

demande (4,2 Mbps sans transfert pour 4 avec), se met à nouveau à diverger et les maximum locaux à croître.

Ce comportement aurait dû être observé pour des TSDU de taille plus réduits mais encore une fois les conditions d'études en environnement mono-Transputer masquent le comportement réel du protocole de Transport tel que nous l'avons implémenté.

Le gain théorique aurait ainsi dû être de 40 %, soit une valeur maximum voisine de 7 Méga-bits par seconde dans la première partie de la courbe.

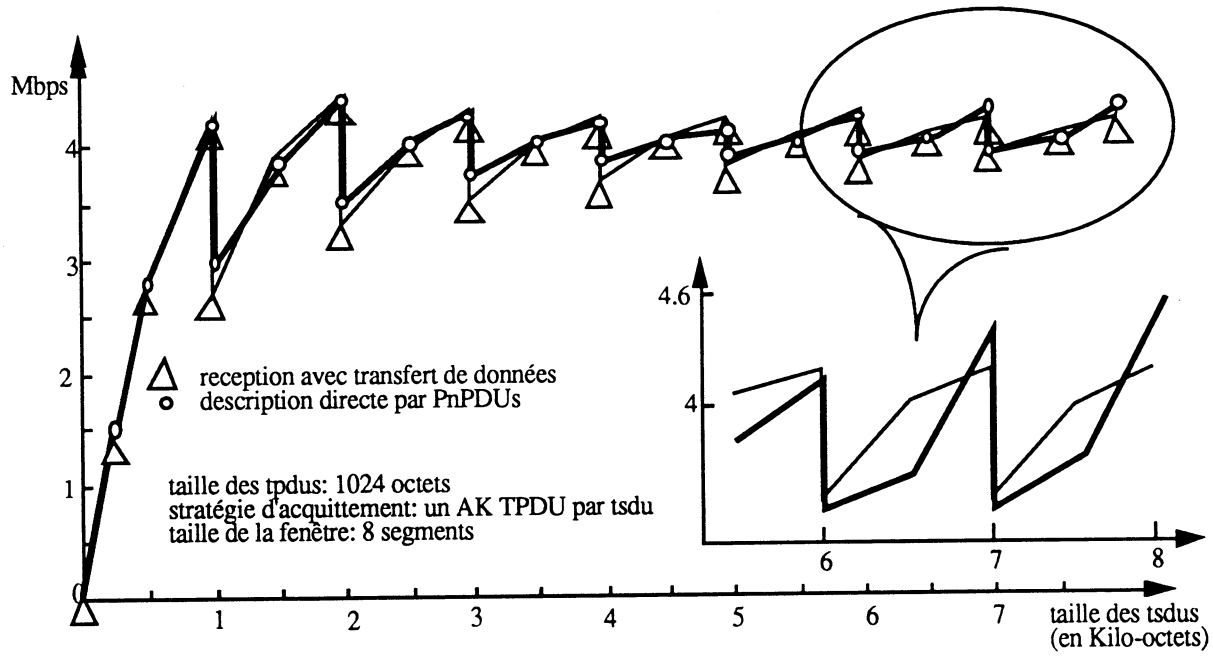


figure 2.1.10; débit comparé de l'application quand la réception est faite par descripteur de demande et quand les TSDU entrantes sont décrites en mémoire partagée par PnPDU

2.1.2 Mesures atomiques

Cette phase de mesures vient compléter la précédente pour nous permettre de *prévoir le comportement attendu de l'application en environnement multi-processeurs:*

.analyse des caractéristiques techniques du protocole, ou encore étude des diverses fonctionnalités implémentées et identification des paramètres fondamentaux de l'application;

.analyse et évaluation des structures de données dédiées et autres algorithmes de restitution de ressources;

.étude des divers aspects du parallélisme.

Dans cette phase de mesures, chaque processus, procédure ou suite d'instructions ayant une cohérence logique (d'où le terme "atomique") va être décortiqué, isolé, mesuré puis analysé de sorte à être capable d'en synthétiser le comportement sous forme d'une équation. En associant les équations établies, nous construirons un ensemble d'équations générales qui représentent le traitement d'une TSDU, d'une TPDU, d'un acquittement ou de toute autre activité en fonction des paramètres spécifiques de l'implémentation étudiée (qui sont différents des paramètres classiques, à cause des techniques d'implémentation utilisées).

Tous les résultats seront présentés et analysés séparément; on tentera cependant de faire ressortir une cohérence logique dans l'étude des équations. La progression logique adoptée *fait progresser l'application vers ses conditions optimales de fonctionnement* (par actions successives sur le modèle des structures de données, sur le choix des paramètres fonctionnels ou sur le choix des algorithmes spécifiques).

Les mesures vont être effectuées sur une seule entité (soit sur un unique Transputer, hôte d'un unique PC); chaque séquence d'instructions étudiée étant exécutée en priorité haute de telle sorte que:

. La séquence évaluée ne soit pas désactivée par le Transputer pendant son déroulement. De manière à mesurer le temps minimum d'exécution de la séquence.
. Les mesures soit effectuées avec une précision (maximale) de 1 µs.

Les équations établies prennent en compte tous les intervalles mesurables par la méthode utilisée à savoir:

. Le temps passé à réaliser la synchronisation entre deux processus n'est compté que pour le temps qu'il représente en terme d'instructions exécutées (ceci ne concerne que les processus d'émission et de réception); de même que les communications par canaux et autres multiplexages de canaux ne sont comptés que pour leur coût en terme d'instructions. Il nous est impossible d'évaluer les attentes introduites par le noyau système intégré du Transputer dans le temps d'exécution de chaque processus. C'est l'implantation qui devra garantir que les attentes introduites sont minimales.

. Le système est considéré comme fluide, c'est à dire qu'il n'y a aucun problème d'accumulation dans les files d'attente, que chaque événement est pris en compte dès son avènement; ce qui serait le cas dans une architecture parallèle efficace.

. Tous les appels de procédures sont inclus dans le temps d'exécution de la procédure (notons que tous les passages de paramètres se font par valeur; dans ces conditions, certains appels de procédures peuvent encore prendre jusqu'à 10 µs).

Par contre, la quantification des accès mémoires, des conflits, des commutations de processus et aussi les calculs de taux d'occupation du Transputer restent impossibles dans cet environnement mono-Transputer. Nous n'y renonçons pas pour autant; la plate-forme d'évaluation développée ultérieurement disposera d'un module d'espionnage qui nous permettra, pour chaque architecture, de collecter ces informations.

Nous ne nous intéresserons qu'aux processus qui font partie de l'application; soit les processus d'interface, et le protocole de transport. Nous analyserons l'application en trois étapes. Nous commencerons par étudier les règles établies pour l'utilisation des structures de données; pour ensuite, dans les conditions optimales définies, étudier le comportement des processus d'interface et enfin du protocole de Transport.

2.1.2.1 Les structures de données

Allocation de ressource et processus de réception

Le processus d'interface en réception est principalement caractérisé par les travaux d'allocation de ressources qu'il effectue. L'allocation de ressources peut revêtir, selon la nature de l'environnement hôte, deux aspects:

. La construction de listes de demandes classiques;

. La description, par le biais d'un PnPDU, de la requête en mémoire partagée (mémoire MAC en réception basse et mémoire utilisateur en réception haute).

L'analyse de cette seconde technique est aussi radicale que performante: l'acquisition d'une requête de taille quelconque, sa description et son chaînage dans la liste de PnPDU nécessite 45 µs. Par contre, la construction d'une liste de demandes, figure 2.1.11, coûte (outre le temps de transfert des informations):

.64 µs pour l'allocation de chaque tampon de donnée, le chaînage de ce dernier et la mise à jour de ses champs de contrôle;

.44 µs pour le descripteur de demande

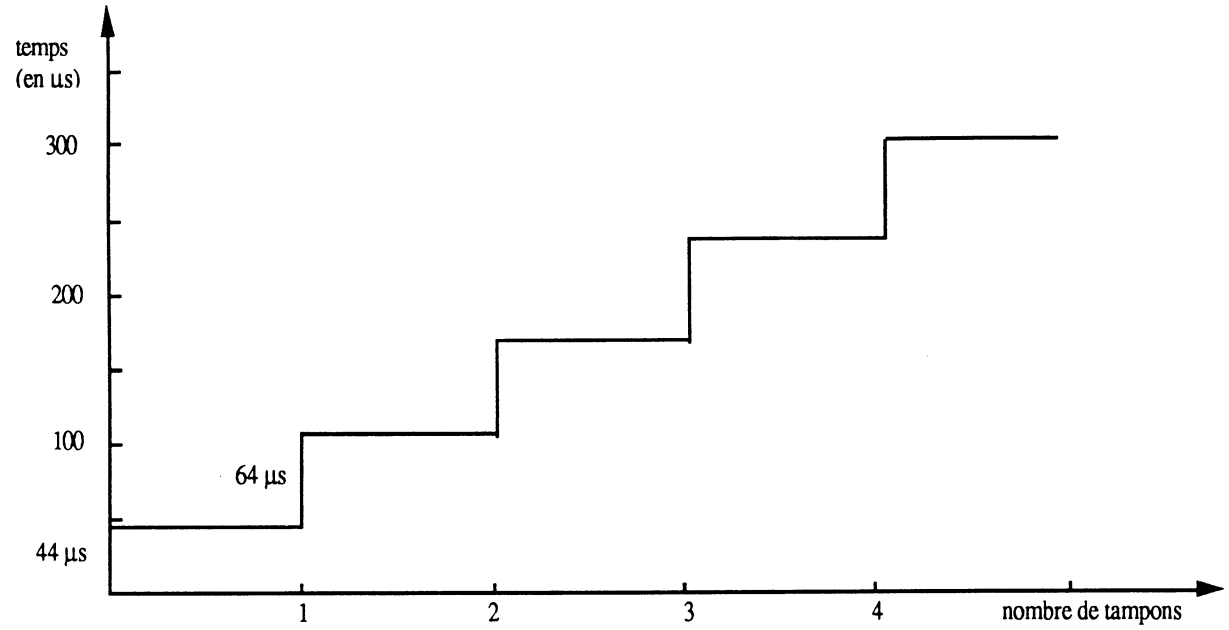


figure 2.1.11; temps de construction des éléments de liste nécessaire au stockage d'une TSDU

Les ruptures de pente correspondent à l'allocation d'un nouveau tampon .

Ainsi, quelque soit le mécanisme de transfert retenu, ces résultats montrent que l'on a tout intérêt à **minimiser le nombre de ressources impliquées dans la description d'une TSDU ou d'une TPDU,** et ce pour une raison bien simple: l'allocation, puis la restitution de ressources coûtent cher!. De plus on observe sur la figure 2.1.11 que l'acquisition d'une SDU est une fois et demi plus longue si elle est stockée sur 3 plutôt que sur un tampon de données.

Etude des règles d'utilisation des structures de données

Nous allons dans ce paragraphe chercher à définir une **stratégie d'allocation de ressources** efficace (nous ne nous intéresserons pas à la réception directe par PnPDU qui ne nécessite aucune étude de stratégie).

Outre la discussion en terme de nombre de ressources à allouer, l'efficacité du modèle de structures de données est liée, via les traitement effectués par les protocoles sur la liste entrante, à la taille de ces structures, et à leur contenu.

Les mesures atomiques réalisées sur le protocole de Transport vont être utilisées, dans un premier temps, pour analyser les règles élaborées au chapitre précédent (paragraphe 1.2.2.1).

Définir une stratégie d'utilisation des structures de données est un problème délicat, intimement lié aux protocoles implémentés. TP4 ne simplifie pas la tâche; la stratégie retenue sera le résultat d'un compromis. XTP, protocole plus régulier, est mieux adapté à l'usage d'un tel modèle. Dans tous les cas, un protocole de niveau quatre intervient sur la liste de demandes construite en réception dans deux situations:

. L'affectation à une connexion, pendant laquelle il parcourt la liste pour y étudier les en-têtes de TPDU.

. La procédure de segmentation/réassemblage; qui nécessite la description des données de l'utilisateur.

Nous étudierons la taille idéale de ces ressources sur quelques exemples quantifiés à partir des résultats obtenus dans la phase de mesures atomiques.

Etude de la taille des descripteurs de demandes

La structure de descripteur de demande laisse à l'implémenteur la possibilité de fixer la taille de la zone réservée au données de l'utilisateur. Nous allons montrer pourquoi il est intéressant de prendre ce champ de même taille que l'en-tête des TSDU ou TPDU reçues. La figure 2.1.12 compare le temps de traitement d'une TSDU de données de 3 kilo-octets dans deux situations:

(1) le descripteur de donnée contient (dans sa zone réservée au stockage de l'en-tête) des données utiles
(2) il ne contient que l'en tête de cette requête.

On y observe alors un gain voisin de 25% au profit de la solution où le descripteur de demande ne contient que l'en-tête de la SDU entrante.

Dans la situation (1), les données utiles débordent sur la zone utile du descripteur de demande; ainsi un PnPDU supplémentaire est nécessaire pour les décrire. De même, prendre une zone de données utiles plus petite que la taille de l'en-tête provoque deux indirections supplémentaires pour étudier l'en-tête complète (une pour l'accès au descripteur de tampon de données puis une seconde pour le tampon de données). Un choix pertinent consiste donc à prendre la taille de la *zone utile du descripteur de demande égale à la taille de l'en-tête du type d'informations traités.* Cette solution est d'autant plus intéressante qu'elle permet, dans une architecture parallèle, le traitement parallèle de l'en-tête (dès que le descripteur de demande est construit) et le transfert des données utiles dans les tampons (uniquement sur des architectures parallèles).

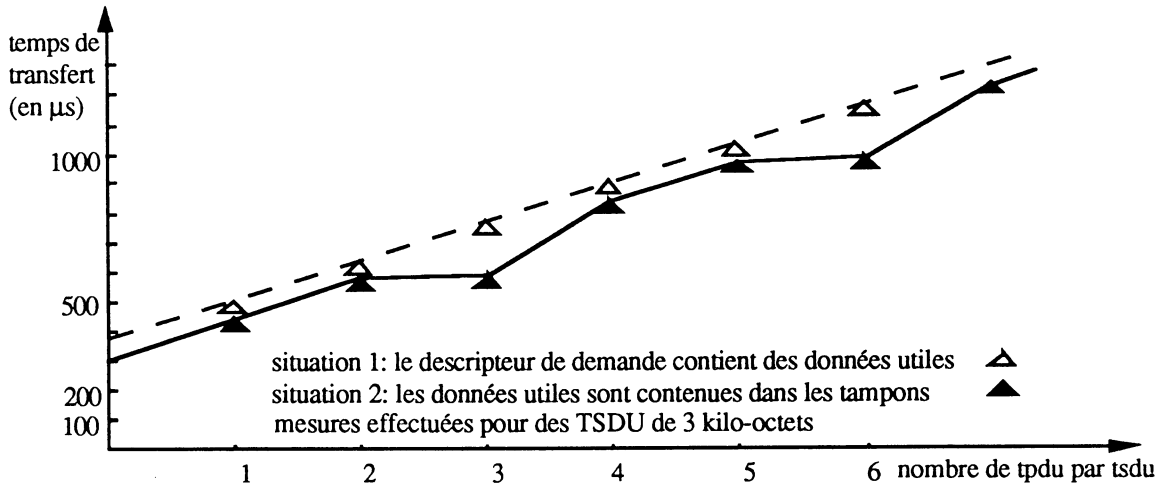


figure 2.1.12; étude de la taille idéale des descripteurs de demandes
sur la mesure du temps de traitement d'une TSDU de 3 kilo-octets par le protocole

Appliqué à TP4, ceci signifie qu'il faut privilégier les TPDU les plus souvent traitées (car le format des en-têtes est variable); et que la taille des descripteurs de demandes en réception haute et basse risque fort d'être différente.

Discussion sur le format des tampons de données

Pour discuter de la taille idéale des tampons de données, nous allons introduire **la taille des segments** (ou taille utile des DT TPDU). Le nombre de segments construits par le protocole dans une TSDU répercute, au niveau du protocole, l'efficacité de la stratégie d'allocation de tampons de données. Ainsi, *si un segment est à cheval sur n ressources, il faudra n PnPDU pour le décrire.* La stratégie d'allocation optimale sera donc celle qui minimisera, au niveau du protocole, le nombre de PnPDU utilisés pour décrire un segment. Deux nouvelles situations vont être analysées:

(1) la taille des segments est supérieure à la taille des tampons de données (le nombre de segments est donc nécessairement inférieur au nombre de tampons).
(2) la taille des segments est inférieure ou égale àla taille des tampons de données.

Les mesures effectuées pour trois segments montrent (figure 2.1.13) que:

. la situation (2) offre des performances toujours supérieures aux performances de la situation (1).
. dans la situation (2), les performances les plus élevées sont obtenues pour l'**égalité entre taille des segments et taille des tampons de données.**

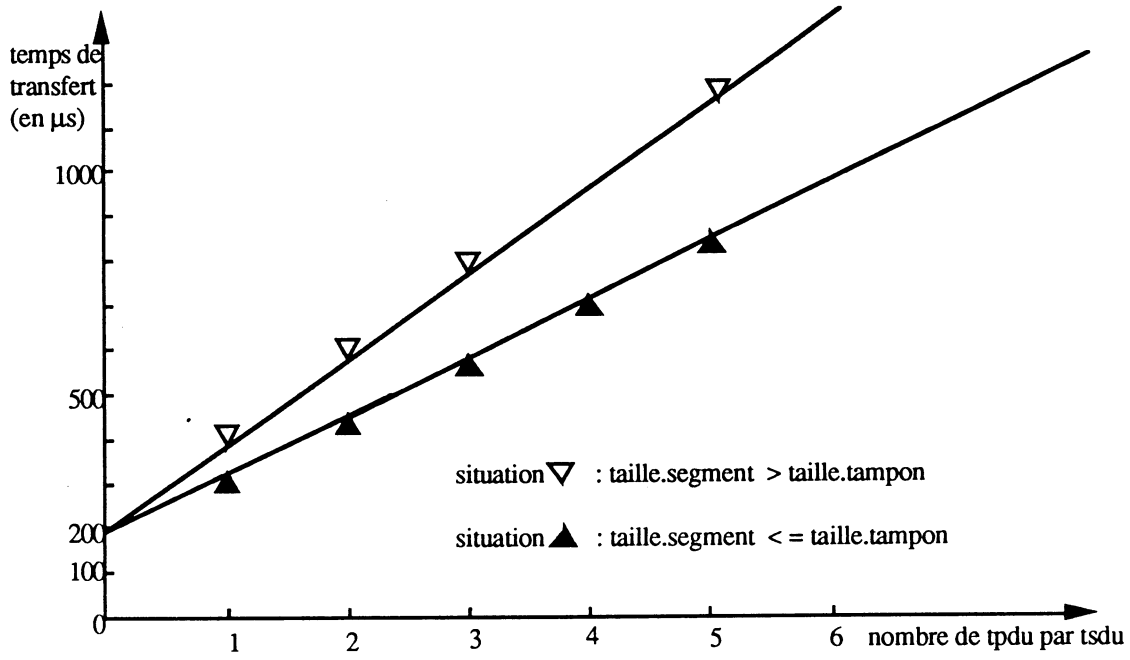


figure 2.1.13; stratégie d'allocation des tampons de données
étudiées sur le temps de traitement d'une TSDU de 3 kilo-octets par le protocole

Reste à analyser quelle solution est la plus optimale entre un unique tampon de données et un tampon par segment:

. du point de vue du protocole de Transport, les deux situations sont quasiment équivalentes. Après description de chaque segment, il faut accéder au segment suivant par deux indirections successives quand on a un tampon par segment; alors qu'aucune indirection n'est nécessaire dans le schéma à tampon unique.

. un tampon par segment demande l'allocation puis la restitution de n + 1 ressources pour n segments. Avec un tampon unique, le gain au niveau des processus d'interface devient considérable.

Mais *allouer des tampons de taille trop grande peut très vite paralyser la mémoire*; et selon le type de gestion mémoire retenue, il peut être plus astucieux d'allouer des ressources de taille réduite (le manque de ressource peut être plus néfaste à l'application que l'allocation et la restitution trop fréquente)

Le choix n'est pas si simple [Boehm 80]; et c'est pourquoi, dans la suite de cette phase de mesures, nous analyserons les deux possibilités, afin de laisser l'implémenteur juge de la stratégie à adopter.

Conséquences sur la complexité en émission

Le processus d'émission reconstitue les informations à émettre en parcourant l'arborescence de PnPDU. L'efficacité de ce processus est donc à la fois liée à:

. l'efficacité de l'algorithme de parcours de l'arborescence;
. le nombre de PnPDU construits par le protocole pour décrire une TPDU ou une TSDU;
. la taille de l'arborescence, soit l'efficacité du processus de restitution.

L'efficacité des performances mesurées en émission est directement liée à la stratégie d'allocation de ressources en réception via le nombre de PnPDU alloués pour décrire les données utilisateur.

L'implémentation réalisée fonctionne avec deux PnPDU minimum par DT TPDU:

. un PnPDU pour décrire l'en-tête de DT TPDU;
. autant de PnPDU que de ressources à décrire pour identifier les données utiles de la TPDU.

On comprend d'autant mieux qu'il est important de minimiser le nombre de PnPDU quand on analyse le travail du processus d'émission sur une arborescence de PnPDU à plusieurs niveaux.

Si chaque PnPDU de niveau n décrit les données utilisateur via p PnPDU de niveau n - 1, on a 3 * p indirections à effectuer pour reconstituer l'élément décrit par un PnPDU du niveau n.

Ce résultat est visualisé figure 2.1.14 où l'on a établi une courbe liant le temps de restitution d'une séquence de PnPDU au nombre de PnPDU associés au niveau inférieur:

.chaque PnPDU du niveau courant doit passer par un PnPDU au niveau inférieur pour accéder aux informations à émettre;

.chaque PnPDU du niveau courant doit passer par deux PnPDU au niveau inférieur pour accéder aux informations à émettre;

. et enfin les PnPDU de la séquence décrivent directement les informations à émettre.

Quand il y a indirection, les PnPDU des couches intermédiaires ne peuvent pas être restitués (un niveau de protocole n'est pas autorisé à restituer une ressource qu'il n'a pas allouée). Le

processus de restitution ne peut que les rendre restituables; quitte à la couche qui les a allouées de les restituer ensuite selon ses propres critères.

