Une architecture de communication efficace en environnement STREAMS

Vincent Roca[1], Christophe Diot[2]

[1]LSR; INRIA Rhône-Alpes; ZIRST; 655, avenue de l'Europe; 38330 Montbonnot S^t Martin; France
vincent.roca@imag.fr; http://www-gram.imag.fr/Les.Personnes/Vincent.Roca/

[2]INRIA
2004, route des Lucioles; BP 93; 06902 Sophia-Antipolis cedex; France
tel : (33) 93.65.77.56; fax : (33) 93.65.77.65
christophe.diot@sophia.inria.fr; http://www.inria.fr/rodeo/

Résumé. *Ce papier aborde le problème de l'efficacité des techniques d'implémentation hautes performances pour les systèmes de communication. Plusieurs aspects sont abordés : machine hôte, environnement d'implantation, et architecture de la pile de communication. Nous montrons tout d'abord qu'une architecture démultiplexée, avec des chemins de données directs entre applications et driveurs réseaux, permet un excellent support du contrôle de flux local et des mécanismes systèmes de QoS. Nous montrons également comment la technique ILP, destinée à limiter les accès mémoire, peut être intégrée à la méthode d'accès d'une pile TCP/IP. Ses bénéfices restent néanmoins largement dépendants des conditions d'utilisation, de la nature de l'application et des manipulations de données.*
Mots clés. systèmes de communication, STREAMS, hautes performances, ILP, qualité de service.
Abstract. *This paper investigates the efficiency of high performance implementation techniques for the communication system. Several aspects are studied : host machine, implementation environment, and communication stack architecture. We show that a demultiplexed architecture, with direct data paths between the application and the network device driver, enables a good support of local flow control and system-level QoS mechanisms. We also show how the ILP technic, meant to reduce the memory accesses, can be integrated to the access method of a TCP/IP stack. Yet its benefits largely depend on the conditions of use, the kind of application, and the data manipulations.*
Key words. communication systems, STREAMS, high performance, ILP, quality of service.

1. Introduction

Les réseaux informatiques ont connu, depuis les origines, un succès croissant. Ces réseaux étaient alors caractérisés par une “faible” bande passante et des taux de pertes élevés. Les applications étaient elles conçues pour tolérer des délais d’acheminement variables. Enfin, la pile TCP/IP, toujours utilisée, a été développée il y a quinze ans, à une époque où un VAX-11/780 avec 4 mega-octets de mémoire était un gros système. Il n’est donc pas surprenant que le système de communication actuel se révèle impropre aux nouveaux réseaux, applications multimédia, et stations de travail :

- Les réseaux sont désormais caractérisés par de hauts débits, de faibles taux d’erreurs, voir certaines garanties de qualité de service (QoS) (ATM).
- Une nouvelle génération d’applications est apparue, en particulier dans le domaine du multimédia [RFC1633]. Ayant des contraintes temps-réel, elles se satisfont mal des services réseaux traditionnels de type “meilleur effort”.
- Enfin les stations de travail actuelles ont énormément évoluées. En dépit de la présence de caches, la technologie des mémoires n’a pas suivi celle des processeurs. [Mosberger95] présente une analyse détaillée durant des transferts TCP/IP des événements de bas niveau du processeur ALPHA 21064 à 175 MHz. Une première conclusion est que ce processeur n’est *jamais actif plus de 37% du temps*. Une deuxième conclusion est que les traitements sont *limités soit par les accès aux instructions, soit par les accès aux données, jamais par le CPU*.

Ces raisons rendent une évolution majeure du système de communication nécessaire. Le but de nos travaux est d’*améliorer le système de communication des stations de travail actuelles*. Plusieurs approches ont été proposées au cours des dernières années, chacune visant à améliorer un aspect particulier. Nos travaux nous ont conduit à *examiner les conditions d’application et l’efficacité de certaines de ces approches*.

Notre article est structuré ainsi : la section 2 est consacrée à la présentation des différent composants de notre plate-forme expérimentale et à leur justification. La section 3 est une étude de performances qui nous permettra de quantifier les impacts de ces composants et de mettre en évidence certains problèmes. Enfin nous concluons.

2. Plate-forme expérimentale

Notre plate-forme expérimentale a été conçue afin d’étudier les aspects suivants :

Les impacts de l’environnement d’exécution de la pile de communication

Son but est de fournir des services de base et des concepts architecturaux qui facilitent le développement des driveurs de communication. Nous nous intéresserons aux deux environnements majeurs des systèmes ouverts : BSD (l'environnement des premières piles TCP/IP), et STREAMS (introduit plus tard par AT&T).

Les impacts de l'architecture du système de communication

Des alternatives autres que la traditionnelle structure en couches sont possibles. Nous nous intéresserons aux architectures démultiplexées. Notre objectif est d'obtenir une vision pratique des différents bénéfices et limites associés à cette approche, difficile à mettre en oeuvre de par le principe d'indépendance des couches.