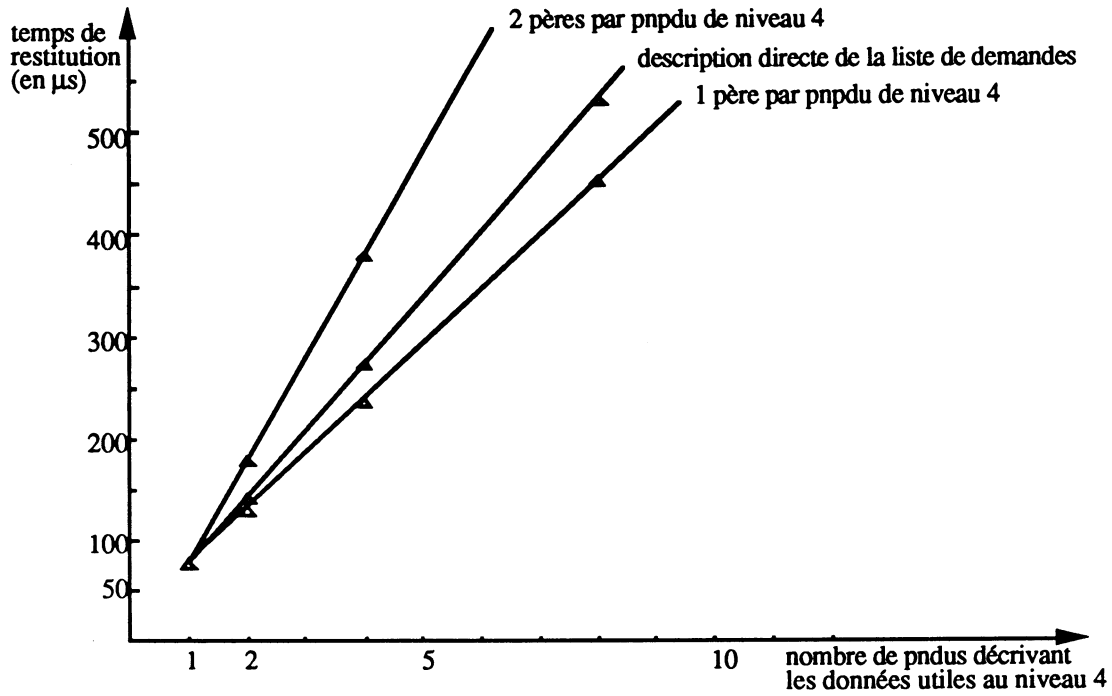


figure 2.1.14; temps de restitution d'une TSDU mémorisée dans un descripteur de demande

Si un PnPDU décrit directement une ressource, il peut (après s'être assuré qu'aucun autre PnPDU ne décrit encore la ressource concernée) restituer à la fois le PnPDU et la ressource décrite.

D'un point de vue local, la restitution directe est plus coûteuse qu'une restitution avec indirection; à condition bien sûr que l'allocation de ressources ait été faite correctement au niveau précédent. L'utilisation de la structure de PnPDU permet donc d'échelonner les travaux de restitution; elle ne les réduit en aucun cas.

27µs est le temps qu'il faut pour émettre un PnPDU qui décrit directement une zone contenant des informations; 28µs est le temps équivalent pour un PnPDU qui décrit une information via un PnPDU de niveau inférieur (dit "père"). Comme en réception, des ruptures de continuité apparaissent chaque fois que le protocole doit accéder à un nouveau PnPDU. La distance entre chaque morceau de droite est constante, égale à 33µs (28 µs plus 5 µs d'accès au suivant).

Le débit du processus d'émission est donc totalement dépendant de la stratégie d'implémentation de ressources au niveau des protocoles implémentés dans le système, eux même dépendant de la stratégie d'allocation en réception.

2.1.2.2 Comportement aux interfaces

Après avoir utilisé les processus d'interface pour évaluer le modèle de structures de données dédié défini au chapitre précédent, nous allons maintenant étudier le comportement et les performances de ces processus. Nous considérons, pour les mesures atomiques, que chaque processus d'interface est implanté seul sur un Transputer. Dans ces conditions, la synchronisation avec le protocole est réalisée au moyen de canaux physiques.

Bien que leurs rôles diffèrent, ils respectent un ensemble de propriétés communes qui nous autorisent à les traiter simultanément:

-Ils assistent le protocole de Transport; ils désynchronisent l'environnement du protocole en gérant, en entrée et en sortie, des listes d'attente.

-Ils sont implantés aux bornes d'un système qui peut intégrer plusieurs niveaux de protocole de communications; leur fonctionnement est indépendant de la complexité du système.

-Ils assurent, de manière identique, l'interface avec l'environnement extérieur, selon des mécanismes basés sur une mémoire partagée et (ou) sur un canal:

. Octet par octet sur un lien série; le Transputer effectue alors une commutation de processus avant le transfert de chaque octet et le coût du transfert est alors catastrophique. Ce mode de transfert a été précisément évalué dans [Diot 89].

. Toujours sur un lien (soit à 12 Méga-bits réels par seconde), on transfère cette fois successivement la longueur de la primitive, puis la primitive complète sous forme d'un tableau d'octets; il s'agit là d'une optimisation de la technique précédente qui ne provoque que deux commutations de processus. Nous l'appellerons transfert par message.

. Par mémoire partagée, avec un débit de 80 Méga-bits par seconde, le processus de réception est averti de l'adresse de la ressource (en mémoire partagée) par canal; on évite ainsi un premier transfert de donnée et on réduit l'utilisation du canal à la synchronisation de deux processus.

-Ces deux processus sont connus pour être des sources de goulots d'étranglements privilégiés [Dang 88b].

Le processus de réception

Outre l'interface avec l'environnement extérieur, le processus de réception est encore chargé de préparer les requêtes entrantes afin de les rendre directement utilisables par le protocole de Transport.

Cette tâche regroupe l'allocation de descripteurs spécialisés (dits descripteurs de demande) et le transfert dans ces descripteurs de la requête entrante, suivi de la mise à jour de ces descripteurs (état, informations de contrôle) et leur chaînage dans une liste destinée au protocole de Transport.

Deux paramètres interviennent principalement dans le temps de réception d'un SDU:

. la longueur de cette SDU (via le temps de transfert);
. la taille des ressources dans lesquelles est transférée la SDU.

De l'étude du processus de réception, nous avons établi les équations suivantes (le temps y est obtenu en micro-secondes):

réception par mémoire partagée: $t = 0.1 * lg\_mess + 64 * nb\_tampons + 44$ (1)
réception par transfert de message: $t = (0.1 + 0.1) * lg\_mess + 64 * nb\_tampons + 52$ (2)

où lg_mess représente la longueur totale de la primitive reçue en octets et nb_tampons le nombre de tampons de données alloués pour mémoriser la partie de la SDU ne tenant pas dans le descripteur de demande. La différence de coefficient directeur des deux équations provient du nombre de transferts requis pour mémoriser la primitive reçue sous liste de demande:

. 0.1 pour un transfert en réception parallèle (depuis la mémoire partagée vers la liste de demande);

. 0.2 pour deux transferts en réception par message sur lien (soit 0.1 pour le transfert par lien virtuel plus 0.1 pour le transfert en liste de demande). Mais les mesures atomiques ayant été réalisées sur un seul Transputer, le transfert sur canal virtuel est simulé par le processeur par un transfert en mémoire. Un véritable transfert par canal physique suivrait un coefficient directeur de 0.66; et l'expression de l'équation (2) devient:

$$t = (0.1 + 0.66) * lg\_mess + 64 * nb\_tampons + 52$$

Rappelons que 44 ou 52 µs correspondent au temps de construction et de chaînage d'un descripteur de demande; 64 µs est le temps équivalent pour un tampon de donnée et de son descripteur.
Enfin la description directe, en mémoire partagée des informations entrantes par PnPDU est une solution alléchante qui évite un transfert de données mais qui n'est pas toujours possible (selon la taille, l'encombrement, le temps d'accès de la mémoire partagée). Afin de minimiser les conflits d'accès à la mémoire, une technique intermédiaire consiste à transférer l'en-tête de la SDU entrante dans la mémoire propre du système pour quelques micro-secondes de plus.

Le processus d'émission

Le processus d'émission assure l'interface entre une suite de protocoles et l'environnement externe:

-Il assure la communication avec l'environnement par un des trois mécanismes de base cité en début de cette étude avec cependant une contrainte: le processus d'émission ne peut éviter un transfert de donnée.
-Il communique avec le protocole de Transport à travers une arborescence de PnPDU de sorte à être complètement désynchronisé du protocole.

Le temps d'émission d'une PDU est donc lié à deux facteurs:

. le temps de parcours de la liste pour accéder à la ressource à émettre; l'efficacité de cette procédure étant elle même dépendante du nombre de PnPDU par TPDU décrite et surtout de l'efficacité de l'algorithme de restitution de ressources qui doit maintenir la liste à sa taille minimum;

. le temps de traitement d'une séquence logique de PnPDU au cours duquel est effectuée l'émission selon un des trois modes étudiés.

Evaluation

Nous avons donc établi deux équations qui définissent complètement le fonctionnement du processus d'émission.

Le **temps de parcours de la liste jusqu'à détection d'un élément à traiter**:

$$\Delta p.liste = 2 + \sum_{1}^{nb\_tpdu} [4 + 11 * nb\_PnPDU] \qquad (3)$$

avec nb_PnPDU le nombre de PnPDU utilisés pour décrire une TPDU et nb_tpdus le nombre de séquences logiques de PnPDU (non émissibles) parcourus avant la séquence à traiter.

le **temps d'émission d'une suite logique** avec indirection sur une seule couche inférieure:

$$\Delta e = 16 + 27 + \sum_{2}^{nb\_PnPDU} [5 + 28 * nb\_père] + x * lg\_mess \qquad (4)$$

avec nb_PnPDU le nombre de PnPDU qui décrivent une même TPDU (ils sont donc, selon le modèle précédemment décrit, liés logiquement). nb_père le nombre de PnPDU de la couche inférieure qui, pour chaque PnPDU de la couche courante, permettent l'accès aux données à transférer. lg_mess la longueur de la SDU transférée en octets et x le coefficient directeur de la droite (0.1 ou 0.76 selon le mode de communication retenu).

L'équation (4) peut être divisée en deux partie distinctes: la recherche des données à émettre et le transfert de ces données (selon l'équation x * lg_mess). Pour une stratégie d'allocation minimale de PnPDU, on retrouve la même pente qu'en réception puisque l'information reconstituée est directement mémorisée dans la mémoire que partage le processus d'émission avec l'environnement externe (soit la mémoire hôte ou la mémoire MAC).

Problème posé par la restitution des ressources

Si le coût de l'indirection par un PnPDU père n'est pas très élevé, le temps de parcours de la liste (équation Δp.liste) est particulièrement coûteux parce que pas indispensable. L'arborescence de PnPDU partagées entre les divers protocoles implémentés et le processus d'émission doit donc, dans la mesure du possible, ne contenir que les éléments à émettre ou susceptibles d'être réémis. Or, pour être restituées la plupart des TPDU doivent attendre le feu vert du protocole qui les a construit.

Un processus de restitution des ressources obsolètes est donc nécessaire. Il doit être

processus. Le coût de la restitution d'une suite logique de PnPDU est donné par l'équation suivante:

$$\Delta r = 16 + 65 + \frac{\sum_{2}^{nb\_PnPDU} [5 + 48 * nb\_père]}{2} \qquad (5)$$

Pour réduire le coût global du processus d'émission, il convient donc de

. minimiser le nombre de parcours des listes.
. supprimer de l'arborescence les séquences de PnPDU devenues inutiles (permet d'accéder plus rapidement aux séquences de PnPDU à émettre)

Discussion en terme de débits; résolution des goulots d'étranglement

Nous en finirons avec cette analyse des processus d'interface par une comparaison de leurs performances en terme de Méga-bits par seconde.C'est en effet le seul moyen que nous ayons pour discuter des possibilités d'interfaçage sur un réseau FDDI. Les équations utilisées pour cette comparaison sont:

$$\Delta input = 0.1 * lg\_mess + 64 * nb\_seg + 44 \quad (\text{équation (1)})$$
$$\Delta output = 0.1 * lg\_mess + 33 * nb\_seg + 43 \quad (\text{équation (4) avec } nb\_père = 1)$$

avec nb_seg le nombre de DT TPDU segmentés dans une TSDU de donnée. Les équations tiennent compte des conditions d'allocation de ressource établies précédemment (soit un tampon par segment en réception, deux PnPDU par segments au niveau Transport et un au niveau Réseau).

Pour des TSDU de 4 koctets segmentés en 4 TPDU (de 1 kilo-octets chaque selon la taille de DT TPDU recommandée par le profil CNMA, prévu pour fonctionner sur Ethernet), le débit en réception est de 45 Mbps; en émission de 55 Mbps. Dans les conditions de FDDI, la taille de TPDU maximum que peut transférer le réseau passe alors à 4 kilo-octets. Les mêmes TSDU ne seront donc pas segmentés et le débit s'élèvera à 63 Mbps en réception et 66 Mbps en émission.

L'étude de la limite, pour une taille de TSDU tendant vers l'infini, du débit en émission et en réception fait apparaître une limite supérieure égale respectivement à 69 Mbps et 74 Mbps. Et dans le meilleur des cas (TSDU et segments de très grande taille) la limite devient identique pour les deux processus: 80 Mbps; limites qui ne pourront être approchées que si chaque

processus est implémenté sur son propre Transputer (Les regrouper deux à deux sur un processeur unique aurait pour conséquence de réduire ce débit de moitié au moins).

Quels que soient les efforts consentis pour accroître les performances du protocole de Transport ***les processus d'émission et de réception limiteront toujours le débit du protocole.***

La limite est imposée par la vitesse des transferts en mémoire sur le Transputer et le seul moyen de la franchir serait de supprimer tout transfert d'information utile. Or, si nous avons vu que les transfert de données pouvaient être évités en réception, ils sont par contre incontournables en émission.

Utiliser les PnPDU pour décrire les primitives entrantes en mémoire partagée permet d'atteindre plus de 300 Mbps pour des TSDU de 4 koctets et plus de 650 Mbps pour des TSDU de 8 koctets.

Par contre, on semble se trouver, en émission, dans une impasse exclusivement due à la vitesse de transfert en mémoire du Transputer (80 Mbps en mémoire externe). Deux solutions totalement différentes pourraient permettre de sortir de cette impasse:

-**accélérer le temps d'accès mémoire du Transputer**; on sait en effet que le Transputer accède en 4 cycles à la mémoire externe et en 1 cycle à sa mémoire locale. Le débit d'un transfert de donnée en mémoire locale serait donc de 320 Mbps, de sorte qu'il serait possible d'atteindre les 100 Mbps de FDDI dans les conditions d'implémentation actuelles. Malheureusement la taille de la mémoire interne d'un Transputer est limitée à 4 koctets donc complètement inutilisable et surtout, elle n'est pas partageable avec un autre Transputer. Mais réduire le nombre de cycles nécessaires à l'accès mémoire n'est pas une solution satisfaisante à long terme; les performances du protocole se trouvant augmentées proportionnellement, le processus d'émission restera un goulot d'étranglement privilégié.

-**Accroître le parallélisme** en émission, à savoir séparer les algorithmes de parcours et de recherche des informations utiles de la procédure de transfert. Cependant le transfert restera limité à 80 Mbps, et la limite ne sera qu'insuffisamment repoussée.

-**Accélérer la vitesse en émission,** soit en accélérant le temps d'accès à la mémoire (solution déjà discutée), soit en utilisant pour ce transfert un lien série haut débit (de style optique par exemple). On remplacerait ainsi la lecture/écriture nécessaire au transfert par une lecture suivie d'une émission sur lien (avec prise en charge parallèlement sur une interface

Restons somme toute méfiants à la lecture des valeurs proposées et à l'optimisme des solutions. Ils montrent que le goulot d'étranglement classique des interfaces peut être supprimé mais, et les mesures effectuées dans ce chapitre le mettent clairement en évidence, pas dans n'importe quelles conditions d'implantation. De plus nous sommes encore incapables d'évaluer l'effet de ces techniques d'implémentation sur le nombre de conflits et sur la disponibilité des mémoires utilisées.

2.1.2.3 Fonctionnalités du protocole de Transport

Toujours dans les conditions de mesures définies au début de ce chapitre de mesures atomiques, et considérant comme acquise la stratégie d'allocation de ressources mise au point dans la section précédente, nous allons maintenant analyser les résultats obtenus pour le protocole de Transport OSI classe 4.

Le détail des équations analysées est proposé en annexe B. Les équations ont été établies pour les unités de données les plus traitées par une couche quatre classique, à savoir:

. les primitives service de requête de donnée (ou t.dt.req), émises par l'utilisateur du service Transport. Le traitement spécifique entraîné par cet événement est la construction de DT TPDU par segmentation de la TSDU entrante; tâche charnière à implémenter avec beaucoup de précautions. Elle introduit dans le débit les notions de taille des DT TPDU et de stratégie d'allocation de PnPDU;

. les DT TPDU reçus de la couche Réseau (donc contenus dans une liste de PnPDUs). La réception d'un tel DT TPDU provoque la vérification de la numérotation, la procédure de réassemblage suivie éventuellement de l'émission d'une primitive de service puis de l'émission d'une AK TPDU (selon une fréquence déterminée par l'implémenteur).

. la réception d'une AK TPDU (toujours depuis la couche Réseau), qui doit permettre au protocole de restituer ou d'autoriser la restitution des ressources allouées en réception.

En ce qui concerne l'utilisation des donnée express (dont le temps de traitement a été mesuré constant sur notre implémentation), [Colella 85] a montré que l'utilisation trop fréquente de ce type de TPDU nuit considérablement au débit de l'application. Pour un débit de 1560 t.ed.req traités et acquittés par seconde et 3300 ED TPDU traités en réception basse, le débit de données normales sur le réseau devient proche de zéro (rappelons que via la procédure de numérotation, elles sont prioritaires sur les TPDU de données normales et en bloquent le traitement).

Nous avons établi, pour les tâches décrites ci-dessus une première série d'équations (donc de courbes) que l'on pourra ensuite combiner à souhait pour en établir d'autres plus complexes et étudier en détail le comportement de chaque paramètre. Ces courbes expriment pour la plupart un débit en terme de TSDU par seconde.

-Le nombre de t.dt.req traitées par le protocole en une seconde, depuis la prise en charge dans la liste de demande jusqu'à émission de tous les DT TPDU construits. Cette équation nous permet d'étudier l'influence de la stratégie d'allocation de tampons de données en réception sur le débit du protocole.

-Le nombre de DT TPDU construites et émises en une seconde par le protocole (débit précédent multiplié par le nombre de DT TPDU construites dans chaque t.dt.req).
-Le nombre de TPDU réassemblés (reçus de la couche Réseau) et émises vers l'utilisateur du service Transport sous forme de TSDU chaque seconde.

-Le nombre d'acquittement (ou d'AK TPDU) que le protocole peut traiter en une seconde; cette équation ne présente aucun intérêt seule, mais combinée avec le débit de t.dt.req, puisque les acquittements sont émis par l'entité distante en réponse à la réception de DT TPDU.

-le nombre de TSDU traités de bout en bout par le protocole; une t.dt.req est définitivement traitée lorsque les ressources allouées pour son traitement ont été restituées. Cette équation intègre le traitement des t.dt.req et des AK TPDU qui autorisent la restitution les ressources. Elle nous permet d'analyser les problèmes de stratégie d'acquittement et de restitution de ressources.

Les paramètres fondamentaux qui apparaissent dans ces équations sont:

-La taille utile d'une TSDU de données (de type t.dt.req ou t.dt.ind) lg_TSDU. La longueur totale d'une TSDU étant égale à la longueur utile augmentée de six octets d'en-tête de service.

-La taille utile des DT TPDU.

-Le nombre de segments nb_seg; nb_seg correspond au nombre de DT TPDU construits dans une TSDU (soit nb_seg = E (lg_TSDU / lg_TPDU) + 1).

Rappelons que la taille des DT TPDU recommandée par le profil CNMA concerne la longueur totale d'une DT TPDU. Ainsi une DT TPDU de 1024 octets ne contient-il que 1019 octets utiles.

-la taille de la fenêtre de contrôle de flux (w_size) mesurée pour des valeurs allant de 1 à 32 segments.

Evaluation préliminaire sans considération des acquittements

Pour commencer cette analyse de mesures atomiques, la figure 2.1.15 donne une première idée des performances que cette implémentation nous permettra d'atteindre en environnement multi-processeurs. Ces valeurs ont été obtenues sans comptabiliser les temps de traitement des acquittements, pour une taille de fenêtre de contrôle de flux au moins égale à la taille d'une TSDU. Nous y observons:

-Le débit de l'application apparaît plafonner à 3000 TSDU par seconde en émission de données et à 2000 TSDU par seconde en réception. La différence est en partie causée:

. par la lourdeur du protocole qui impose des en-têtes de taille variable, longues à analyser, à la réception de chaque DT TPDU;
. par la procédure de segmentation-réassemblage, plus coûteuse au réassemblage (on doit systématiquement étudier chaque DT TPDU, les mémoriser à leur place et analyser le contenu de la fenêtre pour y détecter la présence d'une éventuelle TSDU complète à émettre) qu'à la segmentation (grâce à l'utilisation des PnPDU, le découpage n'est que virtuel et l'émission des DT TPDU immédiate).

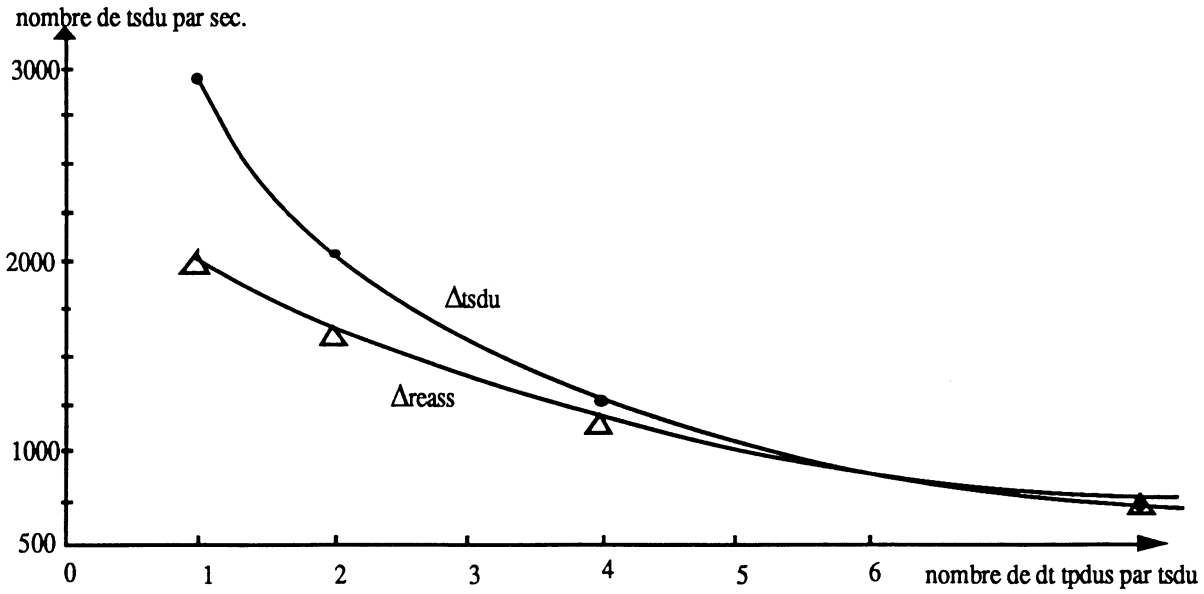


figure 2.1.15; débit du protocole de transport en tsdu par seconde reçues de l'utilisateurs (Δtsdu) et remises à l'utilisateur (Δreass)

Le chiffre de 3000 TSDU montre de manière indiscutable, que les très hauts débits ne pourront être atteints, pour ce type d'implémentation, que pour des TSDU de très grande taille. Pour des TSDU de 4 kilo-octets segmentés en TPDU de 1 kilo, le débit de l'application est limité à 40 Mbps. Ces mêmes TSDU segmentées en TPDU de 4 koctets seront transmissibles à 100 Mbps! Considérant enfin des TSDU de 8 Kilo-octets, et des TPDU de 4, on atteint aisément 130 Mbps. Ainsi les performances d'un réseau FDDI semblent accessibles depuis notre implémentation de TP4.

Le protocole de Transport OSI classe 4 semble donc pouvoir s'insérer dans le club très fermé des protocoles 100 Méga-bits pour des TSDU de grande taille, soit pour des applications plus proches du transfert de fichier que du temps réel (notons que les chiffres étudiés ne prennent pas en considération les acquittements qui, selon la fréquence à laquelle ils vont être transmis par l'entité distante, vont plus ou moins affecter les performances actuelles).

-le traitement d'une TSDU est indépendant de la longueur de cette TSDU; et les structures de donnée dédiées fonctionnent donc correctement. Par contre cette dépendance est reportée au niveau de la procédure de ***segmentation***: le nombre de DT TPDU segmentés dans une TSDU est directement lié à la taille de cette TSDU et à la taille négociée des DT TPDU. La procédure de segmentation se trouve donc projetée au centre du débat puisqu'elle provoque une diminution du débit de moitié si on passe, pour une TSDU donnée, de 1 à 4 segments lors de sa fragmentation.

La perte au réassemblage n'est quant à elle que de 30% (courbe Δreass): si la segmentation est systématique au traitement d'une t.dt.req, le réassemblage n'est entrepris que sur les fin de TSDU ou les DT TPDU reçus hors séquence.

-A partir de quatre segments par TSDU, les performances en Emission et Réception sont sensiblement égales. Ceci s'explique par le fait que, pour des TSDU peu segmentés, la procédure de réassemblage doit être effectuées très souvent. Quand le nombre de segments est plus important, on réassemble alors plus rarement (le travail spécifique de réassemblage n'est exécuté qu'à la réception d'une fin de TSDU ou d'un TPDU hors séquence). Notons enfin qu'il manque, pour équilibrer les deux courbes, la considération du temps de traitement des acquittements.

Stratégie d'acquittement et performances

C'est de la stratégie d'acquittement employée par l'entité distante, de la complexité de l'en-tête d'AK TPDU, et des travaux de restitution à effectuer à la réception d'une AK TPDU que vont

dépendre les véritables performances du protocole de transport. Notons que cette quantification ne tient cependant pas compte des problèmes de gestion dynamique de la mémoire et des problèmes de latence; Ainsi le délai de réponse de l'entité distante n'est pas comptabilisé. Nous allons, dans ce paragraphe, évaluer l'influence de la stratégie d'acquittement sur les performances présentées figures 2.1.15.

Effet de la stratégie d'acquittement

Pour une fenêtre de contrôle de flux de taille 8, toujours dimensionnée de telle sorte qu'une TSDU au moins puisse y être mémorisée, nous avons envisagé deux stratégies d'acquittement classiques:

.dans la première stratégie (dite par TSDU), l'entité Transport destinataire des données émet un acquittement pour toute TSDU qu'elle a réassemblée et retransmise.

.selon la taille de sa fenêtre de contrôle de flux, l'entité destinataire n'envoie d'acquittement que lorsque sa fenêtre est pleine (stratégie dite par fenêtre).

La figure 2.1.16 nous permet donc de comparer le débit du protocole en Emission (transfert des TSDU transmises par l'utilisateur du service Transport):

. sans compter le temps de traitement des acquittement reçus de l'entité distante (courbe Δtsdu de la figure 2.1.15);
. pour une stratégie à un acquittement par TSDU
. pour une stratégie à un acquittement par fenêtre

Rappelons tout d'abord que, dans TP4, c'est le récepteur des données qui est seul responsable de la fréquence à laquelle il émet des acquittements; et c'est lui qui a la charge de calculer cette fréquence. Les deux stratégies choisies sont donc les plus classiques rencontrées sur TP4. Nous n'avons pas retenu la stratégie d'acquittement sur temporisation qui est à la fois coûteuse (dépendance du gestionnaire de temporisations) et moins sûre.

Une analyse fonctionnelle préalable des deux mécanismes révèle quelques différences:

. Plus on acquitte souvent, plus la gestion des ressources est efficace, et enfin plus le fonctionnement du protocole distant est régulier (soit qu'il ne fonctionne pas par vagues de données). Par contre on encombre alors le réseau avec quantité de TPDU qui ne transfèrent aucune donnée utile et on multiplie chez l'émetteur les travaux de mise à jour

des bornes de la fenêtre du récepteur, avec un sur-coût entraîné par le décalage des informations dans la fenêtre et par les autres mises à jour du contexte de connexion.

. Acquitter trop rarement peut entraîner, en cas de retard d'un acquittement ou de perte des retransmissions massives de données déjà reçues, et, à court terme, l'étouffement du réseau.

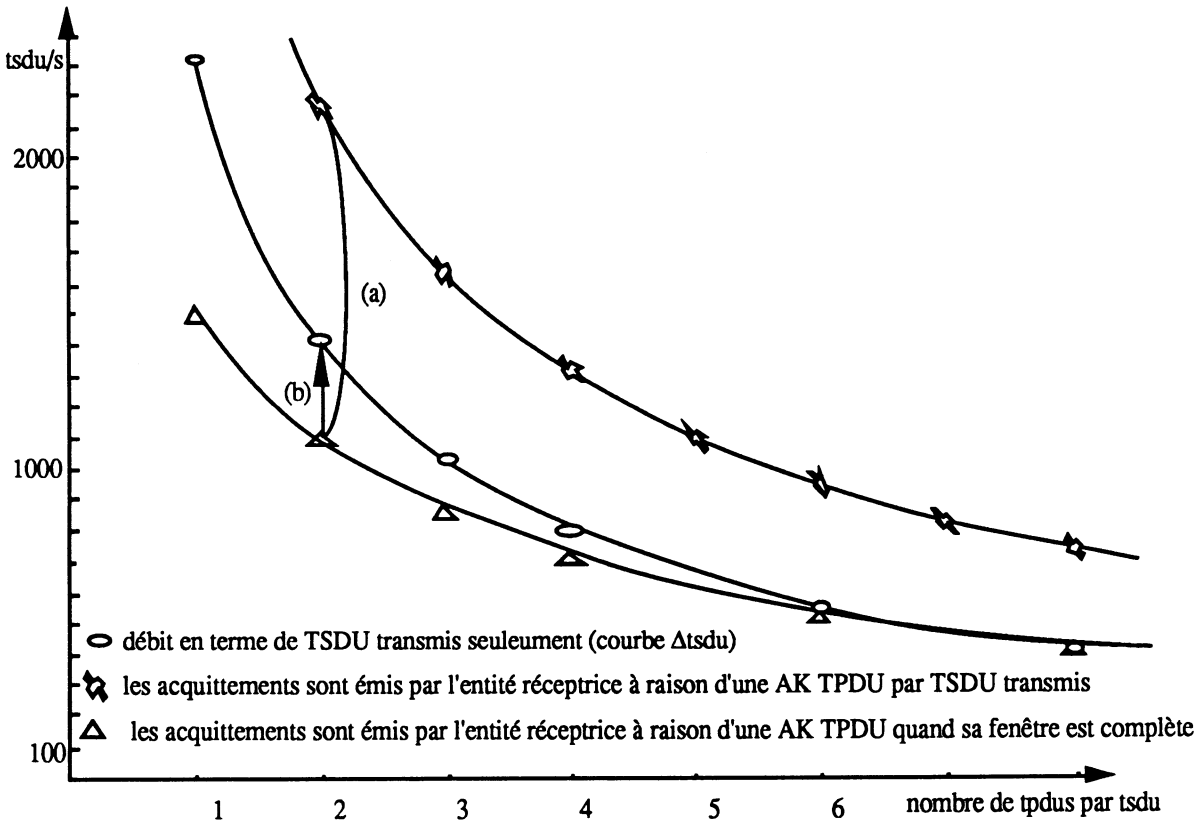


figure 2.1.16; débit du protocole de Transport en TSDU par seconde

Le choix d'une stratégie plutôt que l'autre est une fois de plus le résultat d'un compromis guidé par les conditions d'implémentation, la gestion de la mémoire, ...

Du point de vue des chiffres, on constate une dégradation des performances de l'ordre de 50% (chemin (a)) avec l'introduction d'une AK TPDU tous les deux TPDU. Passer d'une stratégie à une autre plus économique (une AK TPDU quand la fenêtre du récepteur est pleine, soit tous les 8 TPDU) permet de reprendre 27% (chemin (b)) sur le débit du protocole et de minimiser ainsi l'effet des acquittements. En terme de fréquence des TPDU sur le réseau, les pourcentages théoriques correspondant sont:

-Stratégie d'acquittement par TSDU: une TPDU supplémentaire sur le réseau par TSDU, soit

un accroissement de (nb_seg + 1 / nb_seg) TPDU à traiter. Dans le cas de figure étudié (chemin (a); chaque TSDU est segmenté en deux TPDU), cela représente 50 % de TPDU en plus sur le réseau.

-Selon le chemin (b), on passe maintenant, pour une fenêtre complète de (w_size / nb_seg) TPDU supplémentaires à une TPDU AK supplémentaire, soit un gain de 25 % pour une fenêtre de taille 8 et deux TPDU par TSDU (pour 27 % lus sur la figure 2.1.16).

Les pertes occasionnées par l'introduction d'acquittement s'expliquent par la structure et le rôle même de l'acquittement:

-La partie fixe qui contient les informations nécessaires à l'identification de la TPDU et le numéro de la prochaine DT TPDU attendue. Le coût du traitement de ces information est réduit si on n'y inclut pas le coût de la restitution des TPDU ainsi acquittées.

-Les paramètres de contrôle de flux destinés à mettre à jour les variables de contrôle de flux côté émetteur. Le coût du traitement de ces paramètres est d'environ 50 µs, sans compter les effets qu'ils entraînent en cas de réduction de crédit. On comprend mieux pourquoi CNMA conseille de ne pas utiliser la procédure de réduction de crédit et surtout de ne pas envoyer systématiquement de paramètres de contrôle de flux dans une AK TPDU.

Le contrôle de flux par fenêtre et acquittement est donc très coûteux. On lui préférera une technique de contrôle de taux (ou "rate control") qui ne nécessite pas d'acquittement, mais quelques rares TPDU de contrôle pour mettre à jour les paramètres spécifiques. La fréquence des acquittements étant laissée, sur TP4, à la seule appréciation du destinataire des données utilisateur, il s'avère difficile de développer une stratégie d'acquittement plus efficace.

Notons pour terminer que, ramenés en Méga-bits par seconde, et pour des TSDU de 8 Kilo-octets, on atteint des débits de 160 Mbps quand la TSDU d'origine n'est pas segmentée et de 95 Mbps pour des DT TPDU de 4 Kilo-octets. On frôle alors les 100 Méga-bits par seconde.

Taille de la fenêtre de contrôle de flux

Partant d'une stratégie d'acquittement par fenêtre, et afin d'accroître les performances de l'implémentation, nous avons recherché quelle était la taille idéale de la fenêtre de contrôle de flux du récepteur (figure 2.1.17). Entre 6 et 16 segments, les performances du protocole sont sensiblement égales; en dessous elles chutent rapidement; au delà la croissance est presque nulle. Ces résultats confirment ceux énoncés dans [Svodobova 88].

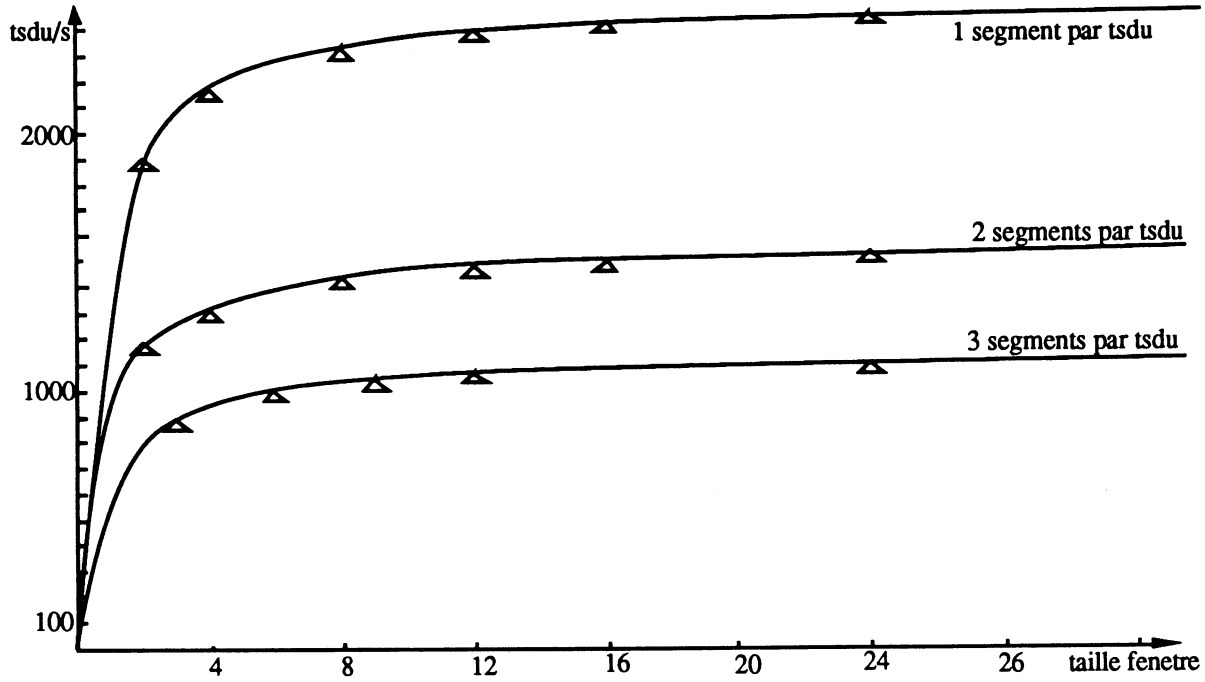


figure 2.1.17; évaluation du débit du protocole de Transport
en fonction de la taille de la fenêtre de contrôle de flux

Le choix ultime de la taille pour une implémentation donnée devant être guidé par les paramètres suivants:

.disponibilité de la mémoire dans laquelle sont allouées les ressources nécessaires à la survie de la connexion.

.dimensions limites de la file d'attente en émission. La fenêtre peut alors jouer un rôle de tampon.

.le nombre de TSDU contenus dans la fenêtre doit si possible être entier. Ce qui simplifie la procédure d'acquittement quand la fenêtre est remplie.

Cet intervalle définit donc des conditions souhaitables à l'ouverture d'une connexion, quand les deux entités échangent leur crédit initial (ou taille initiale de la fenêtre).

Conséquences de la gestion de ressources aux interfaces

Directement liée à la fréquence des acquittements, la restitution de ressources est garante de l'efficacité du protocole pour plusieurs raisons:

. elle garantit une exploitation optimale de la mémoire,
. minimise la quantité instantanée de ressources allouées,
. protège l'application contre les pénuries de ressources.

Minimiser le coût de ces tâches de restitution apporterait aux performances du protocole un gain non négligeable. Choisir une solution plutôt qu'une autre est délicat sans connaître le contexte de travail. Nous allons envisager quatre méthodes pour réduire le coût de la restitution au niveau du protocole de Transport. Nous discuterons de la stratégie à adopter suite à l'étude menée sur plate-forme de test.

allocation minimale de tampons de données

Optimiser l'allocation de ressources en réception minimise le temps de traitement en réception. Nous allons comparer (figure 2.1.18) les performances de l'application lorsque le processus de réception alloue une ressource unique en réception et lorsqu'il alloue un Tampon par TPDU.

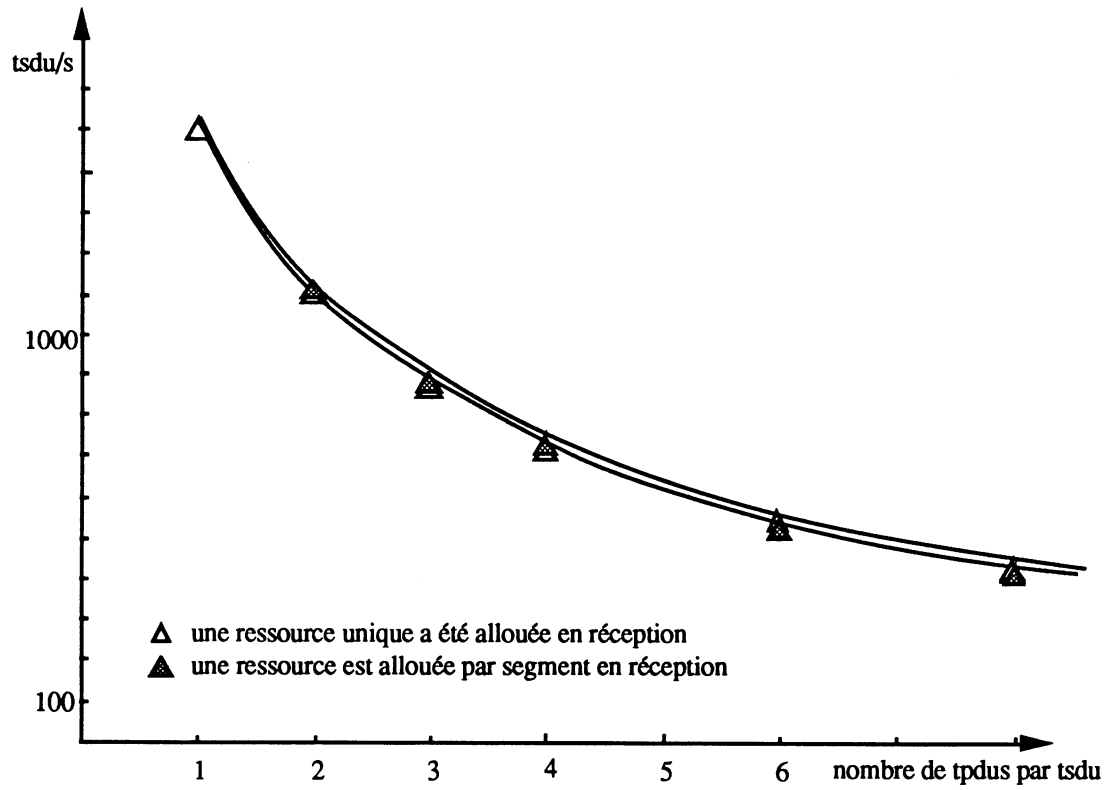


figure 2.1.18; évaluation du gain obtenu à l'allocation d'un unique tampon de données.

Un gain minimal est obtenu au profit de la solution à ressource unique. La faiblesse de ce gain est pourtant simple à justifier:

.nous l'avons montré, l'une ou l'autre des solution n'affecte pas le nombre de PnPDU utilisés pour décrire une DT TPDU;

.Seule tâche du protocole où peut se ressentir l'effet de la stratégie d'allocation en réception: la restitution des ressources devenues obsolètes. Outre la restitution à proprement parler des ressources, le processus chargé de cette tâche doit auparavant s'assurer que plus aucun PnPDU ne décrit la ressource qu'il est chargé de restituer. Dans le cas d'une ressource unique en réception, cette étude est nécessaire et coûteuse; par contre si une ressource a été affectée par TPDU, la restitution est systématique, sans étude préalable. Les deux tâches semblent s'équilibrer.

Localisation du processus de restitution

La restitution de ressources peut revêtir divers aspects:

-Elle peut être prise en charge par le processus d'émission, ce qui minimise le nombre de parcours de liste mais ralentit le délai avant l'émission d'une SDU (il peut en effet avoir plusieurs séquences à restituer avant de détecter la séquence à émettre).

-On peut encore demander au processus d'émission de ne restituer que lorsqu'il n'a rien à envoyer; solution qui en cas de surcharge à l'émission peut très vite paralyser l'arborescence.

-Le processus de restitution peut être totalement indépendant du processus d'émission; solution qui peut s'avérer efficace si ce processus est implanté sur son propre processeur, mais qui rajoute un utilisateur supplémentaire sur une liste que partageaient déjà les protocoles et le processus d'émission.

-Enfin une dernière solution consiste à faire restituer par le processus d'émission les séquences qui sont fonctionnellement restituables après émission et de faire restituer les autres (soit les DT TPDU et autre ED TPDU) par le protocole courant puisque c'est lui qui décide si cette restitution est possible. On minimise alors, dans une solution de compromis, le nombre de parcours et la promptitude à restituer.

Afin d'évaluer ces diverses possiblités, La figure 2.1.19 compare les performances du protocole de Transport quand il restitue lui-même les ressources acquittées par AK TPDU (dernière solution proposée) et quand il se contente de les rendre restituables (dans ce cas, la restitution peut être prise en charge par un processus indépendant).

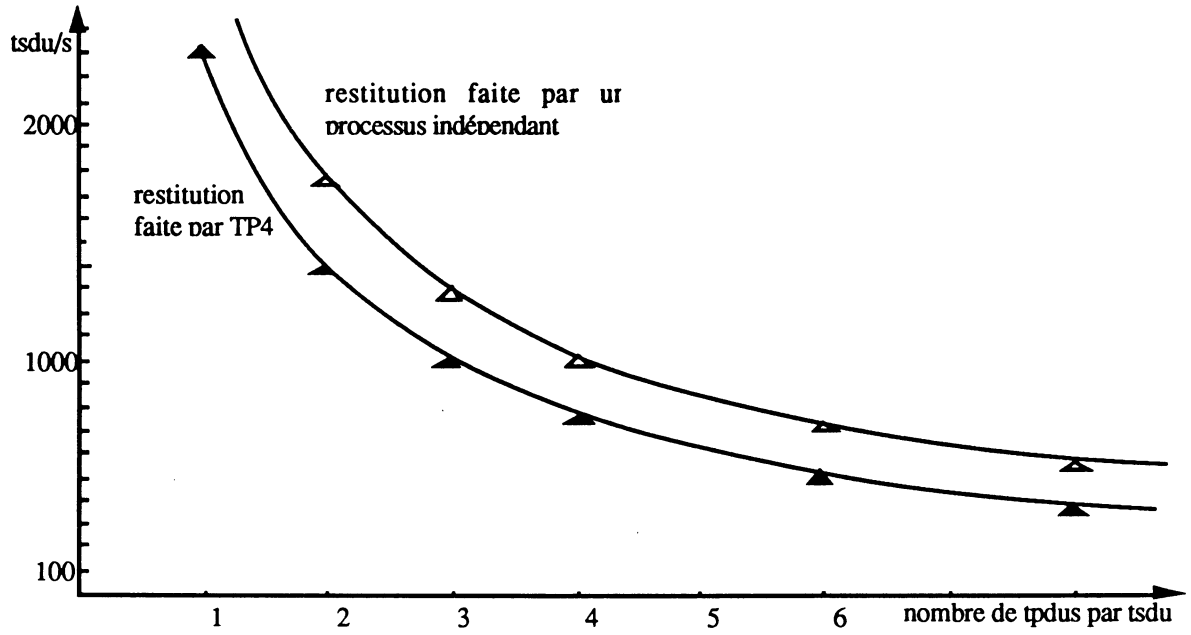


figure 2.1.19; débit du protocole en fonction de la localisation du processus de restitution

Le gain est cette fois très intéressant puisqu'il nous permet d'augmenter les performances du protocole de l'ordre de 20 % (quand il n'y a pas de segmentation de la TSDU à transmettre) à 30 % (les TSDU sont dans ce cas segmentées). En d'autres termes, on observe un débit de nos fameuses TSDU de 3000 TSDU de 8 kilo-octets par seconde pour des DT TPDU de même taille, et de 1800 TSDU par seconde pour des TPDU de 4 Kilo-octets. Soit respectivement 180 Méga-bits par seconde et 150 Méga-bits par seconde; performance intéressantes pour un protocole comme TP4, qui semblait confiné dans des performances médiocres de par sa structure et ses fonctionnalités.

Gestion de la mémoire

La gestion mémoire est d'une certaine manière responsable des performances en restitution de ressources. Nous avons montré dans le chapitre précédent que deux types de gestion étaient possibles:

. la gestion dynamique; ne réserve pas de ressources particulières à l'ouverture des connexions. C'est à l'avènement de chaque SDU que les ressources sont allouées. Cette technique, bien que multipliant les allocations et les restitutions de ressources, permet une meilleure exploitation de la mémoire.