Les impacts de la technique ILP

Si la technique d'intégration de manipulations de données ILP [Clark90] est d'un point de vue théorique simple, en revanche son incorporation dans le système de communication est complexe. C'est là encore la conséquence de la remise en cause de la structuration en couches. Nous nous intéresserons à l'intégration des copies de données et du calcul de checksum TCP. Nous montrons comment l'appliquer et quels sont les bénéfices/limites que l'on peut en attendre dans des conditions réelles.

2.1. Comparaison des environnements BSD et STREAMS

Cette section n'a pas pour but de présenter ces deux environnements (voir [Stevens94] pour BSD, et [STREAMS90, Roca94] pour STREAMS), mais d'en comparer les caractéristiques. Cette comparaison a ceci d'intéressant que BSD et STREAMS reposent sur des concepts radicalement opposés :

- BSD fonctionne par *appels de fonctions*, alors que
- STREAMS fonctionne par *passage de messages.*

Table 1: BSD et STREAMS, résumé des différences.

Environnement BSD	Environnement STREAMS
⇨ favorise l'intégration ⇨ n'impose aucune structuration interne ⇨ utilisé uniquement avec le système de communication ⇨ méthode d'accès orientée transport	⇨ favorise l'indépendance, la portabilité, et la flexibilité ⇨ interfaces internes standardisées ⇨ utilisé par divers systèmes d'E/S ⇨ méthode d'accès générique

Cette opposition se retrouve dans les objectifs de ces environnements (Table 1). Alors que BSD vise à favoriser l'intégration et les performances des composants, à l'opposé STREAMS met en avant des aspects d'indépendance, de portabilité, et de flexibilité. Des interfaces normalisées (TPI, NPI, et DLPI) entre les différentes couches de la pile de communication sont également définies dans STREAMS, alors que BSD n'impose aucune structuration interne. Enfin, si l'environnement BSD est restreint aux systèmes de communication, en revanche STREAMS a des vues plus larges (TTY par exemple). C'est la raison pour laquelle les appels systèmes BSD (Socket) sont spécifiquement conçus pour accéder aux services d'un protocole de transport, alors que les appels systèmes STREAMS sont génériques. Ainsi, STREAMS distingue les notions d'API (XTI ou Socket) et de méthode d'accès (appels systèmes STREAMS).

2.2. Pile TCP/IP classique sous STREAMS

L'étude d'une pile TCP/IP classique sous STREAMS nous a permis de mettre en évidence plusieurs problèmes :

Certains sont issus du *caractère multiplexé de l'architecture* (les driveurs TCP, UDP, IP, et DLPI multiplexent plusieurs streams supérieurs au dessus de plusieurs streams inférieurs). Ceci a des impacts négatifs sur le contrôle de flux local dont la visibilité est limitée à un stream : applications et driveurs physiques sont de ce fait indépendants ce qui conduit aisément à des problèmes d'engorgement.

Une deuxième catégorie de problèmes provient d'une *mauvaise utilisation de STREAMS*. En particulier, les messages de transmission de données sont fréquemment mis en queue afin de bloquer l'application lorsque TCP a épuisé sa fenêtre d'émission. L'inconvénient de cette technique est qu'elle induit des traitements asynchrones qui augmentent d'autant la latence des transmissions.

Enfin *STREAMS souffre de limitations fonctionnelles*. Des performances médiocres sont obtenues en raison de la stratégie de gestion des buffers de réception applicatifs au sein du Stream-Head (Section 3.1) : contrairement à la couche Socket qui essaie d'optimiser le remplissage des buffers applicatifs, le Stream-Head retourne les données à l'application dès qu'elles arrivent. [STREAMS90] ne propose aucune autre stratégie. La conséquence est qu'un nombre accru d'appels systèmes de réception (`getmsg`/`read`) est requis et le nombre de changements de contextes augmenté.

2.3. Pile TCP/IP démultiplexée sous STREAMS

L'analyse de ces problèmes nous a conduit à nous intéresser à une architecture démultiplexée regroupant toutes les tâches de multiplexage/démultiplexage en un seul lieu. Dans cette approche tous les protocoles sont implantés sous forme de modules plutôt que de driveurs multiplexés. Lorsqu'une application ouvre un point d'accès transport, un stream est crée et les modules implantant les protocoles appropriés sont automatiquement insérés sur ce stream. Nous appelons *Canal de Communication* (ou CC) l'association d'un stream et de ses modules.

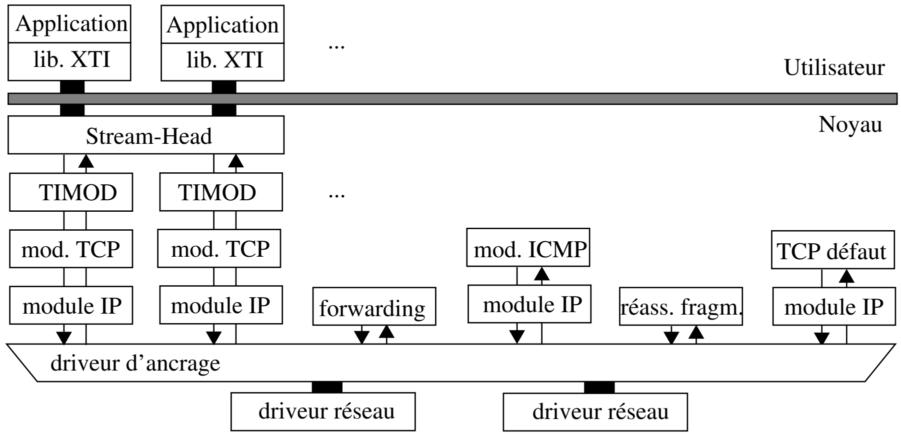

Figure 1: La pile TCP/IP démultiplexée sous STREAMS.

Le driveur d'ancrage

Le driveur d'ancrage a trois rôles :

- il sert de point d'ancrage pour les différents streams,
- il fournit une interface de type DLPI aux driveurs réseaux[1], et enfin
- il démultiplexe les paquets entrants. Cette opération est fondée sur l'examen des différentes en-têtes de protocoles du paquet (ethernet, IP, TCP/UDP).

Les canaux de communication supplémentaires

Les tâches de "forwarding" (retransmission des paquets qui ne sont pas arrivés à destination), les traitements ICMP, et le réassemblage des fragments IP ne sont pas liées à un chemin de données particulier. Ces tâches sont prises en charge par des CC dédiés créés lors de la mise en place initiale.

L'architecture démultiplexée a mis en évidence une difficulté supplémentaire : une connexion TCP peut vivre plus longtemps que l'application qui l'a créée, par exemple lorsque celle-ci ferme son point d'accès alors qu'il reste des paquets à échanger

1. Cette interface n'est pas complètement conforme à l'interface DLPI car celle-ci est prévue pour le modèle traditionnel où chaque stream entre IP et DLPI correspond à une interface réseau unique. Ce n'est plus le cas dans l'architecture démultiplexée et un champ a été ajouté à la primitive DL_DATA_REQ afin d'indiquer l'interface choisie par IP.

(données et/ou acquittements). Avec une implantation standard, le contexte TCP reste actif et traite les derniers paquets. Avec l'implantation démultiplexée, fermer l'accès démantèle le CC totalement. La solution consiste à déplacer le contexte vers un CC permanent qui prend en charge les éventuels paquets échangés après fermeture du CC.

Motivations pour l'utilisation de STREAMS

Bien qu'une communication par passage de messages (STREAMS) soit plus coûteuse qu'une communication par appels de fonctions (BSD), nous avons choisi la première solution. La raison est que la flexibilité de STREAMS est indispensable à notre architecture démultiplexée. La flexibilité est également un aspect crucial avec l'avènement de plate-formes multimédia. En effet, celles-ci utilisent différents types de protocoles, différents modules de manipulation de données, et se doivent de cohabiter avec le système de communication traditionnel. La flexibilité est un atout important qui permet de construire des CC adaptés à chaque type d'application.

2.3.1. Bénéfices de cette approche

Les impacts sur le chemin de données principal

L'architecture des composants d'un CC est simplifiée par la connaissance précoce (à l'ouverture) des protocoles utilisés et par la suppression de tout multiplexage. Cependant cette approche conduit aussi à un accès supplémentaire aux en-têtes (par le driveur d'ancrage). Globalement *l'approche démultiplexée déplace davantage certains traitements qu'elle ne simplifie le chemin de données principal*.

Les impacts sur le contrôle de flux local

Le contrôle de flux local est grandement amélioré. Le contrôle de flux STREAMS se faisant au niveau du stream, avoir un unique stream au sein du CC reliant une application au driveur réseau signifie qu'une application gourmande peut immédiatement être bloquée si le driveur est saturé (flux sortant). Cette technique *permet de créer un mécanisme de contrôle d'accès au médium par CC*.

L'algorithme est le suivant : si, lors d'une tentative de transmission, le driveur réseau indique qu'il est saturé, alors le driveur d'ancrage insère le paquet dans la queue d'écriture du CC. Ce CC devient bloqué. Le module de transport le détecte et évitera par la suite de créer de nouveaux segments. Enfin une temporisation est armée au sein du driveur d'ancrage afin de relancer plus tard tous les CC bloqués.

Réciproquement, après démultiplexage d'un paquet entrant, le driveur d'ancrage peut contrôler, par vérification de l'état de saturation du CC, si l'application suit le rythme. Il décide alors de l'attitude à adopter : faire remonter le paquet, jeter un paquet UDP en cas de saturation, etc. Dans tous les cas, ce choix est effectué tôt, avant tout traitement important.

Les impacts sur le support de la qualité de service

L'algorithme utilisé pour décider quand bloquer ou débloquer un CC peut être fondé sur des informations de saturation comme on vient de voir, sur un mécanisme de déblocage périodique (pour effectuer un contrôle de débit sur un flux UDP par exemple), ou bien sur des notions de priorité.