. avec une gestion statique, on alloue à l'ouverture de la connexion un nombre de ressources fixes (autant que la taille de la fenêtre) qui seront utilisées pendant toute la vie

de la connexion. Ainsi pas de charge de restitution de PnPDU suite à la réception d'acquittement. Les seules ressources à restituer sont les ressources allouées en réception.

La comparaison des deux techniques (figure 2.1.20), appliquées à notre implémentation, montre qu'un gain moyen de 25 % peut être obtenu grâce à une gestion statique.

D'un point de vue théorique, la gestion statique permet d'éviter l'allocation systématique de trois ressources, (et le formatage de deux de ces ressources), pour le traitement d'une DT TPDU. Ce travail d'allocation est reporté au moment de l'ouverture de la connexion et les ressources allouées sont la propriété de la connexion pendant toute la durée de vie de cette dernière. La restitution de ces mêmes ressources sera effectuée sans autre forme de procès à la fermeture de la connexion. Il ne reste alors à restituer régulièrement que les ressources allouées en réception.

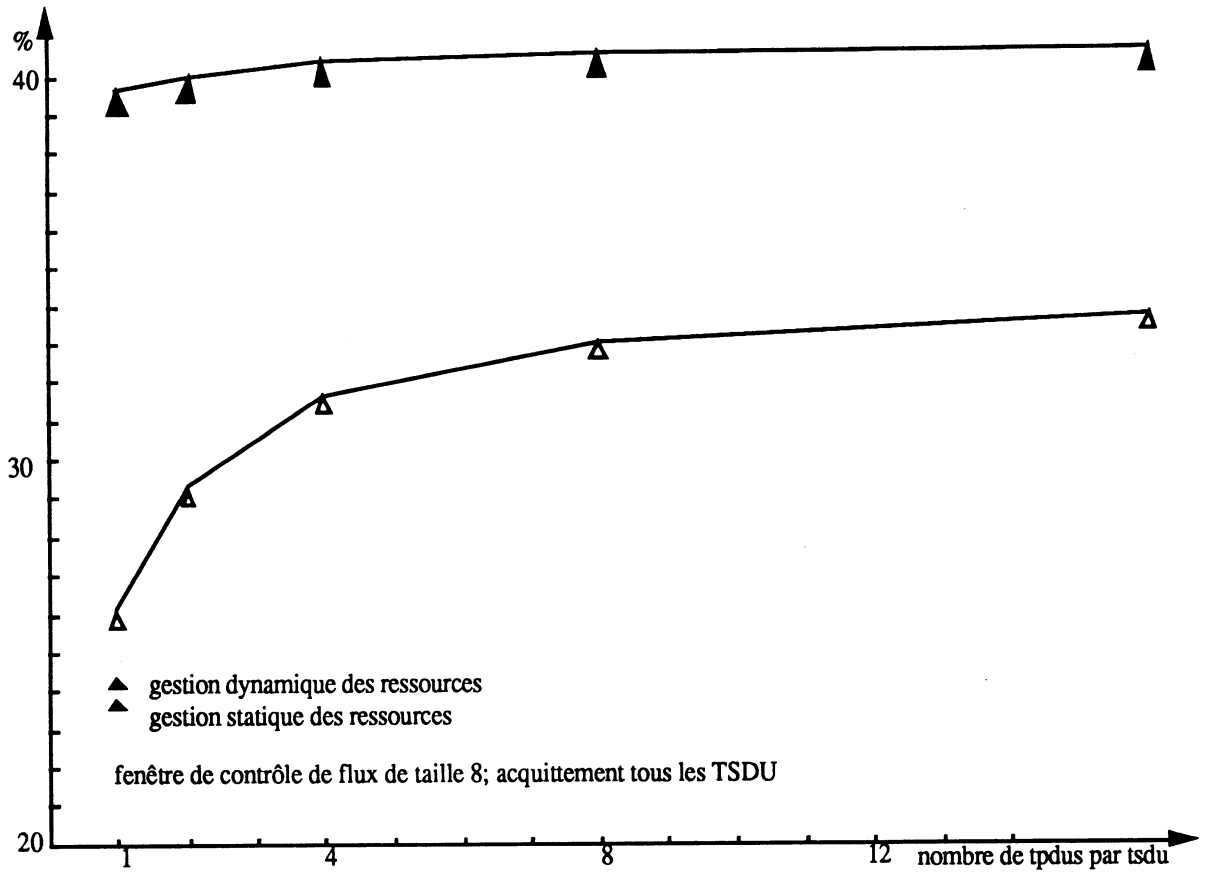


figure 2.1.20; proportion du temps de traitement d'une TSDU consommé par les travaux d'allocation et de restitution de ressources en fonction du type de gestion de ressources

Le gain réalisé, en terme de proportion de travaux de gestion de ressources, représente, en fonction de la taille de la fenêtre, entre 80 et 100% (100% est la limite supérieure impossible à

atteindre). Ramené aux performances de l'application, ce gain représente encore de 20 à 40% du débit de l'application.

Optimisation du modèle de PnPDU

A partir de l'implémentation avec gestion de mémoire dynamique, nous avons évalué le gain réalisable avec un mécanisme de *prédiction d'en-tête* sur les PnPDU (une zone réservée à l'en-tête de la trame décrite est rajoutée au modèle de PnPDU). Un gain de 50 à 65% des travaux de gestion mémoire est réalisable grâce à cette optimisation du modèle de PnPDU (un PnPDU suffit alors pour décrire une TPDU).

Mais ces mesures ne sont pas suffisantes pour évaluer tous les effets de bord de l'allocation et de la restitution de ressources. Les chiffres établis nous permettent de mieux cerner le problème pour qu'il soit plus simple à discuter en environnement multi-Transputers.

Segmentation et réassemblage

Depuis le début de ces analyses de performances, nous n'avons cessé de constater des chutes de débits considérables à chaque apparition de la procédure de segmentation-réassemblage. Et tous les travaux entrepris pour accroître le débit du protocole n'ont en rien affaibli l'effet dévastateur de cette procédure. Nous allons donc consacrer la dernière partie de cette étude de performances à l'étude de la procédure de segmentation-réassemblage.

Coût à la segmentation

La procédure de segmentation a pour but d'adapter la taille des TSDU transmis par l'utilisateur du service Transport à la taille des LPDU acceptables par le réseau. Cette taille maximum est imposée par des contraintes techniques dues à l'architecture du réseau sous-jacent, à sa taille. Il est donc inconcevable de se passer de la procédure de segmentation. Cette procédure est en effet une fonctionnalité du protocole par laquelle doivent passer toutes les TSDU traitées par le protocole. Implémentée classiquement, elle multiplie les mouvements locaux de données. La structure de PnPDU est spécialement adaptée à cette procédure puisqu'elle permet d'éviter les transferts de données, et le découpage physique des TSDU.

Nous commencerons par étudier la figure 2.1.21 qui donne, pour des TSDU de taille constante, le débit du protocole en Méga-bits par seconde (afin de ne pas s'encombrer de phénomènes parasites pour l'analyse du problème, nous n'avons pas comptabilisé le temps de traitement des acquittements).

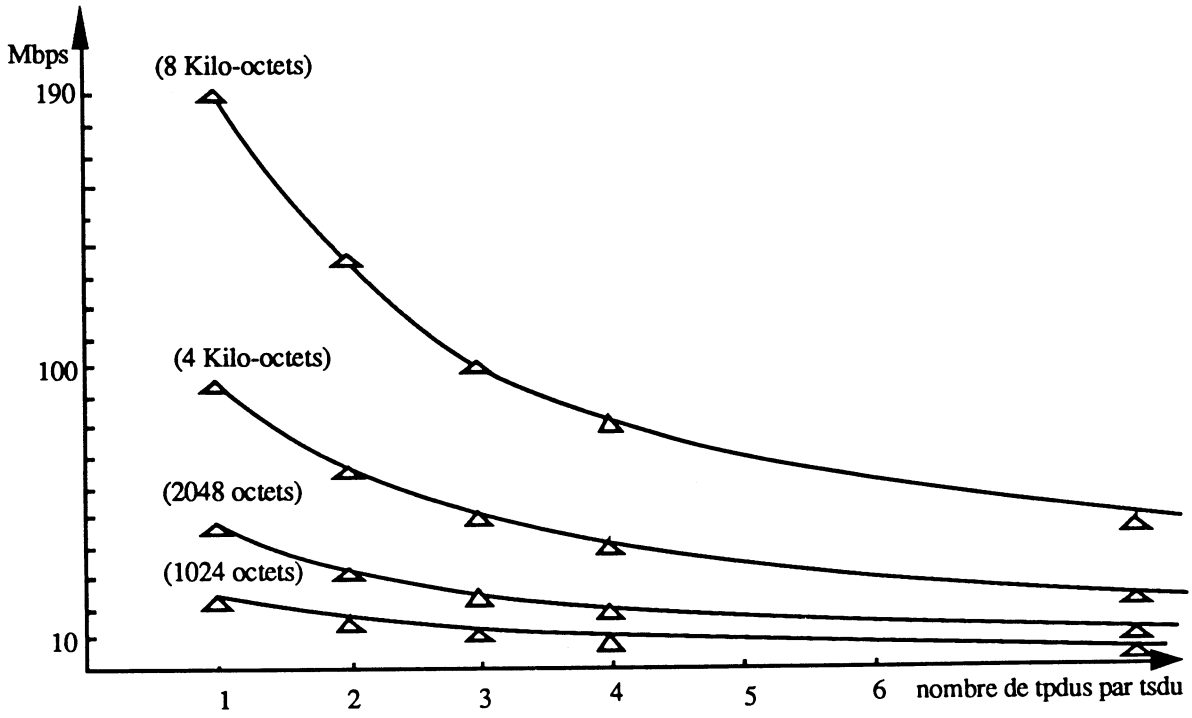


figure 2.1.21; évaluation du coût de la procédure de segmentation
sur la mesure du débit de l'application mesuré pour des TSDUs de taille constante

On y observe une diminution du débit régulière, quelle que soit la taille des TSDU reçues. Cette chutte des performances est de 30% lorsque l'on passe de un à deux segments par TSDU! Et l'on dépasse allègrement les 50 % pour quatre segments par TSDU. C'est un écart suffisamment important pour que l'on apporte le plus grand soin à l'algorithme de la procédure de segmentation.

Ce résultat semble prouver que les effets de la procédure sont à la fois inhérents aux techniques d'implémentation utilisées et au protocole lui-même. Les concepteurs d'XTP prétendent en effet que sur ce protocole, la segmentation ne réduit les performances que d'environ 10 %. Gain en partie dû au format plus simple des TPDU. Pourtant, grâce à l'utilisation des PnPDU, la segmentation d'une t.dt.req est quasi-immédiate et, si elle est effectuée dans les conditions optimales d'allocation mémoire définies précédemment, elle ne nécessite que l'allocation d'un unique PnPDU (qui décrit le nouveau segment). Le coût élevé de la procédure est donc bien dû à la construction de plusieurs en-têtes de TPDU et à l'émission des TPDU respectives.

Comparaison segmentation / réassemblage

La tâche au réassemblage est tout autre et beaucoup plus coûteuse, même si elle n'intervient dans son intégralité qu'à la détection d'une fin de TSDU ou à la réception d'une DT TPDU hors

séquence. La figure 2.1.22 compare le débit du protocole à la segmentation les TSDU (courbe basse de la figure 2.1.19) et au réassemblage des DT TPDU. Afin de comparer plus précisément les performances en émission et en réception, les acquittements sont cette fois comptabilisés (avec une stratégie par TSDU, et une fenêtre de contrôle de flux de taille 8).

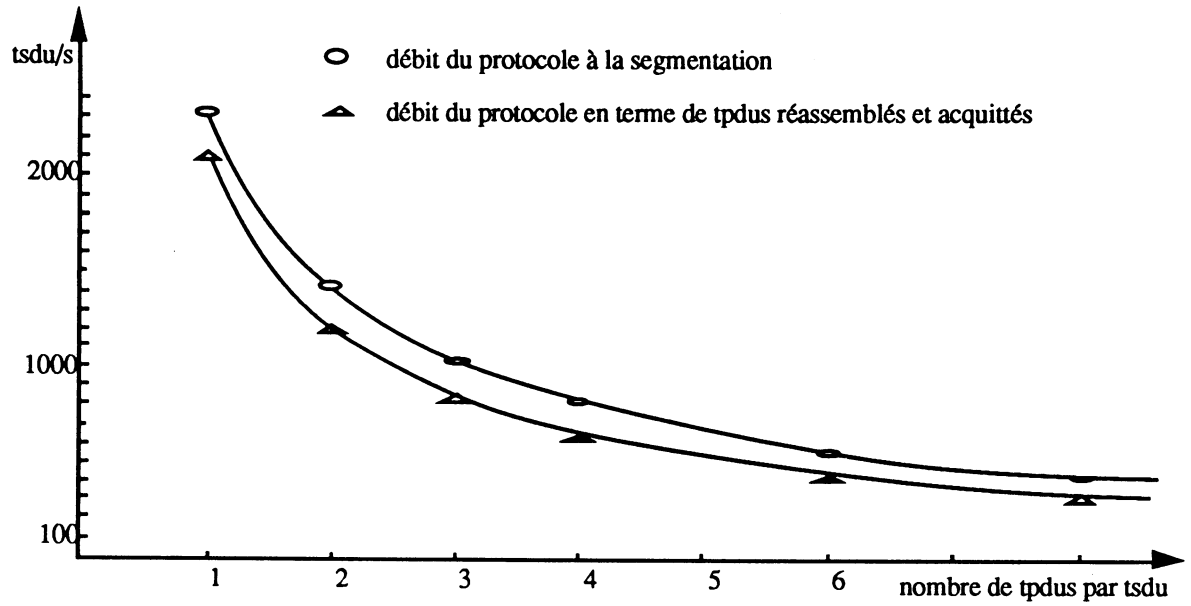


figure 2.1.22; débit du protocole comparé à la segmentation et au réassemblage

Le débit au réassemblage reste systématiquement inférieur au débit à la segmentation. La différence observée est d'environ 20%. Et ce n'est pas l'influence de l'émission des AK TPDU côté entité réceptrice puisque la construction et l'émission de cette TPDU (sans les paramètres de contrôle de flux) ne représente qu'un temps constant inférieur à 80 micro-secondes.

A la réception d'une DT TPDU, l'entité réceptrice doit systématiquement l'insérer à sa place dans la fenêtre de réception, détecter si elle est hors séquence, dupliquée ou en dehors de la fenêtre pour ensuite mettre à jour les variables qui reflètent l'état de la fenêtre. Toujours systématiquement, elle doit chercher si les conditions de réassemblage sont réunies et, le cas échéant, transmettre la TSDU ainsi construite vers l'utilisateur du service Transport local, puis émettre vers l'entité d'origine un acquittement.

Le coût de la procédure de réassemblage est en partie dû à ces tâches, mais aussi à l'analyse laborieuse des DT TPDU reçus. Analyse qui serait à la fois plus simple et plus rapide si le format de tous les TPDU était unifié.

On observe aussi un quasi parallélisme entre les débits en émission et en réception; parallélisme qui se justifie toujours avec les mêmes explications: la partie la plus coûteuse de la procédure

n'est pas la segmentation ou le réassemblage mais bien la construction de DT TPDU à l'émission et l'analyse de ces mêmes TPDU en réception avec des coûts respectifs de 80 µs et 150 µs.

Synthèse locale

L'analyse des mesures atomiques étant terminée, et avant d'exploiter les résultats obtenus pour définir l'architecture de la plate-forme d'évaluation, procédons à une synthèse des résultats.

La figure 2.1.23 servira de base à cette étude de synthèse. Elle établit le débit du protocole de Transport implanté seul sur un Transputer. Ce débit est mesuré pour des TPDU de 1 kilo-octets (taille définie par le profil CNMA) et pour des TPDU de 4 kilo-octets (taille maximale acceptable sur un réseau FDDI). Les conditions d'implémentations sont celles retenues pour la phase de mesures globales (stratégie d'acquittement par tsdu, fenêtre de 8 segments, le protocole restitue lui-même ses ressources, les tampons de données contiennent 1019 octets).

Comparaison avec les mesures globales

Nous avons commencé par comparer le débit optimal que peut atteindre notre implémentation de la couche Transport (figure 2.1.8) et le débit obtenu en mesures globales sur un unique Transputer pour l'application complète (figure 2.1.9). Par un calcul simplifié, nous avons tenté d'évaluer le débit d'un processus sur Transputer en fonction du débit des autres processus partageant le même processeur

$$\tau(p_i) = \alpha(p_i) * \tau!(p_i)$$

avec $\tau!(pi)$ le débit du processus p_i s'il occupait seul le circuit et $\alpha(pi)$ est le taux d'occupation de p_i sur le Transputer tel que $\sum \alpha(p_i) = 1$, pour $1 \leq i \leq n$ (n est le nombre de processus exécutés en parallèle par le Transputer).

Notons que cette équation ne tient pas compte (1) des problèmes de remplissage des files d'attentes et de conflits d'accès aux descripteurs (et autres ressources partagées); (2) des tâches de gestion du parallélisme pour plusieurs processus parallèles sur un même Transputer qui, nous l'avons montré dans le chapitre précédent, ont une influence fâcheuse sur les performances de ces processus. Il s'agit uniquement d'une tentative d'approximation basée sur quelques expériences simples et sur le mécanisme de mesure d'un barycentre.

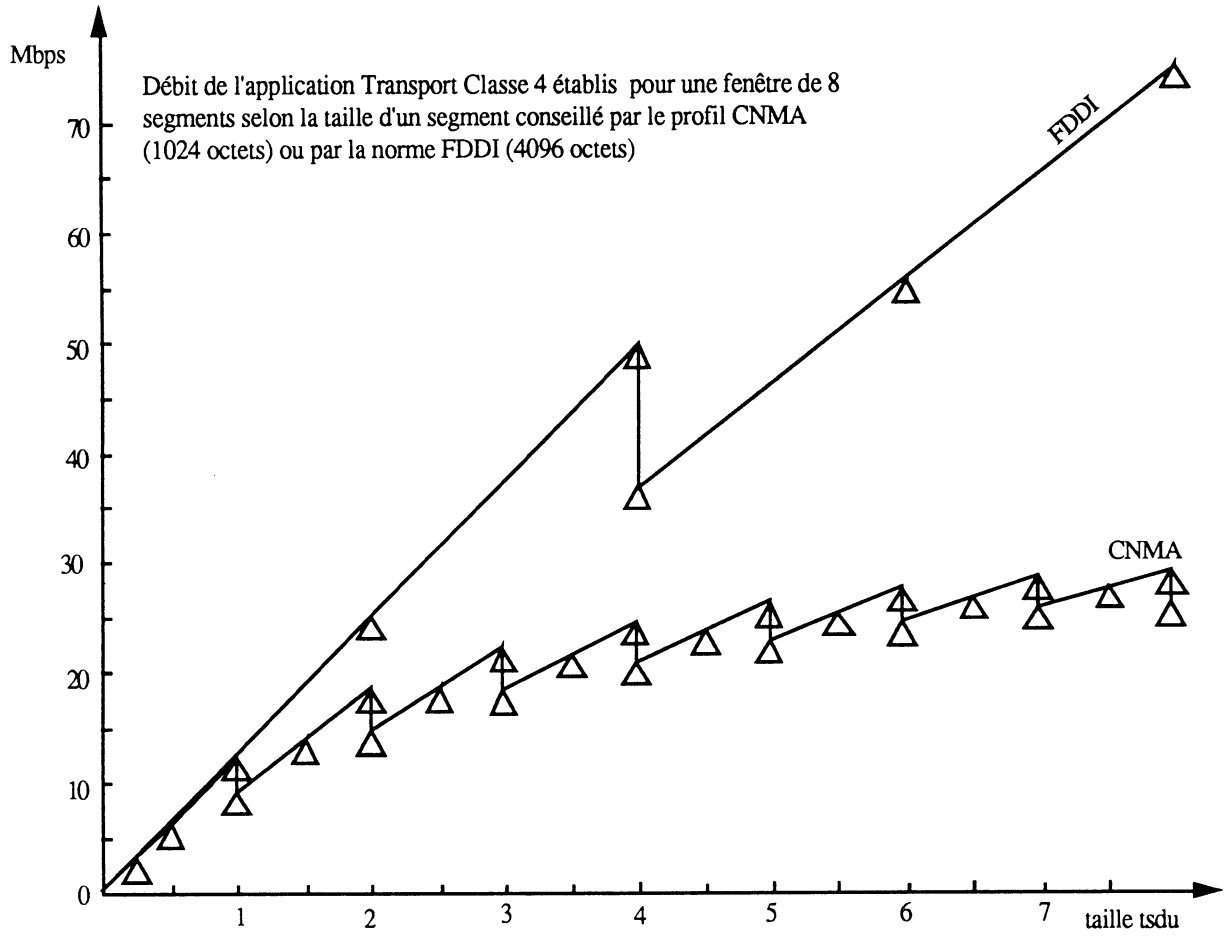


figure 2.1.23; débit du protocole en fonction de la taille des DT TPDU

Une étude pour des TSDU de très grande taille établit un débit maximum en mesures atomiques de 36 Mbps, pour seulement 4 Mbps en mesures globales. Si l'on tient compte uniquement de la présence, lors des mesures globales des processus d'émission réception (dont le débit est borné par 80 Mbps), dans le calcul (selon l'équation établie ci-dessus) du débit réel de l'application, on arrive avec un coefficient α égal à 0.18, à un débit de 6.5 Mbps. Ceci signifie que si l'on sature le système de données, les processus d'entrée/sortie vont à eux deux monopoliser 80% du temps CPU pour ne laisser que 20% au protocole de transport. Ces chiffres sont sans doute faussés par la répartition qu'effectue le Transputer, et surtout réduits par le temps que consomme les tâches systèmes puisque la formule établie n'est qu'une approche mathématique peu précise.

Ces chiffres semblent cependant confirmer les mesures effectuées dans la première section sur le débit de saturation sur un unique Transputer. Ils montrent encore qu'une architecture cible performante devra réserver un processeur indépendant au protocole, ainsi qu'au processus d'interface.

Enfin, dans une situation où l'utilisateur du service Transport saturerait le protocole en entrée, le temps de travail accordé au protocole de Transport serait insuffisant pour absorber le surplus de demandes et la mémoire saturerait bientôt pour, dans un premier temps, perdre des requêtes et ensuite congestionner complètement le système.

Répercussion de la taille des TPDU sur le débit du protocole

La différence de performance existant pour une même implémentation selon qu'elle est implantée sur un réseau classique (Ethernet par exemple) et sur un réseau optique hautes performances tel FDDI est sensible via la taille maximum des TPDU acceptables par le réseau sous-jacent.

On observe (toujours figure 2.1.23), pour des TSDU de 8 koctets, que le débit est jusqu'à 4 fois supérieur au débit dans les conditions définies par le profil CNMA (soit pour des DT TPDU de 1024 octets), et ce sans aucune modification sur le protocole et ses fonctionnalités. Les ruptures de pentes ne sont pas situées aux mêmes valeurs dans les deux cas puisque, l'en-tête de t.dt.req représentant 5 octets, les adjonctions d'un nouveau segment interviendront sur les multiples de 1019 pour des TPDU de 1024 octets et sur les multiples de 4091 pour des TPDU de 4 koctets.