Montrons comment on peut construire un mécanisme de support de la QoS. Le driveur d'ancrage est à cet égard privilégié : il a accès à tous les CC et peut être étendu afin de connaître les besoins en QoS de chacun d'eux. On peut donc y implanter un mécanisme de contrôle d'accès de type CBQ (Class Based Queueing) [Wakeman95]. Ce mécanisme peut aussi être étendu aux différents flux de données qui traversent la machine lorsque celle-ci est utilisée en tant que routeur.

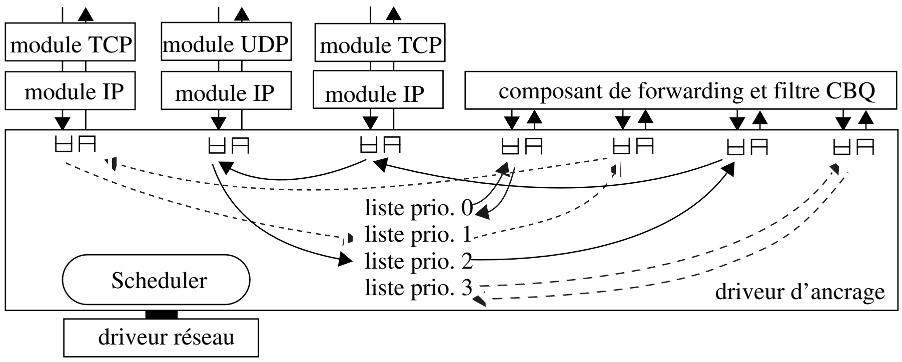

Figure 2: Extension QoS de la pile TCP/IP démultiplexée sous STREAMS.

La Figure 2 montre l'architecture obtenue. Les différentes queues sont chaînées en listes de priorité. Ce chaînage est dynamique pour les CC (modifié à chaque ouverture/ fermeture de CC), et statique pour le composant de "forwarding" (mis en place à la configuration). Le "scheduler" est un composant asynchrone qui, à la fin de chaque transmission sur le réseau, va choisir, en fonction des priorités, le prochain paquet à transmettre. Cette architecture traite ainsi de façon uniforme le cas où la machine possède une fonctionnalité routage (composant de "forwarding"/filtre CBQ présent), et le cas où elle constitue une station terminale (seuls les CC sont présents).

L'approche démultiplexée permet donc de bloquer automatiquement toutes les applications qui ne sont pas les plus prioritaires à un moment donné. *Agir sur le blocage des CC est ainsi un moyen aisé et efficace de contrôler flux par flux, l'allocation des ressources processeur, mémoire et médium.*

Nous n'avons décrit que le moteur du support système de la QoS. Des mécanismes connexes (politiques de scheduling, le contrôle d'admission, le contrôle du respect des engagements de QoS, etc.) sont nécessaires. Ils ne sont pas abordés dans ce papier.

2.3.2. Techniques complémentaires

Des techniques complémentaires sont utilisées pour résoudre certains problèmes :

Utilisation des queues et routines de service

La mise en queue des messages et les traitements asynchrones dans les routines de service ont été bannies du chemin de données principal. Au lieu de cela, dans TCP :

- un service de queues spécialisé, plus performant que la version STREAMS car plus simple, est utilisé pour les files de données entrantes et sortantes de TCP,
- les TSDUs sont traitées au sein des routines `put` afin d'éviter tout délai supplémentaire, et
- le blocage de l'application est réalisé en insérant un message dédié dans la queue d'écriture (message de taille supérieure à la taille de la queue) et à le retirer quand la condition de blocage disparaît.

Gestion des buffers de réception applicatifs

Afin de remédier au problème décrit à la Section 2.2, nous avons choisi d'utiliser une *option qui force le Stream-Head à remplir le buffer applicatif*. Cette solution simple n'est cependant pas portable (non définie dans [STREAMS90]), et en cas de trafic interactif où il n'est pas possible d'attendre la fin du remplissage du buffer, un message additionnel doit être envoyé au Stream-Head afin de l'en informer. Pour éviter ce dernier problème, il est possible de laisser à l'application le soin de décider ou non de l'utilisation de cette option.

2.4. Une méthode d'accès ILP

Nous montrons maintenant comment la gestion des buffers peut être simplifiée et comment la technique ILP peut être introduite dans une pile TCP/IP. Nous tirons profit de la copie de données entre espaces utilisateur et noyau pour effectuer le calcul de checksum. Si ce type d'intégration n'est pas récent ([RFC817, Clark89, Clark90] en évoque la possibilité), nous montrons ici comment le mettre en oeuvre sur un système ouvert, et quels sont les gains obtenus.