Cette comparaison, ainsi que la totalité des résultats de ce chapitre, prouve que ce n'est pas nécessairement le protocole qui est à mettre en cause pour justifier de mauvaises performances mais que l'environnement, l'implémentation et l'implantation sont les facteurs primordiaux d'un protocole performant.

2.2 Proposition d'Architectures Flexibles pour Implantation Hautes Performances

2.2.1 Phase de spécification

2.2.1.1 Les attributs fondamentaux

Au delà de l'analyse des performances de l'implémentation du protocole de transport OSI classe 4, et de l'étude comportementale des techniques développées à cet effet (modèle de structures de données, noyau système), les deux phases de mesures du chapitre précédent nous ont permis de discuter de l'opportunité des parallélismes dans TP4.

Malheureusement, il s'avère difficile d'analyser le comportement d'une application parallèle sur un unique processeur:

-Le traitement des processus parallèles, tels que géré par le Transputer, interdit toute visibilité sur les listes de processus actifs en priorité haute et basse. Impossible donc de séparer le temps de traitement réel des divers délais introduits par le Transputer (inactif en attente de rendez-vous ou actifs en attente de CPU); et surtout impossibilité de quantifier ces délais.

-Le fonctionnement par mémoire partagée double accès est inexploitable. Chaque processus accède séquentiellement à l'unique mémoire; l'exclusion mutuelle pour l'accès aux descripteurs est alors simulée par passage en priorité haute (plus coûteux qu'un Test&Set câblé). Il y a donc risque de blocage en attente active lors d'un Test&Set.

-Les synchronisations enfin sont simulées sur canal virtuel, par transfert interne en mémoire. Les conditions de synchronisation efficace, définies au chapitre 1, ne peuvent être implantées.

Beaucoup trop de raisons qui rendent impensable l'évaluation précise d'architectures complexes sur un Transputer unique. On se retrouve donc face à la nécessité de définir une machine parallèle, flexible, qui nous permette:

. d'étudier les différents aspects du parallélisme préservés par l'implémentation;

. d'analyser le comportement du noyau système et du modèle de structures de données en environnement parallèle (quantification et qualification des conflits d'accès, Test&Set, occupation CPU);

. de tester comment s'associent synchronisation par rendez-vous et communication par mémoire partagée en environnement parallèle (grâce à l'emploi de mémoires partagées double accès).

. d'exploiter l'environnement mis au point sur d'autres protocoles de niveau trois et quatre.

La machine devra, pour des raisons de flexibilité et d'efficacité, être définie selon les principes et les contraintes suivantes:

-Modularité des éléments qui la composent; de manière à pouvoir distribuer les ressources (processeurs et mémoires) avec rapidité, souplesse et efficacité.

-Les Transputers partagent deux à deux des mémoires double accès. Cette limite est imposée pour des raisons évidentes de distribution du système:

. le Transputer ne gère qu'un bus, données et adresses multiplexées, qu'il faut partager encore avec les mémoires auxquelles il accède;

. une mémoire double accès permet l'accès simultané de deux processeurs via deux bus distincts

Une mémoire centrale sous-utiliserait quant à elle les Transputers (qui devraient s'en partager l'accès) et serait très complexe à gérer (donc coûteux).

-un Test&Set câblé permet, pour chaque mémoire partagée, de réserver l'accès à un Transputer pendant un cycle de lecture et un cycle d'écriture. Le second Transputer cherchant à accéder à la même adresse mémoire est bloqué en attente active (pour une durée d'au pire 4 cycles).

La machine ainsi définie présente l'intérêt d'être complètement distribuée puisque,

. le CPU et le noyau système sont distribués;
. l'espace mémoire est distribué entre les processeurs (chaque mémoire peut donc être associée à un type particulier de descripteur; avec un processeur qui y alloue et l'autre qui restitue, de manière à minimiser les conflits et à simplifier l'exclusion mutuelle);

Notons enfin que, de par l'architecture interne du Transputer, la machine peut être étendue en utilisant les liens physiques; les processeurs rajoutés ne partageant pas de mémoire avec les

ressources de la machine.

Les différents aspects du parallélisme

Nous allons maintenant définir l'architecture de la machine à partir de l'étude des parallélismes possibles, et des associations processeur/processus conséquentes, sur le protocole de Transport implémenté.

-**Parallélisme de type "pipe-line"**. On a clairement défini les éléments qui peuvent fonctionner en "pipe-line", synchronisés par lien série, et qui communiquent par liste ou arborescence partagée (figure 2.2.1):

.les processus d'interface en émission et réception;

.le protocole qui peut lui même être divisé en un processus frontal d'étude de l'événement entrant et d'affectation à une connexion, et en un second processus de traitement de l'événement sur sa connexion.

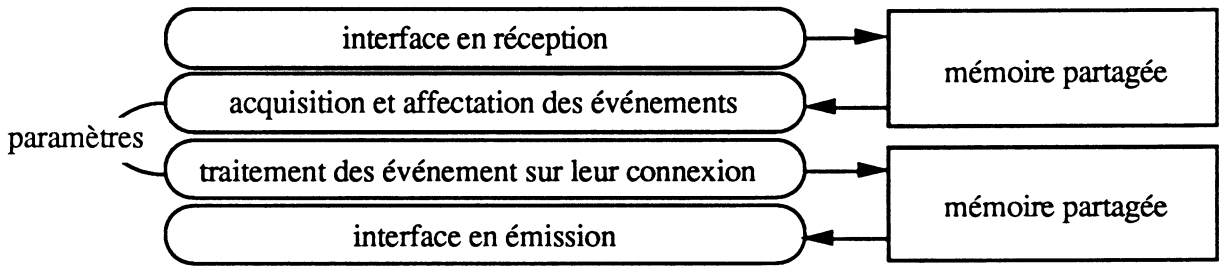


figure 2.2.1; schématisation du parallélisme de type Pipe-line sur TP4

Vu les débits observés lors de l'évaluation, il ne fait aucun doute de la nécessité de séparer physiquement le protocole des processus d'interface.

-On peut alors introduire la notion de **parallélisme de structure** du protocole; qui consisterait à séparer, pour l'interface haute par exemple, processus en émission et en réception.

Cependant, en cas de réception par PnPDU, le débit du processus de réception ne justifie plus l'affectation à un processeur particulier. L'architecture de la machine devra permettre de lui associer éventuellement le processus de réception dual, une partie du protocole qui lui est fonctionnellement proche, ou tout autre processus complètement indépendant et dont le débit et les fonctionnalités autorisent cette association.

A partir de la figure 2.2.1, le parallélisme de structure peut être introduit à divers niveaux.

. traitement des événements sur leur connexion;
. traitement complet des événements reçus de l'hôte et du réseau sous-jacent;
. tout niveau de parallélisme intermédiaire.

-Un dernier type de parallélisme peut être affecté au **traitement des fonctions spécifiques** du protocole. Un processeur peut par exemple être affecté au calcul du checksum, à la gestion des temporisations ou encore à la restitution des ressources.

Interface réseau

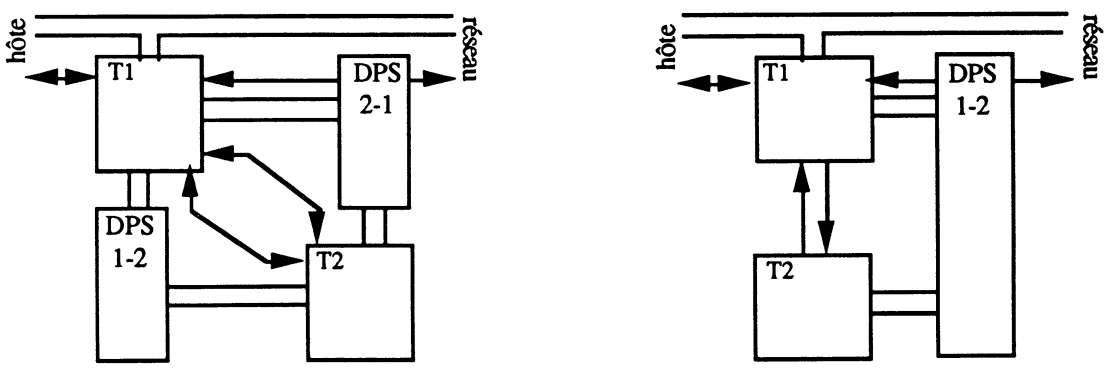
Outre les considérations de parallélisme, la machine devra être capable de s'insérer dans divers environnements réseaux, et en particulier dans un environnement d'évaluation qui nous permette d'en exploiter les résultats avec un maximum de précision. Deux modules seront donc nécessaires:

.interface réseau, afin de nous connecter à un environnement standard (Ethernet, FDDI pour le réseau sous-jacent par exemple), ou a une carte processeur (RISC ou CISC classique) pour l'hôte;

.pour l'espionnage des différentes ressources de la machine.

2.2.1.2 Les architectures retenues

De l'analyse précédente nous avons retenu, pour des architectures utilisant deux et trois Transputers, les configurations, dites de base, présentées figure 2.2.3.



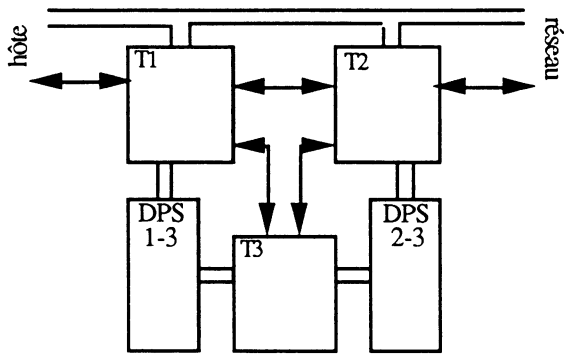


figure 2.2.3.c

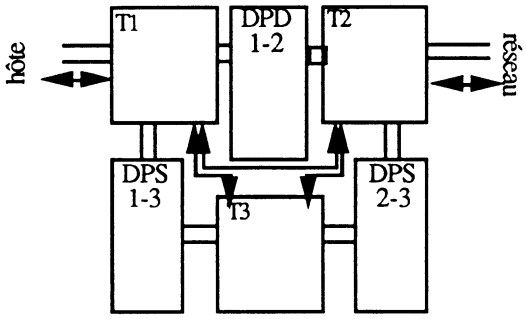


figure 2.2.3.d

figures 2.2.3; architectures de base configurables sur plate-forme d'évaluation de protocoles

Ces quatre configurations permettent d'étudier un éventail d'implémentations assez large, par une exploitation:

Directe: on utilise alors uniquement 2 ou 3 Transputers; soit les configurations de base. L'implanteur décide alors de la répartition physique des Processus. Nous avons, par exemple, envisagé pour le protocole de Transport OSI classe 4 (ces exemples ne constituent pas une liste exhaustive des possibilités d'implantation):

-Configurations 2.2.3.a et 2.2.3.b; T1 reçoit les processus d'interface et T2 le protocole

-Configuration 2.2.3.d; T1 et T2 reçoivent les processus de réception et émission, respectivement haute et basse. T3 reçoit le Protocole. Sur le traitement d'un événement issu de l'utilisateur du service transport, T1 construit un descripteur de demande dans DPS 1-3, puis construit les tampons de données dans DPD 1-2. Pendant cette seconde phase, T3 peut entreprendre le traitement de l'événement décrit dans DPS 1-3 (puisque le descripteur de demande contient les informations suffisantes) et construit ses PnPDU dans DPS 2-3. T2 enfin reconstitue les TPDU à partir des PnPDU contenus dans DPS 2-3 et des données de DPD 1-2.

-Configuration 2.2.3.c; Les deux processus de réception sont implantés sur T1, les deux processus d'émission sur T2; T3 reçoit le protocole. DPS 1-3 reçoit les descripteurs de demandes, DPS 2-3 les PnPDU. Ainsi, T1 construit la liste de demande en décrivant les données utiles directement par PnPDU dans la mémoire hôte (ou la mémoire MAC), T3 traite l'événement et génére des PnPDU dans DPS 2-3. Quitte à T2 d'émettre, à partir des données contenues dans la mémoire hôte (ou la mémoire MAC), et de l'arborescence contenue dans DPS 2-3.

Dupliquée: on duplique les configurations de base pour exploiter le parallélisme de structure. On peut par exemple utiliser deux fois la configuration 2.2.3.c ou 2.2.3.d, chacune d'elle étant affectée à un sens de transfert (hôte vers réseau et réseau vers hôte). La synchronisation sera réalisée via T3 pour empêcher la modification simultanée d'un contexte de connexion.

Etendue: on associe à certains Transputers d'autres Transputers connectés exclusivement par canal physique (soit sans partage de mémoire). Prévu pour permettre à T3 d'être assisté dans la gestion des temporisations; ou selon la figure 2.2.4, permettre le traitement successif de 2 couches (T30 remplace T3 et gère l'accès aux deux mémoires, T31 implante le premier protocole et T32 le second).

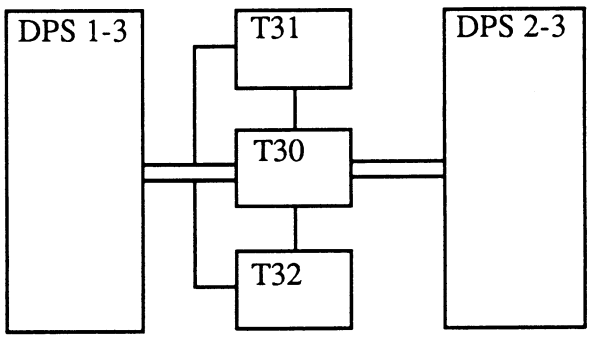


figure 2.2.4; exemple d'extension possible de la plate-forme d'évaluation

Quelle que soit la configuration étudiée, l'exclusion mutuelle pour l'accès aux mémoires partagées est en premier lieu préservée par la structure de l'implémentation, et l'implantation physique des processus:

. on associe à chaque mémoire double accès un processus producteur et un consommateur;

. Les deux processus qui partagent une ressource n'ont pas d'accès simultané sur une même ressource.

Ainsi le Test&Set, susceptible de provoquer un blocage en attente active d'un Transputer, n'est qu'une garantie redondante en cas d'accès simultané non prévu, ou en prévision de l'implémentation d'autres protocoles pour lesquels on ne pourrait préserver naturellement l'exclusion mutuelle.

2.2.2 Conception d'une plate-forme d'évaluation

Des spécifications de la section précédente est née une plate-forme d'évaluation de protocoles configurable, à base de Transputers et de mémoire partagée.

Sous l'appellation *"Protocols Engine on Transputers" (PET),* la société APTOR a conçu, développé et mis au point la dite plate-forme. Les phases de conception sont présentées en détail dans le "dossier technique de la machine PET" [Savard 90]. Nous nous contenterons d'étudier, pour les expliciter, les points intéressants, ainsi que les problèmes pressentis.

Les contraintes

Les problèmes majeurs à résoudre dans la phase de conception d'une telle machine sont:

.la flexibilité de la plate-forme
.l'instruction de Test&Set câblé
.les facilités d'espionnage de la machine

auxquels s'ajoutent quantité d'autres problèmes techniques liés au partage de mémoire et au timing de la machine.

Les modules de base

Ces contraintes nous ont conduit à concevoir une plate-forme modulaire. Cinq modules de base ont ainsi été définis:

-Un module Transputer contient un processeur, de la mémoire locale et deux connecteurs de bus.

-Un module mémoire dynamique double accès (DP D RAM); 4 Méga-octets de mémoire dynamique standard accessibles par deux bus distincts (soit depuis deux modules Transputers), gérés par logique externe.

-Un module mémoire statique double accès qui utilise de véritables modules double accès de 64 Kilo-octets qui permettent l'accès simultané par deux bus, sans logique d'exclusion. Chaque module DP S RAM contient 256 Kilo-octets de mémoire, ainsi que le sélecteur de Test&Set.

-Un module interface réseau; situé aux interfaces de la machine, ce module permet d'insérer la machine dans un environnement réseau réel, aussi bien que dans notre environnement d'évaluation.

-Le module espion regroupe un ensemble de compteurs et de logique câblée; il se connecte à chaque module pour y acquérir des informations préalablement définies.
Les avantages de ce type de conception sont multiples, avec en particulier:

-Simplification de la mise au point. Chaque module peut être testé séparément, puis assemblé par étapes.

-Réutilisation possible des modules pour une extension de la machine ou pour une toute autre utilisation.

-Conception de la plate-forme indépendante de la conception des modules.

Mise en oeuvre

L'association des divers modules, pour former une plate-forme configurable d'évaluation de protocoles, est réalisée par un fond de panier local et spécifique (figure 2.2.5). La structure du fond de panier permet de configurer simplement les architectures représentées sur les figures 2.2.3.a à 2.2.3.d de la section précédente.

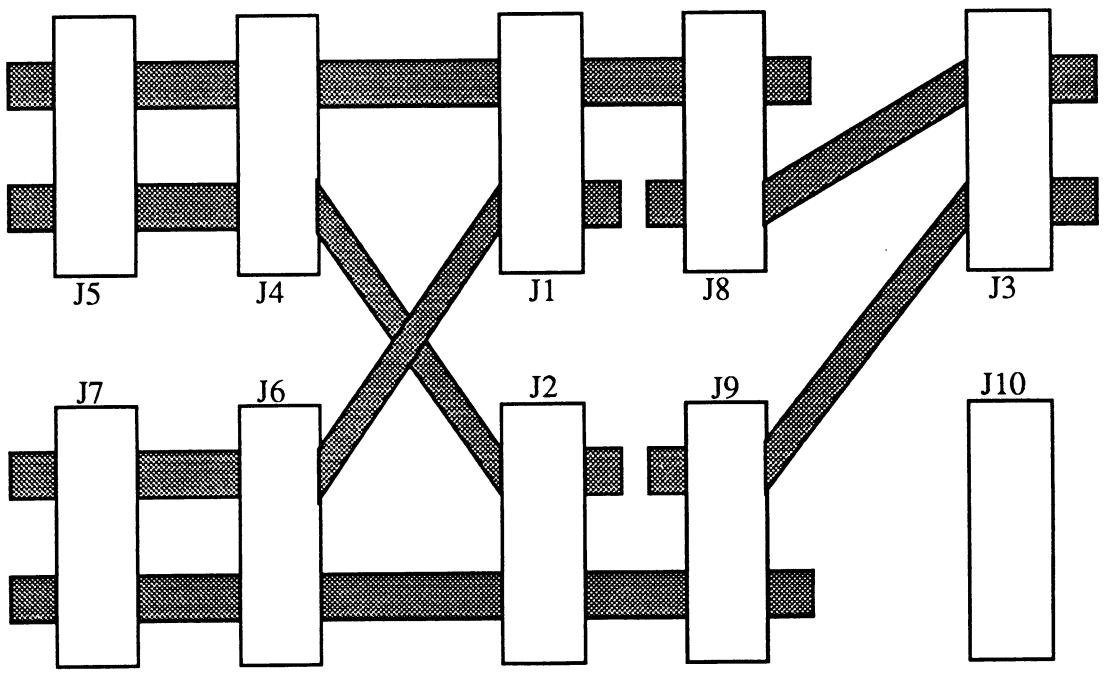


figure 2.2.5; structure du fond de panier de la plate-forme d'évaluation

2.2.3.d par exemple est réalisée en plaçant les modules de la manière suivante:

.T1 en J1, T2 en J2 et T3 en J3;
.DPS 1-3 en J8, DPS 2-3 en J9;
.DPD 1-2 en J4.

J5 et J7 sont systématiquement réservés aux modules d'interface réseau et J10 au module espion.

Une telle structure garantit une grande modularité; l'utilisation de ces modules pour une autre application nécessite uniquement la conception d'un nouveau fond de panier pour un coût réduit.

Nous ne nous engagerons pas plus loin dans la description technique de la machine PET, si ce n'est pour apporter quelques précisions sur le fonctionnement du mécanisme de Test&Set câblé et sur les capacités du module espion.

-Un Test&Set câblé a été implanté sur la DP S RAM, pour parer aux manquements du Transputer en terme de gestion de l'exclusion mutuelle. Le principe est de protéger l'accès à une adresse mémoire, pour le processeur qui exécute le Test&Set, pendant au moins un cycle de lecture et d'écriture consécutives.

Pour ce faire, le processeur concerné adresse un port réservé à cet effet, puis place immédiatement sur son bus l'adresse de l'emplacement mémoire à protéger. Le sélecteur de Test&Set câblé protège, dès qu'il a identifié la requête de Test&Set, l'accès à l'emplacement mémoire sélectionné contre tout accès (direct ou par Test&Set) par le Transputer concurrent. Pour sortir de l'instruction Test&Set, le Transputer qui l'a validée doit adresser à nouveau le même port.

Si le second processeur cherche à accéder à un emplacement mémoire alors que ce dernier est protégé par Test&Set, il se retrouve bloqué par le sélecteur en attente active (qui lui envoie des "Wait-State" jusqu'à libération de l'emplacement; soit pour une durée réduite).

-La carte espion enfin doit permettre à l'utilisateur de connaître des informations générales sur la machine qu'il était impossible d'obtenir sur un unique Transputer. Elle est conçue pour analyser un module à la fois, pour y acquérir les information suivantes:

.nombre d'accès en mémoire partagée de part et d'autre d'une mémoire double accès;
.nombre de conflits d'accès à ces mêmes mémoires;

.nombre de Test&Set, ainsi que nombre de conflits (de demande et d'accès);
.temps passé à gérer les ressources, par type de ressource et par zone mémoire;
.taux d'occupation de chaque Transputer.

Soit de recueillir un ensemble d'informations qui nous permettront d'acquérir une connaissance plus précise de l'implémentation, et d'optimiser à la fois l'implémentation elle même et son implantation sur architecture multi-processeurs.

Chapitre 3: SYNTHESE ET RECHERCHES FUTURES

3 SYNTHESE ET RECHERCHES FUTURES

Synthèse qui va s'efforcer de répondre à deux questions autour desquelles ont été organisés ces travaux sur l'implémentation du protocole de Transport OSI Classe 4.

-De l'avenir de TP4 (ou de tout autre protocole de la même famille) devant la nouvelle génération des protocoles temps réels hautes performances; de son aptitude à s'adapter aux applications nouvelles (temps réel, implantation VLSI).

-Transputer et Occam; leur contribution à l'implémentation haute performance des protocoles de communication de niveau Transport ou Transfert.

Puis ensuite d'analyser comment poursuivre ces recherches vers les implémentations Giga-bits des protocoles de niveau intermédiaires.

3.1 TP4, la Force Tranquille.

Nous l'avons montré, des performances voisines et même nettement supérieures à 100 Méga-bits par seconde sont possibles avec une implémentation soignée du protocole de Transport OSI. Par implémentation soignée nous entendons respecter un ensemble de règles que nous avons tenté de mettre au point durant cette étude et qui concernent

. le choix avisé des paramètres fonctionnels et des fonctionnalités de la couche implémentée;

. l'exploitation des latitudes laissées par la norme à l'implémenteur (algorithmes, structures de données);

. l'utilisation de structures de données spécialisées qui tendent à minimiser les transferts de données en mémoire, associé à un noyau de primitives systèmes dédié rapide;

. évolution dans un environnement souple, et aussi proche du niveau matériel que possible; et, grâce à une connaissance parfaite de l'environnement, utiliser les facilités offertes pour en optimiser le rendement.

. conception d'un noyau système qui minimise l'intermédiaire entre le protocole et l'environnement hôte;

. implantation du protocole sur une architecture vraiment parallèle, où les divers processus communiquent à la fois par lien série et par ressources partagées.

Ainsi les fonctionnalités du protocole ne sont pas toutes à mettre en cause ou à supprimer pour justifier les mauvais résultats d'une implémentation. S'il est vrai que certains services offerts par TP4 sont trop complexes à implémenter ou tout simplement inutiles ou redondants, c'est souvent l'implémentation réalisée qui est à mettre en cause. L'influence de l'implémentation intervient à deux niveaux:

Fonctionnalités et performances

Au niveau des fonctionnalités donc, certaines d'entre elles sont reconnues pour être très coûteuses. Le choix de l'algorithme, et aussi de la complexité de la procédure doit être étudié en fonction des caractéristiques du langage de développement et de l'environnement hôte, ainsi qu'en fonction des fonctionnalités des autres couches qui composent le système.

Reprenons la liste des fonctions transport typiques établie dans le chapitre 2 pour analyser, à la lumière des mesures effectuées, dans quelles conditions elles sont optimales, et proposer ainsi une comparaison technique de TP4 aux autres protocoles.

-La gestion de connexion n'est pas à mettre en cause; un protocole digne de ce nom doit fournir un service en mode connexion.

On peut par contre remettre en cause la nécessité des données express qui, de par leur fonctionnement et leur utilité, peuvent être remplacées avantageusement par un service datagramme qui resterait utilisé, comme les données express, pour transférer en priorité des TSDU express de taille réduite.

-L'ouverture et la fermeture de connexions sont relativement lourde par poignée de main. On retiendra ce mécanisme parce que plus fiable et plus souple; à condition bien sûr que les connexions ne soient pas créées anarchiquement pour le transfert d'une quantité réduite d'information.

Le mécanisme de poignée de main doit minimiser le nombre d'échanges nécessaires à l'ouverture et à la fermeture de la connexion. TCP est donc plus performant que TP4 et Datakit dans le domaine; XTP associe une ouverture de connexion implicite, c'est à dire provoquée par la première TPDU de données, à une fermeture par poignée de main.

-Dans une TPDU, la partie données et la partie contrôle doivent être définies formellement afin de simplifier l'analyse des TPDU. Dans XTP, tous les champs de la TPDU ont un format fixe.

-La qualité de service doit être utilisée pour configurer le protocole de Transport en fonction du type d'application traité. Les paramètres définis par TP4 sont généralement trop complexes et trop abstraits, ou tout simplement difficiles à contrôler.

-Le démultiplexage des TPDU est rapide quand les TPDU ont un format prédéfini, et surtout quand le format de l'en-tête est connu à l'avance et unique pour toutes les TPDU. L'affectation à une connexion est alors immédiate.

-La stratégie d'acquittement est loin d'être optimale sur TP4. Mais dans la mesure où l'on doit transférer des acquittements sur le réseau, existe-t-il des stratégies efficaces? La stratégie d'acquittement doit être considérée depuis le contrôle de flux. Ainsi, que cette stratégie soit basée sur l'émetteur ou le récepteur nous semble peu important. Par contre, le fait d'associer à un contrôle de flux par acquittement un contrôle de débit est à considérer comme un atout pour le protocole. Les acquittements ne doivent être alors utilisés qu'occasionnellement, pour contrôler que tout ce passe correctement sur le réseau. Ils doivent de surcroît fournir des informations suffisamment précises pour permettre une retransmission des informations perdues seulement.

Une fréquence élevée d'acquittement n'est en effet requise, dans les protocoles qui ne font pas de retransmission sélective, que pour minimiser les retransmissions inutiles.

Encore une fois, TP4 est assez peu performant dans ce domaine, tout comme TCP d'ailleurs. Seuls NETBLT, VMTP et surtout XTP offrent un système de contrôle de flux complet et performant (XTP dispose de contrôle par fenêtre, de contrôle de débit, et de la retransmission sélective).

-Pour terminer, le discours sur traitement des erreurs est un peu redondant avec le précédent.

TP4 qui utilise AK, DR et ER TPDU pour signaler les erreurs n'est pas très optimal dans ce domaine. CNMA propose d'ailleurs de ne pas utiliser les ER TPDU mais uniquement des DR TPDU. Un format unique est suffisant pour toutes les TPDU de contrôle, nous l'avons dit maintes fois.

Le checksum, autre moyen de détecter les erreurs dans une trame, est plus coûteux sur TP4 (parce que transmis en en-tête) que sur XTP (transmis en queue). De plus, la présence d'un

checksum au niveau Transport n'est pas indispensable, des checksums étant déjà définis aux niveaux inférieurs. Ainsi le checksum est optionnel dans tous les protocoles que nous avons analysé.

Enfin les erreurs sont plus difficiles à détecter dans des TPDU au format fantaisiste que dans des TPDU au format défini de manière unique et formelle.

Cependant, si l'on a montré que TP4 pouvait être performant quand il était correctement implémenté, ses caractéristiques et fonctionnalités en rendent l'implémentation scabreuse et surtout très contraignante. Le rendre compétitif avec les nouveaux protocoles hautes performances, ou en faire une version implémentable sur VLSI nécessiterait d'y apporter quelques modifications:

. minimiser les types de TPDU différents, et surtout, choisir pour tous ces TPDU un format d'en-tête commun. Ceci afin de rendre plus rapide l'analyse de l'en-tête, l'affectation à une connexion et la segmentation, et aussi de faciliter l'emploi de techniques comme la prédiction d'en-tête ou la gestion de ressources statique;

. coller les informations utiles (ainsi que tous les champs de l'en-tête) sur 4 octets. L'information élémentaire dans TP4 est l'octet; or les processeurs actuels disposent tous de chemins de données de 32 bits;

. alléger la procédure de contrôle de flux et lui associer un contrôle de débit. La conséquence de ces modifications est directement mesurable sur le format des acquittement qui devient plus simple et surtout sur la fréquence de ces acquittements;

Le modèle OSI

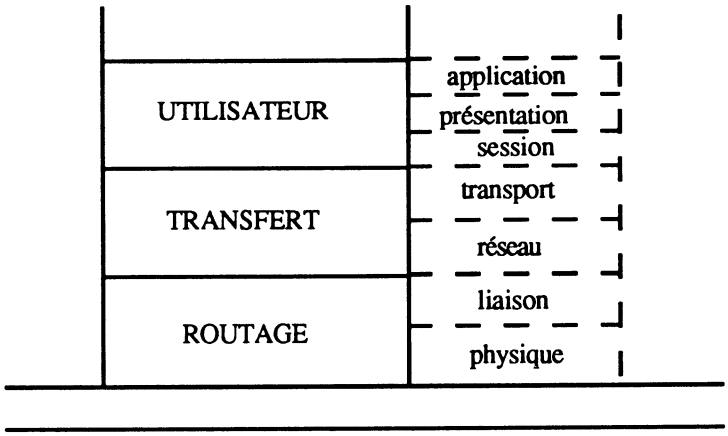
L'implémentation de tout ou partie de la pile OSI introduit des problèmes qui n'existaient pas à l'implémentation d'une couche seule et qui concernent la gestion de la mémoire, le traitement "en ligne" des couches sans transfert de données, le partage de ressources et les algorithmes de restitution des ressources. L'implémentation de plusieurs couches doit

. s'efforcer de supprimer les goulots d'étranglement en créant une continuité entre les couches implémentées via des structures de données spécialisées.

. supprimer les redondances entre les fonctionnalités proposées par chaque couche. Mais cette adaptation est plus liée au bon sens de l'implémenteur (qui doit lui même

sélectionner les fonctionnalités dont il a besoin en fonction de ce qui lui est fourni par les autres couches) qu'à une modification à apporter aux spécifications des couches;

. le service défini entre chaque couche est pénalisant pour les performances; on y préférera une implémentation en ligne telle qu'elle existe sur XTP par exemple entre Transport et Réseau. Ceci afin d'optimiser les transfert d'informations et les multiples synchronisations entre couches. On rappellera le modèle en 3 couches défini par [GAM-T 87] pour les réseaux temps réels militaires:



ROUTAGE regroupe les couches 1 et 2; LLC étant laissé à l'appréciation de l'utilisateur.
TRANSFERT englobe 3 et 4; le niveau de complexité de 3 étant lié à l'étendue et à la vocation du réseau (des classes peuvent donc être définies).
UTILISATEUR interface directement l'application sur les deux couches du réseau; cette couche doit définir plusieurs services pour s'adapter à diverses applications; et surtout pour configurer le réseau sous-jacent aux contraintes de l'application.

Le schéma en sept couches est donc trop lourd, et même trop puissant pour n'importe quelle application classique. Mais il ne faut pas perdre de vue que TP4 à été défini il y a 10 ans déjà, comme un standard, et qu'ainsi il est obligé de regrouper l'état de l'art au niveau des fonctionnalités de niveau 4, et surtout d'être portable, même si avec l'évolution des besoins, une grande partie des services définis par TP4 se sont révélés inutiles ou même inadaptés.

TP4 en particulier (et la pile OSI en général)n'a pas été conçu pour être performant mais pour servir de base commune à des implémentations cohérentes des protocoles de communications.

Dans un premier temps la démarche consistant à définir des profils était satisfaisante. Devant les besoins temps réels et hautes performances auxquels nous sommes confrontés, TP4 n'est pas

toujours suffisant; il reste par contre que les leçons retirées de l'étude de TP4 peuvent être utilisées

. pour concevoir de nouveaux protocoles; cette solution peut dans certain cas s'avérer plus intéressante que d'essayer d'adapter un protocole général à des besoins pour lesquels il n'est pas adapté.

. mettre au point des techniques d'implémentation et circuits de communications dédiés pour utiliser, avec des performances élevées, les protocoles existant.

Ainsi si une implémentation VLSI est visée, nous pensons plus astucieux de concevoir un nouveau protocole parallèlement à l'architecture du circuit hôte. Les contraintes imposées par un environnement de ce type font que TP4 sera particulièrement inadapté et que le modifier serait alors une tâche considérable; on en conserverait que certaines fonctionnalités pour modifier complètement toute la structure du protocole et surtout les formats et types des TPDU.

Pour une application temps réel, qui requiert des fonctionnalités supplémentaires (on pense à l'adressage de plusieurs destinataires, au traitement de TSDU plus courtes, à la sécurité et surtout au respect des délais) et doit supporter des contraintes de temps de réponse beaucoup plus sévères, adapter TP4 serait encore une entreprise difficile [Minet 89], incertaine. Les modifications à entreprendre seraient du même ordre que pour une implémentation VLSI; les protocoles temps réels étant de plus en plus, d'ailleurs, directement associés à un circuit qui les implémente.

Enfin pour effectuer du transfert de fichier rapidement entre deux machines ou systèmes, TP4 implémenté soigneusement peut suffire; moyennant un choix approprié des fonctionnalités et d'interface avec l'environnement hôte. Du point de vue du coût, cette solution reste plus avantageuse puisqu'on est à même d'atteindre des performances élevées avec un protocole classique, standardisé. Mais, en utilisant des techniques d'implémentation différentes que celles étudiées dans ce document, TP4 doit permettre d'atteindre, pour une application temps réelle, des performances élevées. C'est un protocole général qui, s'il est bien implémenté offre à un éventuel utilisateur tout ce que l'on peut demander à un tel protocole.

Pour conclure temporairement cette discussion sur les protocoles de nouvelle génération en général, on est à même après cette étude d'analyser les ingrédients de base à fournir pour atteindre de hautes performances avec une sécurité maximale et des temps de réponse minimaux:

-Parallélisme massif, que ce soit au sein d'un circuit ASIC spécialement conçu ou sur une architecture construite à partir de circuits spécialisés.

-Conception du protocole et de son circuit hôte en parallèle; ou tenir compte à la conception du protocole des contraintes et avantages qu'apporte l'implémentation sur circuit VLSI.

-Définir les fonctionnalités de manières à minimiser les accès à l'operating system (lui-même rapide et simple) et étudier un protocole qui minimise les TPDU de contrôle et les transferts de données.

Globalement, un protocole de niveau quatre qui se veut performant doit être plus *«léger»*, et plus *«carré»* que TP4.

3.2 Transputer et Occam, environnement à risque

La philosophie du Transputer est nouvelle dans le domaine des micro-processeurs qui correspond à une évolution sensible vers les préoccupations liées à la communication et à la conception d'architectures parallèles. Mais ces considérations sont encore trop générales pour faire du Transputer l'outil idéal pour l'implémentation de protocoles de communication.

Le Transputer n'en reste pas moins l'outil le mieux adapté. Ce qu'il a prouvé, que ce soit par les performances atteintes (en supprimant du circuit les processus de gestion de l'utilisateur du service Transport et du noyau réseau on pense être en mesure d'atteindre les 10 Méga-bits par seconde sur un unique processeur) ou dans la phase de conception d'une plate-forme d'évaluation configurable.

L'utilisation d'un environnement commercial pour implanter un protocole standard nous a contraint à adapter mutuellement l'application à son environnement, tâche simplifiée par les fonctionnalités du Transputer et les structures de données dédiées.

Ces travaux nous ont amené à mettre en évidence un certain nombre de défauts de conception, ou d'absences délibérées, dont il aurait été souhaitable de disposer dans l'environnement d'implantation pour développer une application de type TP4 (et son operating system) dans de bonnes conditions:

-Le Transputer ne gère que deux niveaux de priorité. On aurait aimé pouvoir hiérarchiser de manière plus complexe les processus puisque cohabitent sur le même processeur:

.l'application ,
.son environnement de test (qui regroupe les processus les moins prioritaires),
.l'operating system (qui ne doit en aucun cas être commuté quand il effectue une tâche).

De plus, il est impossible de passer simplement et rapidement d'une priorité à l'autre et il est même impossible de passer directement de la priorité haute à la basse en cours de traitement d'un processus qui a été déclaré en priorité haute.

-L'instruction Test&Set n'existe pas sur le Transputer et le circuit n'offre aucun moyen de le simuler dans une architecture multi-processeurs communicant par mémoire partagée.

Le Transputer a été conçu pour faire de la communication par lien, ce qui pourrait justifier l'absence d'instruction de type Test&Set; mais nous avons montré que, pour garantir des performances élevées, il fallait minimiser les transferts de données, et que le seul moyen d'y arriver était de communiquer par mémoire partagée, au risque d'aller à l'encontre de la philosophie du Transputer.

-Autre handicap du circuit lié à son architecture interne: Le Transputer ne possède pas d'outil de débogue digne de ce nom. Vu la visibilité externe de l'état du Transputer à n'importe quel moment de son exécution, nous sommes conscient qu'il est difficile d'écrire un outil de déverminage, et ceci d'autant plus que les applications à mettre au point font usage du parallélisme et de la communication par canal.

Ainsi le développeur n'a aucun moyen de faire à un instant donné une photographie de l'état interne du Transputer, ni même d'arrêter une exécution et de la relancer. D'autres informations qui auraient été nécessaires à l'analyse du protocole de transport sont impossibles à obtenir sur le Transputer:

. temps total de traitement d'un processus (sans comptabiliser les temps d'attente);
. nombre d'accès mémoires effectués par le Transputer sur sa mémoire interne ou externe;
. état des canaux;
. état des différents processus et position dans la liste (le Transputer ne nomme pas les processus qu'il exécute et toute surveillance est impossible).

-Il est impossible de protéger un processus contre une commutation de processus; soit qu'un processus en cours d'exécution puisse conserver le CPU consécutivement à une instruction de communication.

-Comme le permet ADA, on aimerait pouvoir écrire sur un canal depuis plusieurs processus; le processus qui lit le canal étant quant à lui unique. Ceci éviterait d'avoir à utiliser un multiplexage coûteux dans des situations dans lesquelles la cohérence des informations transmises ne le nécessite pas.

La future gamme de Transputers est donc attendue avec impatience. INMOS annonce pour bientôt le Transputer H1, machine RISC 32 bits qui triple les performances du T 800:

.8 liens à 100 Méga-bits par seconde
.horloge interne à 50 MHz
.accès en mémoires externe possibles en 40 ns
.32 Koctets de mémoire locale accessibles en 1 cycle

Le TH1 posséderait en plus un moyen de faire de la communication par mémoire partagée!

Parallèlement à ces remarques concernant directement le Transputer, nous avons observé un certain nombre de faiblesses propres à Occam qui ne sont pas toujours justifiées par l'architecture et les caractéristiques du Transputer:

-Les structures de données du langage Occam sont inadaptées à un langage de ce niveau. Ainsi un certain nombre de types de base parmi les plus importants sont inexistants. Le type enregistrement et le type pointeur au sens de Pascal n'existent pas. La manipulation de liste chaînée, fréquemment utilisée pour traiter les listes de PnPDU et autres descripteurs de demandes devient une tâche coûteuse et lourde. Le traitement de ces mêmes descripteurs (de l'initialisation à la mise à jour ou à l'étude) aurait été facilité par l'existence d'un type enregistrement.

La seule alternative proposée par Occam est l'instruction RETYPES qui est moins simple à utiliser, qui nécessite plus d'instructions et qui est donc beaucoup plus lente. L'absence enfin de types construits en Occam rend la manipulation des descripteurs pénible.

-La récursivité n'est pas autorisée par Occam (par analogie au modèle de référence, CSP). Si l'on peut comprendre que la gestion de processus récursifs parallèles est un problème complexe, cela ne justifie en rien l'absence de récursivité au sein des procédures simples (qui ne contiennent pas de passages en priorité haute, de construction alternative ou de sous procédure parallèle).

-Il est impossible de passer à une fonction Occam des paramètres par variable. Occam n'accepte

que des paramètres passés par valeur, et seul le résultat peut être affecté. Il s'agit là d'un manque de souplesse considérable qui rend l'utilisation des fonctions impossibles dans bon nombre de situations.

-La possibilité d'insérer directement à la compilation le code d'une procédure dans le code du processus qui l'appelle permet de supprimer à la fois l'appel de procédure et le passage de paramètres. Pour un code produit plus gros, cela permet de gagner jusqu'à cinquante pour-cent sur les performances d'un processus ainsi programmé. Cette facilité est prévue par Pascal au niveau du compilateur (option in line) et par ADA directement dans la définition du langage (pragma in line).

-L'indentation et le vocabulaire réservé en majuscule dénote d'un esthétisme certain! Rappelons que le système de développement du Transputer ne possède même pas d'indenteur automatique.

-Le rattrapage d'erreurs à la mode ADA est une facilité intéressante, surtout avec tous les cas d'erreurs qui sont imposés par le protocole de Transport.

-Impossible de connaître l'état d'un canal: savoir qui écrit sur ce canal, si le vis à vis est déjà en attente sur le canal, s'il reste des informations sur le canal après lecture par le récepteur. La connaissance de ces informations directement au niveau du processus Occam permettrait d'entreprendre des traitements plus efficaces, de rattraper certaines erreurs de traitement et surtout de rendre les synchronisations plus fluides pour des applications dans le domaine des protocoles de communication (cette limite est une conséquence de l'architecture RISC du Transputer).
-Toujours dans le même esprit, Occam n'offre pas la possibilité de pouvoir supprimer, bloquer ou suspendre un processus depuis un autre. Il est d'ailleurs impossible d'avoir, depuis un processus en cours d'exécution, des informations sur l'état des autres processus.

-Le Transputer n'ayant pas été conçu pour fonctionner en mémoire partagée, il est impossible, en Occam, de faire de la communication non blocante (que ce soit par canal ou directement par mémoire). Cette facilité est à rapprocher de la possibilité (évoquée dans le paragraphe précédent) de communiquer sans être commuté.

Toutes ces remarques concernent des facilités dont on a ressenti le besoin à un moment ou à un autre de la programmation en Occam du protocole de Transport. Elles nous auraient certainement simplifié le travail de programmation, mais rien ne nous permet d'affirmer que l'implémentation obtenue aurait été plus performante.

On doit rappeler que le Transputer est une machine RISC et qu'à ce titre on ne peut pas lui réclamer les mêmes capacités de contrôle qu'un microprocesseur CISC, ni des instructions de trop haut niveau. Il en va de même pour Occam qui reste un langage évolué de bas niveau, comparé à ADA, autre langage parallèle qui n'a pas été conçu pour faire du parallélisme réel mais pour simuler ce parallélisme sur un unique processeur. Ainsi ADA paye sa complexité et les facilités qu'il propose par une moins grande souplesse d'adaptation dans les applications parallèles.

Il faut donc respecter le compromis que représentent Occam et le Transputer, même si certaines absences peuvent être qualifiées d'impardonnables: Test&Set, construction de type et récursivité.

Il semblerait cependant que le salut puisse venir du langage C. Plusieurs compilateurs C existent déjà pour le Transputer (que ce soit des produits commerciaux [Logical 89] ou universitaires [Favre 90]).

Les plus évolués généreraient un code plus performant que le compilateur Occam d'INMOS, et surtout [Favre 90] permet de contrôler les processus par sémaphore, soit en évitant, pour la synchronisation interne sur un Transputer, la communication par lien et les effets de bords qu'elle provoque.

Avantages qui ne sauront nous laisser indifférents; d'autant plus que C est un langage globalement plus puissant qu'Occam (en particulier au niveau des structures de données et de la gestion des pointeurs); même si c'est un langage moins rigoureux.

Ainsi le Transputer utilisé seul ne nous a pas permis d'aller aussi loin que nous l'aurions voulu dans l'étude d'un protocole de niveau quatre implémenté en Occam. Si l'on peut dire que la gestion du parallélisme intégrée, la présence de liens série et la mémoire locale accessible en 1µs sont des atouts pour l'implémentation de ce type de protocole, on n'a toujours pas pu mesurer si le Transputer, associé aux structures de données dédiées, permettait de réduire le pourcentage de temps passé à effectuer des tâches systèmes (et s'il renversait les immuables 80/20). Par contre, on a pu montrer que le faux parallélisme était dangereux, la communication par canal coûteuse, les liens séries trop lents et la mémoire locale trop petite.

Attendons le verdict de la plate-forme de test pour apporter des conclusions plus argumentées sur les possibilités de TP4 et les aptitudes du Transputer.

3.3 Recherches Futures

Nous avons, au cours de ces travaux, voulu montrer l'importance de la phase d'implémentation d'un protocole de communication de niveau Transport; et par conséquent justifier l'importance des moyens mis au service de cette implémentation:

.techniques logicielles;

.circuit de communication.

La difficulté étant de définir un environnement d'implémentation hautes performances suffisamment flexible pour qu'il soit efficace avec différents protocoles de niveau trois et quatre.

-Le modèle de structures de données a été conçu à cet effet.

-La plate-forme de test étudiée pour implanter, dans un environnement parallèle configurable, un quelconque protocole de Transport.

-Le noyau système a quant à lui été développé pour optimiser l'utilisation du modèle de structures de données dans l'environnement hôte, et plus généralement pour minimiser la dépendance du protocole des tâches systèmes.

Reste à analyser l'influence du protocole (fonctionnalités, structure, propriétés) sur les performances d'une implémentation réalisée dans des conditions de hautes performances.

Plutôt que de prendre parti dans la polémique qui oppose détracteurs et thuriféraires des protocoles standards (TCP et TP4), nous préférons une position d'observateur critique dans laquelle nous proposons:

. d'implanter d'autres protocoles, tels TCP et XTP, dans l'environnement que nous avons défini, afin de faire évoluer chacun vers des performances optimales (par adaptation du modèle de structures de données aux caractéristiques du protocole et étude de l'architecture cible la plus performante);

. de comparer, d'un point de vue efficacité des techniques d'implémentation, degré de parallélisation, dépendance du noyau système, efficacité des services, coût de l'architecture hôte, les protocoles ainsi implémentés;

. de comparer enfin les résultats de ces implémentations avec les études parallèles existantes (on pense à [Kaiserswerth 89] pour les projets de recherche ou à [Schwaderer 90] dans le domaine commercial).