2.4.1. Limites des méthodes d'accès traditionnelles

Considérons une méthode d'accès traditionnelle utilisée avec un protocole de type “flot” tel que TCP, voyant les TSDUs comme une suite d’octets pouvant être segmentée comme bon lui semble. La gestion des buffers durant la copie entre espaces utilisateur et noyau est alors faite sans considérations pour les frontières des futurs segments (encore inconnues). Les conséquences sont multiples :

- un segment est souvent à cheval sur plusieurs buffers, ou ne constitue qu’une partie d’un buffer. Ceci complique les opérations de recherche, duplication, et libération de segments dans la liste de transmission de TCP : calculs de déplacement, dupliquer un segment chevauchant plusieurs buffers nécessite plusieurs descripteurs, un par buffer, etc.
- un segment pouvant commencer n’importe où dans le buffer, un buffer supplémentaire doit être alloué pour les en-têtes de protocoles.
- les frontières de segments n’étant pas encore connues durant la copie entre espaces utilisateur et noyau, il n’est pas possible d’intégrer cette dernière avec le calcul de checksum qui se fait sur la base d’un segment.

Ces limitations n’existent pas avec un protocole tel que UDP où chaque TSDU correspond à un datagramme. Un usage intelligent de cette caractéristique et le fait que UDP n’a pas à conserver de copies des données peut résoudre les problèmes précédents. Ainsi l’intégration copie/checksum a été ajoutée au driver UDP BSD d’AIX.

2.4.2. Méthode d’accès ILP

Il ressort que les limitations de la méthode d’accès dans le cas de TCP proviennent de la connaissance tardive des frontières de segments. Nous proposons donc de *déplacer la fonctionnalité de segmentation des TSDUs depuis TCP vers l’application* (librairie XTI dans notre cas), *au niveau utilisateur*. Cette technique consiste à identifier dans le flux de données de l’application les frontières de segments, et à contrôler durant l’appel système la façon dont les données sont copiées dans les buffers noyaux. Les avantages sont multiples :

- un segment correspond à un buffer noyau, ce qui simplifie les opérations de recherche, duplication et libération dans la liste de transmission de TCP,
- de la place est réservée dans les buffers noyau pour les en-têtes de protocoles,
- la copie de données peut être intégrée au calcul de checksum, et
- le nombre d’appels systèmes peut être considérablement réduit lorsqu’une application utilise de petites TSDUs et n’est pas préoccupée par la latence. La librairie attend alors d’avoir suffisamment de données pour remplir un segment de taille maximum.

D’un autre coté cette méthode d’accès crée un certain nombre de difficultés :

- TCP peut être amené à resegmenter les “segments” anticipés par la librairie (e.g. les segment d’un octet de sondage de la fenêtre distante [Stevens94]). Dans ce cas, un nouveau buffer est alloué, les données y sont copiées et le checksum recalculé. La règle “un segment, un buffer” est ainsi préservée.

- des copies de données supplémentaires peuvent être nécessaires dans la librairie (espace utilisateur). Cela survient lorsque le contrôle doit être retourné à l’application alors que les données n’ont pas encore été envoyées à TCP.

- en cas de TSDU isolée ou si la latence importe, l’application doit pouvoir s’affranchir de l’accumulation dans la librairie. Le positionnement du flag `PUSH` indique à la librairie de passer les données à TCP sans attendre.

- enfin un nouvel appel système, `putextmsg`, a été conçu. Il reconnaît les frontières de blocs de données (segments identifiés par la librairie), alloue les buffers noyau en conséquence, et réalise le copie/checksum. Il est important que plusieurs segments puissent être envoyés en un unique appel système.

Une caractéristique importante de cette technique est d’être *quasiment transparente aux applications* : tout est caché dans la méthode d’accès (librairie, Stream-Head, TCP/UDP).

3. Etude de performances

Nos mesures mettent en jeu deux types de transferts : les transferts de masse (type FTP) où l’émetteur essaie de saturer le système de communication, et les transferts en mode écho (type telnet) où les données envoyées sont retournées en écho.

Le buffer de réception de l’application est de 8 kilo-octets, et les fenêtres d’émission/réception de 16 kilo-octets. Chaque test dure entre 10 et 30 secondes et le point de mesure retenu est la moyenne sur cet intervalle. La plupart des tests ont été effectués en mode rebouclé mémoire. Enfin les machines utilisées sont des DPX/20 modèle 420 (POWER à 41,7 MHz), et le système d’exploitation est soit l’AIX/3.2.5, soit l’AIX/4.1.2 dans une version monoprocesseur.

3.1. Piles BSD et STREAMS classiques

La Figure 3 montre les performances d’une pile BSD et d’une pile STREAMS classique (Section 2.2). Les transferts se font au travers d’une connexion TCP en rebouclé mémoire (MSS, ou taille maximum de segments, de 1460 octets). La machine est un DPX/20 sous AIX/3.2.5.