Afin de nous aider à faire évoluer la discussion vers des critères d'aptitudes aux performances élevées pour les protocoles de niveau intermédiaires; l'opportunité du parallélisme au sein d'un circuit de communication; de l'intérêt d'un circuit de communication ASIC ou d'un environnement flexible programmable.

Autant de questions auxquelles nous n'avons pas pu répondre, mais que l'avènement de la plate-forme de test va nous permettre d'étudier précisément.

REFERENCES BIBLIOGRAPHIQUES

Références Bibliographiques

[Ansade 88] Y. Ansade, M. Dang, M. Diaz-Nava, G. Michel, J. Rarivomanana, L. Sponga, Architecture of a new ASIC data communication circuit intended for the ISO level 3, 4, 5: Network, Transport and Session, Proceedings of the IEEE 1988 International Symposium on Circuits and Systems, Helsinki, June 1988.

[Aoki 86] M. Aoki & al., Protocol processing for high-speed packet switching systems, Proceedings of the Int. Zurich Seminar, Zurich. March 1986.

[Aptor 89] Le Réseau Local Industriel FACTOR. APTOR. 1989.

[Aronoff 87] R. Aronoff, K. Mills and M. Weatley, Transport layer performance tools and measurement, IEEE Network, Vol. 1, No. 3, July 1987.

[Berry 85] G. Berry, L. Cosserat, The Synchronous Programming Language ESTEREL and its Mathematical Semantics, Proceedings of the Seminar on Concurrency, Springer-Verlag LCNS 197, 1985.

[Bochmann 83] G. v. Bochmann, E. Cerny, M. Maksud and B. Sarikaya, Testing Transport Protocol Implementations, Proceedings of the CIPS' conference, Ottawa, May 1983.

[Bochmann 88] G. v. Bochmann, Protocol Specification, Rapport de Recherche de l'Université de Montreal, No. 669, Juillet 1988.

[Boehm 80] B. Boehm, Software Economic Engineering, 1980.

[Bolognesi 88] T. Bolognesi and E. Brinksma, Introduction to the ISO Specification Langage LOTOS, Computer Networks and ISDN Systems, January 1988.

[Bricker 86] A. Bricker, L. Landweber, T. Lebeck and M. Vernon, ISO Transport Protocol Experiments, Publication MTR-8600002, MITRE Corporation, Washington CCCI Division, Mc Lean (VA), January 1986.

[Cabrera] L. F. Cabrera, M. J. Karels and D. Mosher, The Impact of Buffer Mangement on Networking Software Performance in Berkeley UNIX 4.2BSD: A Case Study.

[CCITT 84] Livre rouge du CCITT, Termes et Définitions, Tome X, Fascicule X.1, VIII iéme Assemblée pléniaire, Malaga- Torremolonos, Octobre 1984.

[Cerf 83] V.C. Cerf and E. Cain, The DoD Internet Architecture Model", Computer Network, Vol. 7, No. 5, October 1983.

[Cheriton 86] D. Cheriton, VMTP: a protocol for the next generation of communication systems, ACM SIGCOMM' 86, Stowe, Vermont, August 1986.

[Cheriton 89] D.Cheriton and C. Williamson, VMTP as the Transport layer for high performance distributed systems, IEEE Communication Magazine, Vol.27, No.6, June 1989.

[Chesson 87] G. Chesson, The Protocol Engine Project, Unix Rev., pp 70-77, September 1987.

[Chesson 89] G. Chesson, XTP/PE design consideration, Proceedings of the IFIP Workshop on Protocols for High Speed Networks, Rüschlikon, May 1989.

[Chesson 90] G. Chesson, Multicast Heuristics, Transfer (Protocol Engines Information), Vol. 3, No. 4, July/ August 1990.

[Clark 87] D. D. Clark, M. L. Lambert and L. Zhang, NETBLT: A Bulk Data Transfer Protocol, Network Working Group Request for Comments, RFC 998, March 1987.

[Clark 88] D. D. Clark, M. L. Lambert and L. Zhang, NETBLT: A High Throughput Transport Protocol, ACM 1988.

[Clark 89] D.D Clark, V.Jacobson, J. Romkey and H.Salwen, An analysis of TCP processing overhead, IEEE Communication Magazine, Vol.27, No.6, June 1989.

[CNMA 87] CNMA Implementation Guide; Revision 3.1, ECMA, December 1987.

[Cohn 88] M. Cohn, A High-Performance Transfer Protocol for Real-Time LANs, Proceeding of the MILCOM 88, San Diego, October 1988.

[Cole 86] R. Cole and P. Lloyd, OSI Transport Protocol - User Experience, Proceedings of the OPEN SYSTEMS '86, 1986.

[Colella 85] R. Colella, R. Aronoff and K. Mills, Performance improvement for ISO Transport, Proceedings Ninth Data Communication Symposium, Whistler Mountain, British Columbia, September 1985.

[Comer 88] D. E. Comer, Internetworking with TCP/IP: principles, protocols and architecture, Prenctice Hall, 1988.

[Cornafion 81] Systèmes informatiques répartis; concepts et techniques, Dunod Informatique, 1981.

[Crocus 77] Systèmes d'exploitation des ordinateurs, Dunod Informatique, 1977.

[Dang 88a] M. Dang, C. Diot, I. Sabouni and L. Sponga, Specific Data Structure intended for the Implementation of High Level ISO standards: Associated Algorithms and Dedicated Hardware, Proceedings of the EUROMICRO 1988, Zurich, August 1988.

[Dang 88b] M. Dang, G. Michel, Y. Ansade, M. Diaz-Nava, C. Diot, J.Rarivomanana, I. Sabouni and L. Sponga, Aspect of Parallelism in the Architecture Definition of an ASIC Data Communication Circuit, Proceedings of the DCT 1988, Limerick, October 1988.

[Dijkstra 65] E. W. Dijkstra, Cooperating Sequential Processes, Technical Report EWD-123 (1965), reproduit dans F. Genuys Editor, Programming Languages, Academic Press, 1968.

[Diot 88] C.Diot et M. Dang, Contribution à l'étude de l'implémentation des protocoles de communication des couches hautes de l'OSI, Research Report RR 741 -I-, LGI Laboratory, Grenoble, octobre 1988.

[Diot 89] C. Diot and M. Dang, Performances of a Communication Circuit Intended for the Implementation of the High Level Layers of the OSI model, Proceedings of the 1989 IEEE International Conference on System Engineering, Dayton, August 1989.

[Diot 90a] C. Diot and M. Dang, Using Transputer in the Design of High Performance Architectures Dedicated to the Implementation of the OSI Transport class 4 Protocol, Proceedings of the 1990 NATUG Spring Meeting, Sunnyvale, April 1990.

[Diot 90b] C. Diot and M. Dang, A High performance implementation of OSI Transport Protocol Class 4; Evaluation and Perspectives, Proceeding of the 15th International Conference on local Computer network, IEEE press, Minneapolis, October 1990.

[Doeringer 90] W. Doeringer, D. Dykeman, M. Kaiserswerth, B. Meister, H. Rudin and R. Williamson, A Survey of Light-Weight Transport Protocols for High-Speed Networks, Research Report, IBM Research Division, Rüschlikon, 1990.

[EF68000 79] Spécifications du microprocesseur EF 68000, Thomson-Efcis, 1979.

[Esprit 89] ESPRIT Project Supernode II (P2528), Technical Annexes, CEC, 1989.

[Favre 90] M. Favre et alii, PCC: Un environnement de programmation parallèle en C pour Transputer, Rapport de Recherche de l'IMAG, à paraître, Grenoble, 1990.

[Fraser 79] A.G. Fraser, Datakit - A Modular Network for Synchronous and Asynchronous Traffic, Proceedings of the ICC, 1979.

[GAM-T 87] Modèle de référence des réseaux locaux temps réels militaires - couche transfert, Ministère de la Défense, Février 1987.

[Gantenbein 88] D. Gantenbein, R. F. Hauser and E. Mumprecht, Implementation of the OSI Transport Service in a Heterogenous Environment, Research Report, IBM Research Division, Rüschlikon, 1988.

[Gerla 80] M. Gerla, L. Kleinrock, Flow Control: A Comparative Survey, IEEE Trans. Comm., Vol. 28, No. 4, April 1980.

[Kaiserswerth 89]M. Kaiserswerth, D. Giarrizzo, T. Wicki and R. Williamson, High-Speed Parallel Protocol Implementation, Proceedings of the IFIP Workshop on Protocols for High-Speed Networks, Rüschlikon, May 1989.

[Heatley 89] S. Heatjey and D. Stokesberry, Analysis of Transport Measurements Over a Local Area Network, IEEE Communication Magazine, June 1989.

[Hoare 79] C.A.R. Hoare, Communicating Sequential Process, Communication of the ACM, April 1979.

[IBM 86] IBM leaks performance details of forthcoming token ring network, IBM Data Communications, June 1986.

[IBM 88] System Network Architecture LU6.2 Reference: Peer Protocols, available through IBM branch offices, order no. SC30-3422, 1988.

[IEEE 85] Institute of Electrical and Electronics Engineers, ANSI/IEE Std. 802.2, ISO/DIS 8802/2; Logical Link Control.

[INMOS 88a] Communication Process Architecture, Prenctice Hall, 1988.

[INMOS 88b] Transputer Instruction Set, Second Edition, Inmos, 1988.

[INMOS 88c] The Transputer Implementation of Occam, Inmos, 1988.

[INMOS 88d] OCCAM 2 Reference Manual, Prentice Hall, 1988.

[INMOS 89a] The Transputer Data Book, Second Edition, Inmos, 1989.

[INMOS 89a] The Transputer Application notebook, First Edition, Inmos, 1989.

[ISO 84] OSI Transport Service Definition, Standard ISO-8072, 1984.

[ISO 86] OSI Transport Protocol Definition, Standard ISO-8073, 1986.

[ISO 87a] OSI Addendum to enable class 4 operations over connectionless mode network service, Standard ISO-8072/DAD2, 1987.

[ISO 87b] OSI Network Service Definition, Standard ISO-8348, 198?.

[Jacobson 88] V. Jacobson and R. Braden, TCP Extensions for Long-Delay Paths, RFC 1072, Network Information Center, October 1988.

[Jeugt 88] G. v. d. Jeugt, E. Dirkx and J. Tiberghien, Protocol Description and Simulation in the OCCAM programming Language, Microprocessing and Microprogramming 24, 1988.

[Kaminsky 86] M. A. Kaminsky, Manufacturing Automation Protocol (MAP), INTERKAMA Kongress 1986, Dusseldorf, october 1986.

[Kanakia 88] H. Kanakia and D. R. Cheriton, The VMPnetwork adapter board (NAB): high performance network communication for multiprocessors, ACM SIGCOM '88 Symposium, Stanford, CA, August 1988.

[Krakowiak 85] Serge Krakowiak, Principes des systèmes d'exploitation des ordinateurs, Dunod Informatique, 1985.

[Krishnakumar 89] A.S. Krishnakumar and K. Sabnani, VLSI Implementations of Communication Protocols - A Survey, IEEE Journal on Selected Areas in Communications, Vol. 7, No. 7, September 1989.

[Lantz 84] K. A. Lantz, W. I. Nowicki and M. M. Theimer, Factors Affecting the Performance of Distributed Applications, Communication of the ACM, 1984.

[Lasker 84] V. Lasker, M. Lien, E. Benhamou, An Architecture for High Performance Protocol Implementation, Proceedings of the IEEE INFOCOM' 84, April 1984.

[LL1T 89] Spécification de la carte à un Transputer LL1T, Document Technique, APTOR, 1989.

[Logical 89] Transputer Toolset V89.1, Logical System Inc., 1989.

[Luderer 81] G. W. R. Luderer, H. Che and W. T. Marshall, A virtual circuit switch as the basis for distributed systems, Proceeding of the 7th IEEE Data Communication Symposium, October 1981.

[Mantelman 88] L. Mantelmann, Upper layers: From bizarre to bazar, Data Communication, January 1988.

[Margolin 71] B. H. Margolin, R. P. Parmelee, M. Schwotzoff, Analysis of free storage algorithms, IBM Systems Journal 10, Avril 1971.

[Martin 87] J. Martin, SNA: IBM's Networking Solution, Prenctice-Hall 1987.

[MC3 87] Projet MC3 : Définition et Validation de l'architecture de MC3 (VLSI pour protocoles de communication Session, Transport et Réseau), APTOR S.A., DOLPHIN S.A., LGI/IMAG, Mars 1987.

[Meister 85] B. Meister, Ph. Janson and L. Svodobova, Connection-oriented versus connectionless protocols: a performance study, IEEE Trans. Computers, Vol. C34, No. 12, December 1985.

[Meister 87] B. Meister, A performance study of the ISO Transport Protocol, Proceedings of the 7th International Conference on Distributed Computing Systems, Berlin, September 1987.

[Mills 85] K.L. Mills, J. W. Gura and C. M. Chernick, Performance Measurement of OSI Class 4 Transport Implementations, NBS Report No. PB85-177657, February 1985.

[Minet 89] Pascale Minet, Performance Evaluation of GAM-T-103 Real Time Transfer Protocols, Proceedings of the INFOCOM'89, Ottawa, April 1989.

[Murray 88] K. A. Murray and A. J. Wellings, Issues in the Design and Implementation of a Distributed Operating System for a Network of Transputers, Microprocessing and Microprogramming 24, 1988.

[Postel 81] J. Postel, Transmission Control Protocol, RFC 793, Network Information Center, September 1981.

[Rolin 90] P. Rolin, Réseaux: normes et Protocoles, 3ième édition, Hermès, 1990.

[Sanders 90] R. M. Sanders, XTP Tutorial, Report of the University of Virginia (Dpt of Computer Science), Charlottesville, 1990.

[Savard 90] P. Savard, Dossier technique de la machine PET, Rapport technique APTOR, Octobre 1990.

[Serre 86] J. M. Serre, E. Cerny, G. v. Bochmann, A methodology for implementing high level communication protocols, Proceedings of the 19th Annual Hawaii International conference on System Sciences, 1986.

[Simon 86] T. Simon, O. Spaniol, J. Suppan-Borowka, Performance Measurement of Different Protocol Layers in a LAN Environment, Proceedings of the EFOC/LAN 86, Amsterdam, June 1986.

[Smith 86] K. Smith et alii, A local-Area-Network VLSI chip Set, Proceedings of the IMS 86.

[Stokesberry 83] P. P. Stokesberry, Use of ISO class 4 Transport on local area networks, publication of the National Bureau of Standard, 1983.

[Strayer 88] W.T. Strayer, A. C. Weaver, Performance Measurement of Data Transfer Services in MAP, IEEE Network, Vol. 2, No. 3, May 1988.

[Svodobova 88] Liba Svobodova, Implementing ISO system, Research Report RZ 1715, IBM Research Division Zurich Research Laboratory, June 1988.

[Svodobova 89] L. Svodobova, Implementing OSI Systems, IEEE JSAC on Architecture and Protocols for Computer Networks: The State-of-Art, Vol. 7, No. 7, September 1989.

[Schwaderer 90] W. D. Schwaderer, XTP in VLSI; Protocol Decomposition for ASIC Implementation, Proceeding of the 15th International Conference on local Computer network, IEEE press, Minneapolis, October 1990.

[TDS 87] The Transputer Development System 2.0, D700C, INMOS Corp., 1987.

[Varghese 87] G. Varghese and T. Lauck, Hashed and Hierarchical Timing Wheels: Data Structures for the Efficient Implementation of Timer Facilities, ACM 1987.

[Vazquez 88] E. Vazquez, R. Colella, J. Vinyes, J. Fox and J. Berrocal, Performance of OSI Transport over ACCUNET and IBERPAC, IEEE, 1988.

[Whaley 89] A. Whaley, XTP, The Kernel Reference Model, Protocol Engines Inc., 1989.

[Watson 81] R. W. Watson, Timer-Based Mechanisms in Reliable Transport Protocol Connection Management, North-Holland, Computer networks 5, 1981.

[Watson 83] R. W. Watson, Delta-t Protocol specification, Report UCID-19293, Lawrence Livermore Laboratory, April 1983.

[Watson 87] R. W. Watson and S. A. Mamrak, Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices, ACM Transactions on Computer Systems, Vol. 5, No. 2, may 1987.

[Woodside 89] C. M. Woodside and J. R. Montealegre, The effect of buffering strategies on protocol execution performance, IEEE Transaction on Communications, Vol. COM-37, No. 6, June 1989.

[Yashiro 86] Z. Yashiro, M. Aoki and M. Nishiwaki, Hardware Design for a High-Throughput Packet Switching System, Review of the Electrical Communication Laboratories, Vol. 34, No.6, 1986.

[XTP 90] Protocol Engines Inc., XTP Protocol Definition, Revision 3.5, September 1990.

ANNEXES

ANNEXE A; illustrations du fonctionnement des PnPDU

Exemple d'utilisation des structures de données dédiées sur le traitement successif, par les protocoles de Transport et Réseau, d'une suite de NPDU (figure A.1) et d'une TSDU (figure A.2).

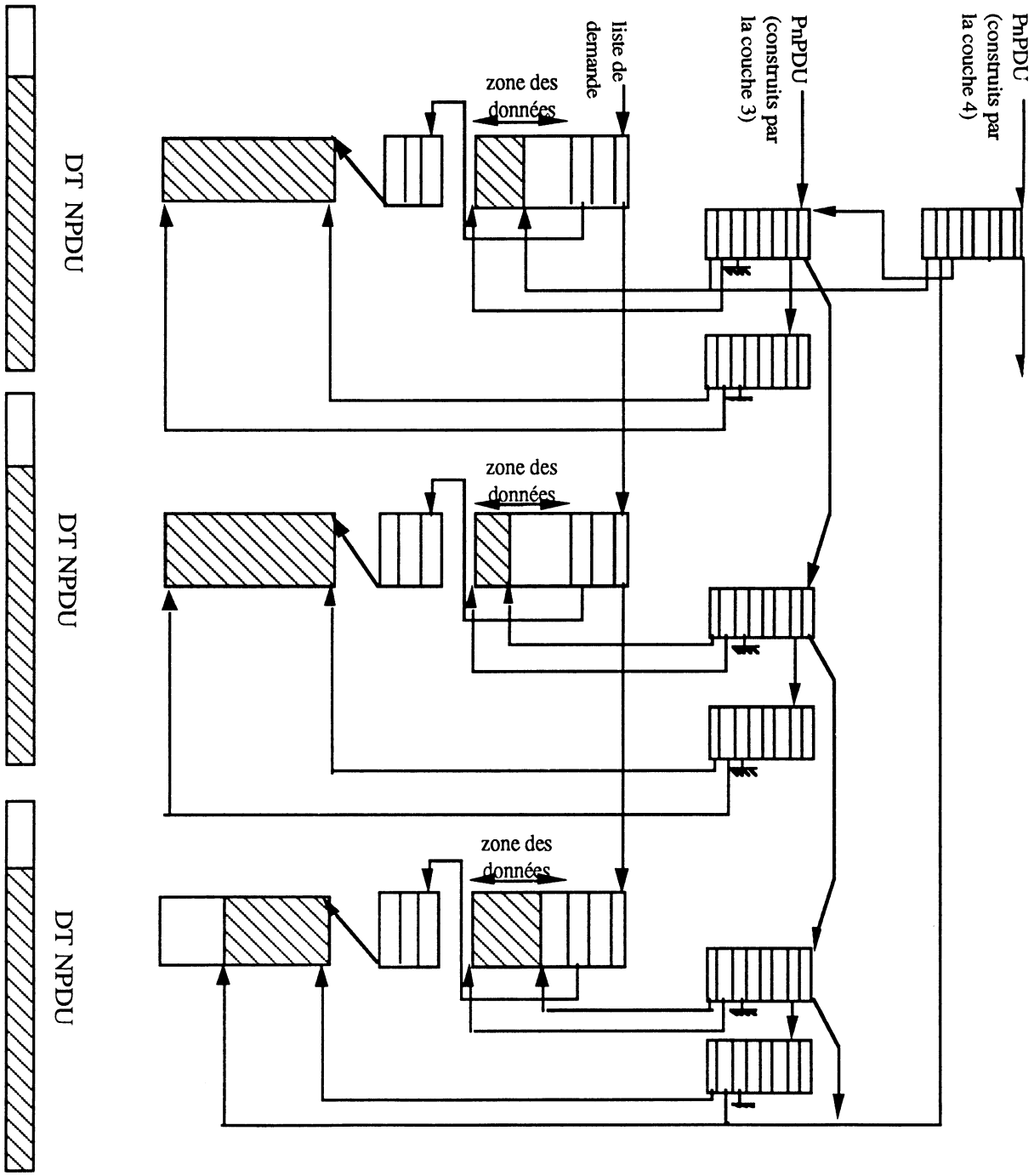


figure A.1

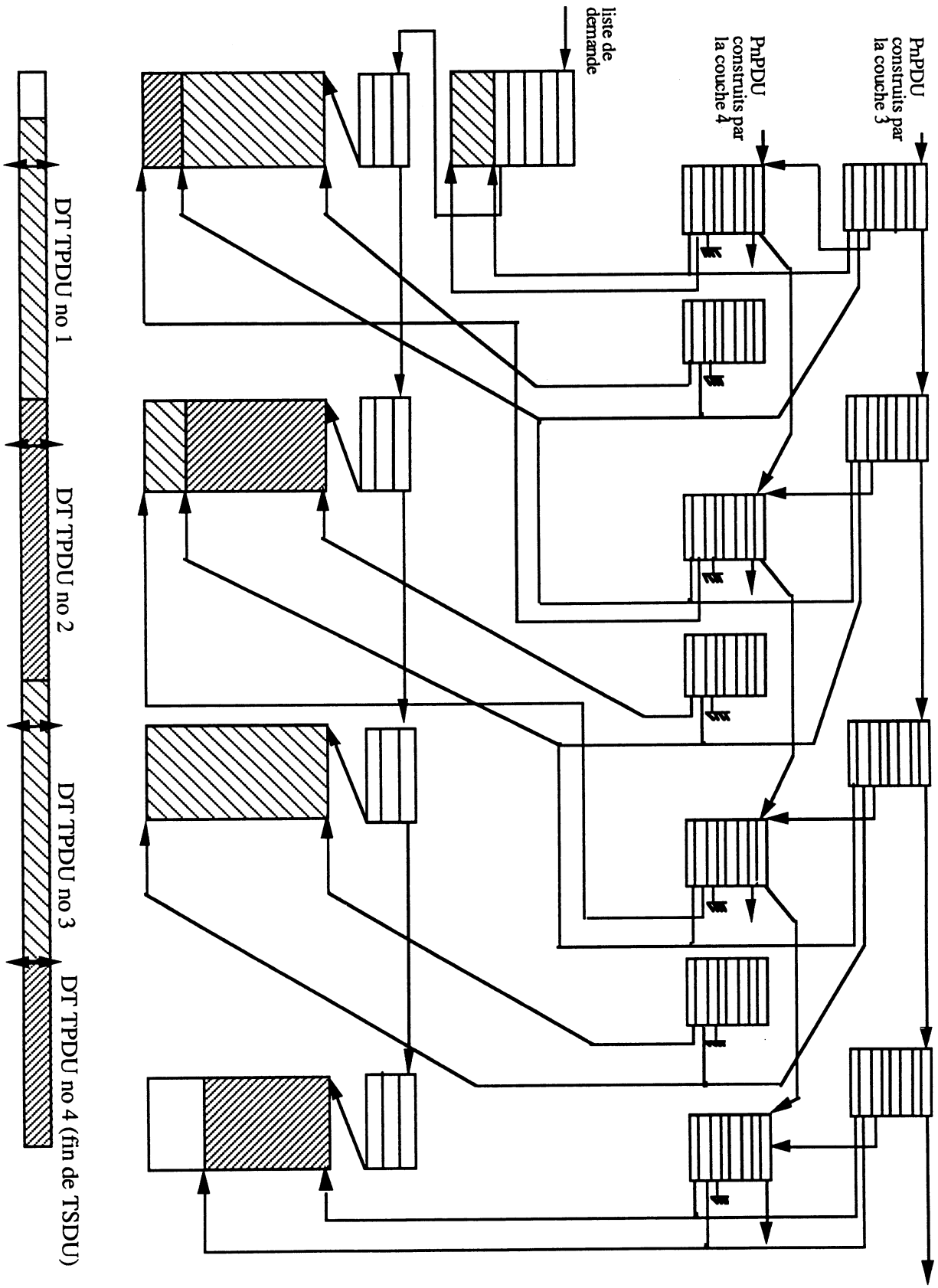

liste de demande
PnPDU construits par la couche 4
PnPDU construits par la couche 3
DT TPDU no 1
DT TPDU no 2
DT TPDU no 3
DT TPDU no 4 (fin de TSDU)

figure A.2

ANNEXE B; mesures obtenues dans la phase d'évaluation du protocole

Les équations utlisées dans cette phase de mesures ont été établies à partir de la mesure du temps de traitement détaillé de chaque événement entrant. A titre d'exemple, la figure B.1 propose une étape de ces mesures pour les événements qui transportent des données utiles.

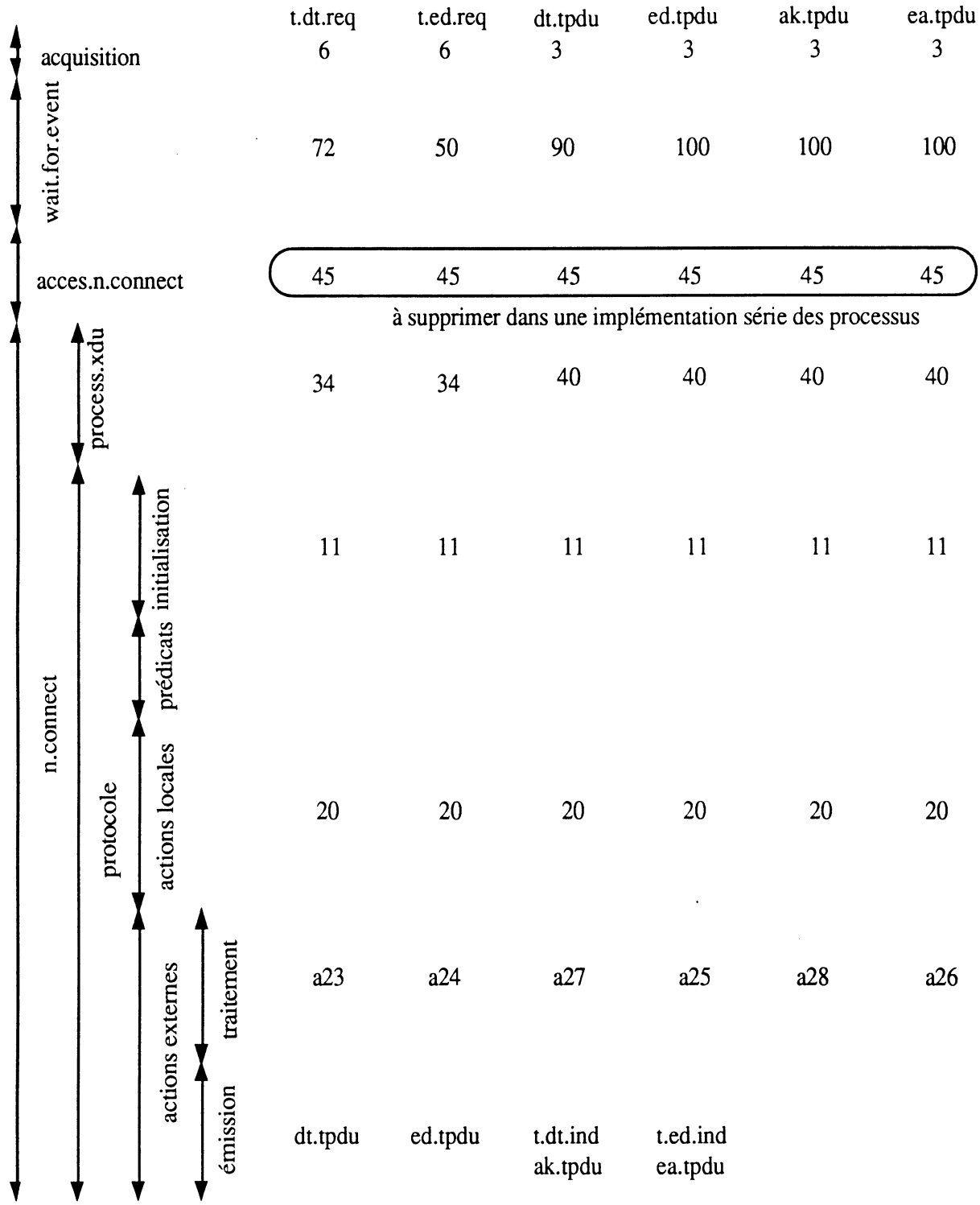


figure B.1

Cette table fait appel à un ensemble d'actions (numérotées de a23 à a27), selon la terminologie retenue par la norme ISO 8073. Elle provoque de surcroît l'émission de TPDU ou de TSDU dont le coût à lui aussi été chiffré.

Tous les chiffres ainsi obtenus ont donné lieu à la construction d'une équation pour chaque événement traité par le protocole. Pour les primitives service étudiées figure B.1, les équations obtenues sont les suivantes:

$$\Delta t.dt.req = 190 + \sum_{1}^{nb.seg} [111 + 61 * (nb.ress - 1) + 49 * (nb.pnpdu - nb.ress)]$$

$$\Delta t.dt.ind = 377 * nb.seg + 115 \quad (+ 78)$$

$$\Delta ak.tpdu = 389 + 7 * nb.pnpdu + 38 * nb.ress + 35 * nb.seg + 28 * (nb.pnpdu - (nb.ress + nb.seg))$$

$\Delta t.ed.req = 3$ $\Delta t.ed.ind = 346$ $\Delta ea.tpdu = 382$

Avec nb.seg, le nombre de segments fragmentés par le protocole dans une TSDU; nb.pnpdu le nombre de PnPDU utilisés pour traiter l'événement et nb.ress le nombre de ressources allouées par le processus de réception pour mémoriser l'événement entrant en liste de demande.

Enfin, l'assemblage de toutes les équations élémentaires produit les équations des courbes commentées dans le paragraphe 2.1:

Nombre de TSDU traités par le protocole en 1 seconde (figure 2.1.15):

$$\tau_{seg} = \frac{nb\_seg \ 10^6}{160 + 150 \ nb\_seg}$$

Nombre de DT TPDU traités et réassemblés en 1 seconde (débit exprimé en TSDU par seconde). (figure 2.1.15):

$$\tau_{reass} = \frac{10^6}{115 + 347 \ nb\_seg}$$

Débit, en TSDU par seconde, de TSDU transmises; compte tenu de la segmentation en TPDU et du temps de restitution des ressources acquittées (la fenêtre de contrôle de flux est de taille 8 et on utilise une *stratégie d'acquittement par fenêtre*) (figure 2.1.16):

$$\tau w\_seg = \frac{\frac{w\_size}{nb\_seg}\ 10^6}{\frac{w\_size}{nb\_seg}(160 + 150\ nb\_seg) + 87\ w\_size + 260}$$

Même courbe que la précédente; Les ressources ne sont cette fois plus restituées par le protocole de transport, mais le temps de traitement des acquittements est comptabilisé (figure 2.1.19):

$$\tau' w\_seg = \frac{\frac{w\_size}{nb\_seg}\ 10^6}{\frac{w\_size}{nb\_seg}(160 + 150\ nb\_seg) + 20\ w\_size + 255}$$

Débit, en TSDU par seconde, de TSDU transmises; compte tenu de la segmentation en TPDU et du temps de restitution des ressources acquittées (la fenêtre de contrôle de flux est de taille 8 et on utilise une *stratégie d'acquittement par TSDU*) (figure 2.1.16):

$$\tau tsdu\_seg = \frac{10^6}{420 + 237 nb\_seg}$$

Nombre de DT TPDU traités et réassemblés en 1 seconde (débit exprimé en TSDU par seconde). Compte tenu du temps de restitution des ressources acquittées (la fenêtre de contrôle de flux est de taille 8 et on utilise une *stratégie d'acquittement par fenêtre*). Figure 2.1.22:

$$\tau w\_reass = \frac{\frac{w\_size}{nb\_seg}\ 10^6}{\frac{w\_size}{nb\_seg}(115 + 347\ nb\_seg) + 78}$$

Remarque: Cette liste d'équations n'est pas exhaustive. D'autres équations ont été en particulier construites pour évaluer les différentes techniques de gestion de la mémoire (statique ou dynamique) et pour étudier la restitution de ressource.

ANNEXE C; les primitives du gestionnaire mémoire

Le gestionnaire mémoire à été conçu pour minimiser, compte tenu du type d'application et de l'environnement d'étude, le temps de traitement des procédures d'allocation et de restitution de ressources. Les procédures écrites sont réutilisables pour d'autres applications dans le même domaine, et son définies sous forme d'une Librairie OCCAM. Trois primitives ont été implémentées:

```
PROC Mem.Init       ([] BYTE        Heap.b,
                    VAL [] INT      Page.size,
                    INT             Result )

PROC Mem.Alloc      ([] BYTE        Heap.b,
                    VAL INT         Page, RecSize,
                    INT             PtBlock, Result )

PROC Mem.Dealloc    ([] BYTE        Heap.b,
                    VAL INT         PtBlock,
                    INT             Result )
```

La procédure *Mem.Init* permet à l'utilisateur d'initialiser les variables qui partitionnent la mémoire en pages indépendantes. Pages qui seront associées à un type de ressource particulier. Le nombre de pages et la taille des pages sont définis par le paramètre *Page.size*. La procédure retourne un compte rendu d'initialisation: la variable *Result*.

La procédure *Mem.Alloc* alloue un bloc mémoire dans la page spécifiée. *Page* définit la page où doit s'effectuer l'allocation et *RecSize* la taille (en octets) du bloc à allouer. Cette procédure retourne un pointeur sur l'adresse du bloc alloué (*PtBlock*), ainsi que la variable *Result* (comme définie dans *Mem.Init*).

La procédure *Mem.Dealloc* rend à la mémoire un bloc alloué précédemment. Le pointeur *PtBlock* identifie l'adresse de ce bloc. Le bloc sera restitué en une seule fois, dans son intégralité.

ANNEXE D; le gestionnaire de temporisations

Comme le gestionnaire mémoire, le gestionnaire de temporisations a été implanté sous forme d'une librairie OCCAM afin d'en rendre l'utilisation possible dans d'autre implémentations. Le gestionnaire de temporisation est basé sur la gestion dynamique de liste et sur un processus en polling dont la période fixe la précision des temporisations. Un unique canal est utilisé pour signaler au protocole les temporisations arrivées à échéance.

Selon les recommandations du profil CNMA, une précision sur les temporisations de 0.1 seconde est suffisante.

Pour simplifier la description, tous les timers ont été considérés de durée égale. Ainsi, à l'initialisation, on ne crée qu'une liste de temporisateurs (rappelons que, selon le principe du gestionnaire mémoire, le gestionnaire de temporisations fonctionne avec une liste de descripteurs de temporisation par type de temporisation). Les primitives implémentées sont au nombre de six:

```
PROC Timer.Manager   ([] INT          Heap,
                     CHAN OF ANY     Timer.Alarm,
                     VAL INT         ticks.wait )

PROC Timer.Init      ([] INT          Heap)

PROC Timer.Make      ([] INT          Heap,
                     VAL INT         Ti.idf, Ti.delay,
                     INT             Pt.timer, Result)

PROC Timer.Start     ([] INT          Heap,
                     VAL INT         Pt.timer,
                     INT             Result)

PROC Timer.Stop      ([] INT          Heap,
                     VAL INT         Pt.timer,
                     INT             Delayed.time, Result)

PROC Timer.Remove    ([] INT          Heap,
                     VAL INT         Pt.timer,
                     INT             Result)
```

La procédure *Timer.Manager* est un processus indépendant, qui s'exécute en polling pour surveiller, avec la précision *Tick.wait*, l'état de la tête de liste des descripteurs de temporisateur. Il signale au protocole toute échéance sur le canal *Timer.alarm*. Notons que ce processus peut être supprimé quand la précision (soit la fréquence du polling) le nécessite. On utilise alors les primitives de gestion intégrées au Transputer pour signaler l'arrivée à échéance d'une temporisation.

La procédure *Timer.Init* permet, à l'initialisation du protocole, de réserver une zone mémoire pour la gestion des listes de temporisation (cette zone est représentée par le tableau *Heap*).

La procédure *Timer.Make* permet à l'utilisateur de créer un descripteur de temporisation, de lui allouer un identificateur propre au protocole (*Ti.idf* permet au protocole d'identifier le type de temporisation et la connexion sur laquelle intervient cette temporisation) et de fixer sa durée (*Ti.delay*). La procédure retourne l'adresse du descripteur alloué dans *Pt.timer* et un compte rendu d'exécution dans *Result*. Pour toute modification de l'état d'une temporisation, l'utilisateur identifiera la temporisation au moyen de la variable *Pt.timer*. Par contre, le gestionnaire de temporisations signale au protocole les temporisations arrivées à échéance à partir de *Ti.idf*.

La procédure *Timer.Start* est utilisée pour armer la temporisation identifiée par *Pt.Timer*. La procédure retourne un compte-rendu d'exécution dans *Result*.

La procédure *Timer.Stop* permet à l'utilisateur d'interrompre une temporisation identifiée par *Pt.Timer*. La procédure retourne, dans la variable *Delayed.time*, le temps restant avant expiration de la temporisation, et le compte-rendu *Result*.

Enfin la procédure Timer.Remove permet de restituer un descripteur de temporisation d'adresse *Pt.timer* et de détruire la temporisation associée.

ANNEXE E; structure de l'implémentation du protocole de Transport OSI classe 4 en OCCAM

Les logiciels écrits en OCCAM sont développés dans un environnement propre au Transputer appelé *"Transputer Development System"*. TDS regroupe tous les outils utilisables pour développer et mettre au point une application écrite en OCCAM (Editeur, Compilateur, déboggueur, gestionnaire de fichier).
Pas question donc d'utiliser un éditeur classique.

TDS propose un éditeur *"foldé"*. Un logiciel OCCAM est donc structuré en *"folds"* ou feuilles. La structure de ces *folds* permet une meilleure lisibilité d'une application parallèle écrite en OCCAM. Chaque fold possède un titre qui permet d'en identifier le contenu. Selon le principe de l'*éditeur foldé*, nous décrivons, dans les pages suivantes, la structure OCCAM de l'implémentation réalisée.

Le contenu d'un *Fold* est délimité par trois parenthèses. L'apparition d'un *Fold* interne est identifiée par trois points. Ces trois points sont immédiatement suivis, éventuellement, du type de fold (F pour fichier, SC pour unité de compilation séparée et EXE pour exécutable).

```
{{{
...EXE  Transputer                  --logiciels de l'application Transport classe 4 et de son
                                    --environnement; pour l'instant implantés sur un seul Transputer
...F     Operating.System.Librairies --déclaration des librairies contenant le gestionnaire mémoire et le
                                    --gestionnaire de temporisation
}}}
```

```
{{{ EXE Transputer

CHAN OF ANY chan.in, chan.out:   --déclaration des canaux de communication avec l'entité distante
PLACE chan.in AT 5:              --affectation d'un lien physique à ces canaux
PLACE chan.out AT 1:

...SC   Transport.application    --processus qui regroupe les processus de l'application

SEQ
  Transport.application (chan.in, chan.out, from.filer, to.filer, screen, keyboard)
:
}}}
```

```
{{{SC Transport.application

PROC transport.application (  CHAN OF ANY chan.in, chan.out, from.filer, to.filer,
                              CHAN OF ANY screen, CHAN OF INT keyboard)

... Librairies utilisées
... Procédures utilisées par tous les processus

VAL   NIL     IS   #FFFFFFFF:    --les champs des descripteurs qui ne sont pas utilisés sont
VAL   BNIL    IS   BYTE #FF:     --initialisés à NIL ou BNIL selon leur format

... VARiables utilisées par tous les processus implémentés
... VARiables systèmes   --utilisées par le gestionnaire mémoire et le gestionnaire de temporisation

...SC   Transport.user          --processus développés pour utiliser le protocole de Transport
...SC   Interface.processus     --processus qui implémente les processus d'interface avec l'utilisateur
...SC   Transclatre             --processus qui implémente le protocole de Transport
...SC   Network.kernel          --ce processus fait à la fois office de couche réseau et d'interface avec le
                                --protocole de Transport

Init.Alloc (Heap, [32, 256, 64, 256, 128, 256, 64, 512, 384], result) --initialisation de la mémoire
... calcul de l'adresse d'implémentation de Heap

PAR    --exécution en paralléle des processus définis ci-dessus
  ... Transport.User
  ... Interface.processus
  ... Transclatre
  ...Network.kernel
:
}}}


{{{SC Transclatre  structure du protocole de Transport

... PROC Transclatre ( {VAR paramètres} )

... Librairies utilisées par Transclatre
...F CONSTantes locales --ce fold contient la déclaration de tous les types de descripteurs et de leurs
                        --champs, ainsi que la déclaration du format des contextes de connexion, des
                        --types d'erreurs, des différentsévénements et états du protocole
...F format de toutes les TPDU et TSDU traitées

... VARiables locales gérées par Transclatre

... Procédures de manipulation des contextes de connexion et de traitement des références
... Procédures de manipulation des PnPDU

...F    processus de traitement des erreurs de protocoles
...F    processus de traitement des erreurs détectées sur les structures de données
...F    processus de traitement des événements sortant  --émission de TPDU ou TSDU
...F    processus de traitement d'un événement sur une connexion
...F    processus d'attente des événements

SEQ
  ... Initialisation des contextes de TSAP et des processus d'interface
  ... Initialisation des références et des contextes de connexion
  ... Initialisation du gestionnaire mémoire
  ... Initialisation du gestionnaire de temporisations
```

```
   attente.sur.événements (Heap, contexte, liste.etat.ref,
                           ad.contextes.de.connexion, ad.descr.dde.h, ad.tampon.h, ad.descr.t.h,
                           ad.ressources.tempo.h, ad.ressources.tempo.l,
                           ad.pnpdu.h, ad.pnpdu.l,
                           cde.prot.h, cde.prot.l, timer.exp) --identification des trois canaux par lequels
:                                                             --peuvent arriver des événements
}}}


{{{processus  d'attente des événements

...PROC attente.sur.événements ( {VAR paramètres} )

   ... processus de gestion de l'acces à n.connect  --n.connect est le processus qui traite un événement
                                                    --sur la connexion à laquelle il a été affecté
   ... procedure d'étude d'une TSDU entrante stockée dans un descripteur de demande
   ... procédure d'étude d'une TPDU entrante décrite au niveau inférieur par PnPDU
   ... procédure d'assignation d'un événement de type temporisation à une connexion

   ALT
         timer.exp ? timer.idf          --une temporisation est arrivée à échéance
               timer.assign (timer.idf)

         cde.prot.h ? ad.descr.de.dde  --des TSDU sont à traiter en réception haute
               WHILE continue
                     continue := verifie.descr (ad.descr.courant)
                     affectation.TSDU (ad.descr.de.dde)

         cde.prot.l ? ad.pnpdu  --des TPDU issus de la couches trois sont à traiter
               WHILE continue
                     continue := verifie.descr (ad.pnpdu)
                     affectation.TPDU (ad.pnpdu)
:
}}}


{{{F    processus de traitement d'un événement sur une connexion

...PROC n.connect ( {VAR paramètres} )

   ...F   machine d'état fini du protocole
   ...F   traitement d'un événement décrit par PnPDU
   ...F   traitement d'un événement contenu dans un descripteur de demande

   SEQ
         IF
               format.xdu = pnpdu
                     process.pnpdu          ( {paramètres} )
               format.xdu = descr.de.dde
                     process.descr.de.dde   ( {paramètres} )
               format.xdu = timer
                     process.timer          ( {paramètres} )
               TRUE
                     Process.error          ( {paramètres} )
:
}}}
```

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Grenoble, le 24 Janvier 1991

DÉPARTEMENT DES ÉTUDES DOCTORALES

Affaire suivie par
Tél : 76.57.

N/Réf. :

Objet :

AUTORISATION de SOUTENANCE

Vu les dispositions de l'arrêté du 23 Novembre 1988 relatif aux Etudes Doctorales
Vu les rapports de présentation de :

- Monsieur ROLIN
- Monsieur JUANOLE

Monsieur DIOT Christophe

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de l'INSTITUT NATIONAL POLYTECHNIQUE de GRENOBLE, spécialité :

"Informatique"

Pour le Président de l'I.N.P.-G.
et par délégation,
le Vice-Président
M. GARNIER

Résumé:

L'implémentation, selon des critères de hautes performances, d'un protocole de communication est une tâche complexe. De nombreux facteurs tels que l'environnement d'implantation, les techniques d'implémentation, l'efficacité du noyau système et les structures de données sont généralement aussi importants que les spécifications du protocole. Nous avons réalisé une implémentation hautes performances du protocole de Transport OSI classe 4, dans un environnement à base de Transputers (microprocesseur RISC d'INMOS). Nous avons montré comment le Transputer, associé au langage OCCAM, pouvait fournir un environnement d'implantation performant. L'implémentation a été conçue pour supprimer les transferts de données (en utilisant des structures de données dédiées) et minimiser les accès aux tâches systèmes (des primitives systèmes ont été développées pour optimiser le temps de gestion de la mémoire et des temporisations). L'étude de performances menée sur cette implémentation a montré qu'il était possible d'atteindre des débits voisins de 5000 TSDU par seconde, et nous a permis de définir l'architecture d'une plate-forme configurable pour l'évaluation des protocoles de niveau intermédiaire, et pour l'implantation hautes performances de ces protocoles. Ces travaux se terminent par l'analyse des fonctionnalités du protocole et des techniques d'implémentation mises au point, pour essayer d'apporter des éléments de réponse au problème de l'avenir des protocoles de Transport classiques (OSI TP4 et TCP) devant les besoins définis par les domaines d'application nouveaux.

Mots clés:

Protocoles de Communication; Hautes Performances; Architecture; Transputer; Transport; Implémentation;

Abstract:

High performance implementation of communication protocols is a complex task. Factors such as architecture of the target digital system, implementation technics, efficiency of the system primitives and data structures are generally as important as the protocol specification caracteristics. Using the Transputer microprocessor, we have realized a high performance implementation of OSI Transport protocol, class 4. We have shown the way in wich a Transputer based environnement (using OCCAM as development language) can be used efficently for protocol implementation. The structure of the protocol implementation has been designed to minimize data transfers (using dedicated data structures) and to reduce treatment time spent in system tasks (an optimized operating system has been developed for memory and timer management). The performance analysis results have shown it is possible to raise performance of 5000 TSDU per second, and has permitted us to define a set of multi-TRANSPUTERs high performance architectures. These architectures will be integrated in an evaluation platform dedicated to "Transfer" protocol implantation. These works stop with discussion on availability of general purpose protocols (such as TP4 or TCP) for new applications type.

Keywords:

Communication Networks; High Performance; Architecture; Transputer; Transport; Implementation;