⇨ Ces expériences soulignent les *performances limitées de la pile STREAMS clas-*

sique à la fois en terme de débit et de latence. La stratégie de gestion des buffers de réception de STREAMS en est largement responsable (Section 2.2). Le nombre accru d'appels systèmes et de changements de contextes justifie les médiocres performances en débit. Du point de vue latence, cette stratégie impose également un cycle `recv`/`send` supplémentaire à chaque frontière de MSS (un paquet supplémentaire est reçu). Ceci justifie la hauteur importante des marches sur la courbe de la pile STREAMS.

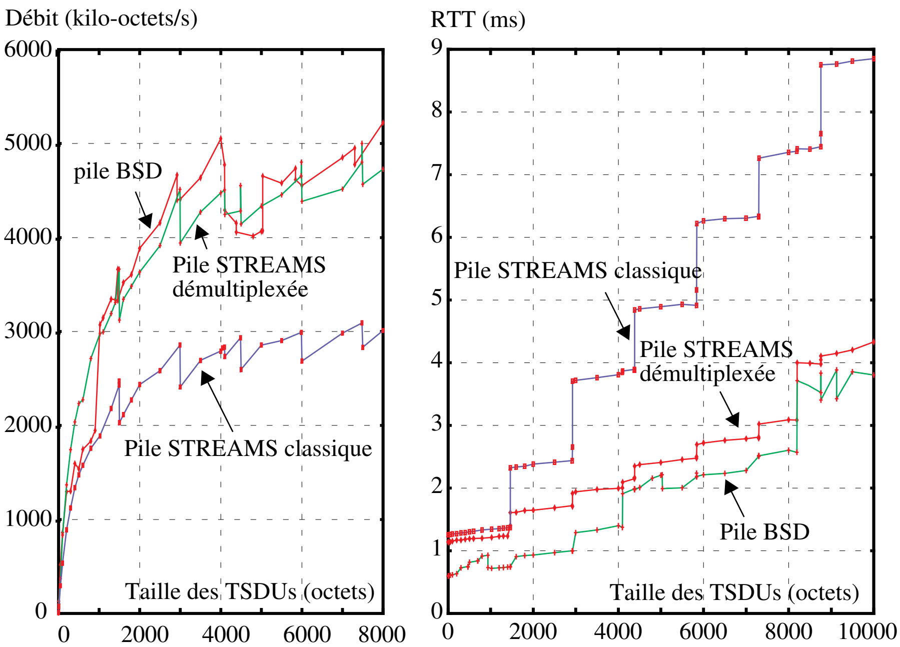

Figure 3: (gauche) Comparaison des débits durant des transferts de données de masse; (droite) Comparaison des RTTs durant des transferts en mode écho.

3.2. Pile TCP/IP STREAMS démultiplexée

3.2.1. Les expériences de base

Nous commençons avec des expériences qui ne tirent aucun profit direct de l'architecture démultiplexée de cette pile. Les conditions de test sont inchangées.

⇨ La Figure 3-gauche montre qu'*une implémentation STREAMS soigneusement conçue peut quasiment atteindre le même niveau de débit qu'une implémentation BSD* sous AIX/3.2.5. La pile BSD est pénalisée par sa gestion mémoire qui alloue jusqu'à quatre `mbufs` (228 ou 236 octets) pour les requêtes entre 1 et 936 octets, et un unique `cluster` de 4 kilo-octets au dessus. Le même phénomène se produit aux multiples de

4 kilo-octets. La pile démultiplexée ayant une meilleure gestion mémoire parvient à dépasser BSD dans ces zones.

⇨ En revanche la pile démultiplexée a un *comportement légèrement moins bon avec les tests de latence* (Figure 3-droite). Ceci est lié aux coûts plus élevés imposés par la communication par messages de STREAMS, et à la stratégie de remplissage des buffers applicatifs utilisée.

3.2.2. Analyse du contrôle de flux

Nous désirons maintenant apprécier l'opportunité d'avoir un mécanisme de contrôle de flux local bloquant l'application en cas de saturation de la carte d'adaptation ou du réseau (Section 2.3.1) (saturer un réseau Ethernet ou FDDI est devenu aisé).

Table 2: Débits en émission et réception pour les piles BSD et STREAMS démultiplexée sur Ethernet.

taille des TSDUs et protocoles		**pile TCP/IP classique BSD**		**pile TCP/IP démultiplexée STREAMS**	
		émetteur (kilo-octets/s)	**récepteur (kilo-octets/s)**	**émetteur (kilo-octets/s)**	**récepteur (kilo-octets/s)**
1024 octets	TCP	0	0	165	165
	UDP	4703	964	780	780
4096 octets	TCP	0	0	145	145
	UDP [a]	9782	0	851	851

a. Les datagrammes UDP étant fragmentés par IP, perdre un fragment conduit à perdre le datagramme complet.

Pour cela nous juxtaposons un flux UDP à une connexion TCP entre deux DPX/20 reliés par un réseau Ethernet privé, sous AIX/4.1.2. Si TCP peut s'adapter à la bande passante disponible grâce à ses mécanismes de contrôle de flux/débit, UDP ne le peut pas. La Table 2 montre que l'absence de contrôle de flux local (cas de BSD) conduit UDP à monopoliser toute la bande passante. Le fait que le débit UDP en réception ne soit qu'une fraction du débit UDP en émission montre également qu'une quantité significative des paquets est perdue. Ces deux problèmes sont évités dès lors que l'on introduit un contrôle de flux local (cas de la pile démultiplexée).

Ces essais peuvent paraître artificiels : UDP n'est pas adapté à un transfert de données de masse à moins qu'existe un mécanisme de rétroaction [Diot95]. Dans ce cas une cohabitation UDP/TCP est possible.

Cependant *l'intérêt majeur de ces tests est de valider le mécanisme de blocage/ déblocage des CC*. L'algorithme utilisé pour décider quand bloquer ou débloquer dépend ensuite du contexte (Section 2.3.1).

3.3. La méthode d'accès ILP

3.3.1. Expériences de haut niveau

Nous nous intéressons maintenant aux performances en débit et latence de notre pile démultiplexée avec ou sans méthode d'accès ILP, en utilisant une connexion TCP au dessus de l'interface rebouclée mémoire. Deux tailles de MTU (Maximum Transmission Unit) sont utilisées : 1500 (comme Ethernet), et 3500 octets.

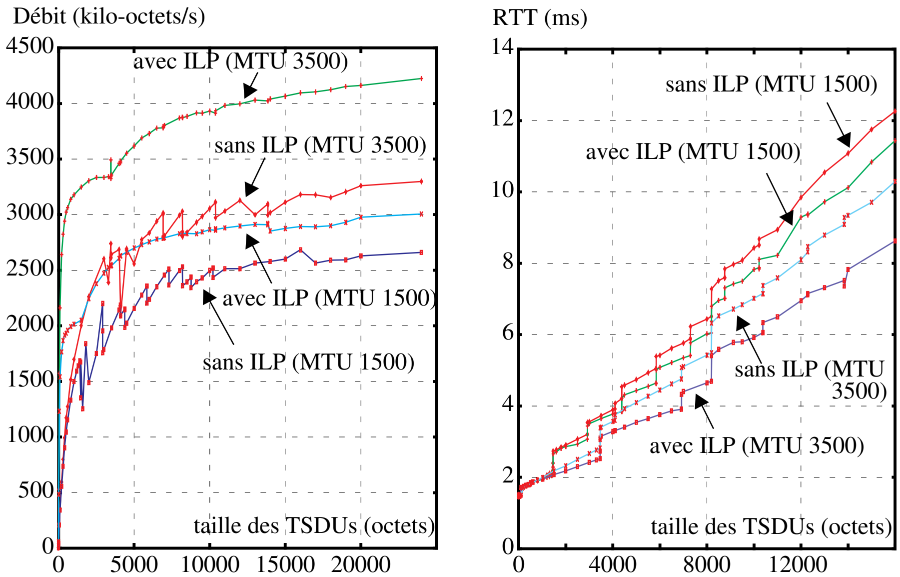

Figure 4: (gauche) Comparaison de débits avec ou sans méthode d'accès ILP; (droite) Comparaison de RTTs avec ou sans méthode d'accès ILP.

⇨ La Figure 4-gauche montre que *la méthode d'accès ILP permet des gains extrêmement élevés*, jusqu'à 400%, *pour de petites TSDUs*. C'est le résultat des accumulations de données dans la librairie XTI qui réduisent le nombre d'appels systèmes.

⇨ *Les gains asymptotiques pour de grosses TSDUs sont plus modestes*, de 13% pour une MTU de 1500 octets, à 28% pour une MTU de 3500 octets. Ces gains sont liés à l'intégration des manipulations de données et à l'amélioration de la gestion mémoire. L'accumulation de données dans la librairie n'a plus d'impact significatif.

⇨ *Les tests en mode écho* de la Figure 4-droite *révèlent des gains plus limités*. L'envoi immédiat des données en est la cause. Les gains ne sont dûs qu'à l'intégration copie/checksum et à la meilleure gestion mémoire. Ces tests montrent qu'utiliser la méthode d'accès ILP permet des réductions de 6%-8% du RTT au dessus de 5 kilo-octets avec une MTU de 1500 octets, et de 13%-16% avec une MTU de 3500 octets.

Les bénéfices de la technique ILP vont donc dépendre des caractéristiques de l'application. La méthode d'accès ILP est principalement bénéfique aux applications effectuant des transferts de masse. Les applications interactives utilisant de petits messages (telnet, rlogin, etc.) ne tireront que peu de bénéfices de cette technique.

La nature du réseau, via le paramètre MTU, est essentielle. La raison en est que la méthode d'accès ILP a des coûts fixes élevés. L'arrivée de nouvelles technologies réseau utilisant une MTU importante (la MTU par défaut d'IP sur ATM est de 9128 octets [RFC1577]) contribuera à rendre cette méthode d'accès plus intéressante.

3.3.2. Expériences de bas niveau

La Table 3 montre la distribution du nombre d'instructions exécutées par la pile démultiplexée (avec et sans méthode d'accès ILP) lors de la transmission d'un segment TCP de 1024 octets sur Ethernet (DPX/20 sous AIX/4.1.2).

Table 3: Nombre d'instructions exécutées sans ou avec méthode d'accès ILP.

	sans ILP		**avec ILP**	
	nb. instr.	**%**	**nb. instr.**	**%**
Niveau utilisateur (librairie XTI)	94	0.9	148	1.7
Méthode d'accès (Stream-head hors copie Utilisateur/Noyau, TIMOD)	1234	12.2	1373	16.1
Protocoles (TCP, IP et ARP)	871	8.6	943	11.1
Gestion mémoire (allocation/libération)	**1776**	17.6	**1407**	16.5
Manipulations données (copie U/N, checksum, copie vers driveur Ethernet)	**2037**	20.2	**950**	11.2
driveur Ethernet et gestion des ITs	1451	14.3	1426	16.8
Synchronisation (masquage ITs, verrous)	**1601**	15.9	**1277**	15.0
Primitives système de bas niveau	1027	10.2	993	11.7
Total	10091	100%	8517	100%

⇨ La Table 3 montre que la *principale contribution de la méthode d'accès ILP provient de l'intégration des manipulations de données* (1087 instructions).

⇨ Elle montre également qu'*avoir une meilleure gestion mémoire a un impact très important*. Pouvoir réserver de la place pour les en-têtes de protocoles et avoir exactement un segment par buffer sont bénéfiques à la gestion mémoire (moins d'allocations/libérations) et à la synchronisation (la gestion mémoire est gourmande en synchronisation). Une économie de 693 instructions est faite, soit plus de la moitié de celle de l'intégration ILP.

⇨ En comparaison l'augmentation des traitements au sein de la méthode d'accès est limitée. La nécessité de contrôler précisément les transferts de données lors du `putextmsg` (Section 2.4) ne représente que 265 instructions. Ce coût, quasiment fixe, justifie le fait que la méthode d'accès ILP devienne plus attractive lorsqu'augmentent la taille des TSDUs et la MTU.

4. Conclusions

Nous avons montré que l'architecture de la pile de communication est essentielle. Une architecture démultiplexée, avec des canaux de communication directs entre applications et driveurs réseaux, est très largement bénéfique. La possibilité de blocage/déblocage de ces canaux permet un *excellent contrôle de flux local, contrôle de débit, et gestion au niveau système de la qualité de service*. Grâce à ces mécanismes, la gestion des ressources principales de la machine (mémoire, processeur, et médium) est rendue à la fois simple et efficace.

Nous avons également montré comment la technique ILP, destinée à limiter les accès mémoire, peut être intégrée à la méthode d'accès d'une pile TCP/IP. Les bénéfices de cette technique restent cependant largement tributaires de ses conditions d'utilisation et de la nature des applications. La méthode d'accès ILP doit donc rester une alternative. Un autre résultat est l'importance de la MTU. L'arrivée de nouvelles technologies réseau contribuera à rendre cette méthode d'accès plus attractive. Enfin, la nature des manipulations de données est essentielle. [Braun95] a montré que trop de complexité nuit à l'efficacité d'ILP. Nous nous sommes affranchis de cette limitation en utilisant des manipulations de données très simples.

Enfin la flexibilité offerte par STREAMS s'est avérée essentielle. Son importance va tendre à s'accentuer avec l'avènement de systèmes multimédia. Une fois résolu certains problèmes (limitations fonctionnelles et mauvais usage), cet environnement est devenu une base de développement très attractive.

5. Bibliographie

[Braun 95] T. Braun, C. Diot, "Protocol implementation using Integrated Layer Processing", ACM SIGCOMM'95, août 1995.

[Clark 90] D.D. Clark, D.L. Tennenhouse, "Architectural considerations for a new generation of protocols", ACM SIGCOMM'90, septembre 1990.

[Diot 95] C. Diot, C. Huitema, T. Turletti, "Multimedia applications should be adaptative", Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95), août 1995.

[Mosberger95] D. Mosberger, L. Peterson, S. O'Malley, "Protocol latency : MIPS and reality", Technical Report TR 95-02, University of Arizona, 1995.

[RFC 817] D. Clark, "Modularity and efficiency in protocol implementation", Request For Comments, juillet 1982.

[RFC 1577] M. Laubach, "Classical IP over ATM", Request for Comments, janvier 1994.

[RFC 1633] R. Braden, D. Clark, S. Shenker, "Integrated services in the Internet architecture : an overview", Request For Comments, juin 1994.

[Roca 94] V. Roca, C. Diot, "A high performance STREAMS-based architecture for communication subsystems", fourth IFIP Workshop on Protocols For High Speed-Networks (PfHSN'94), Vancouver, Canada, août 1994.

[Stevens 94] W.R. Stevens, "TCP/IP illustrated volume 1 : the protocols", Professional Computing Series, Addison-Wesley Publishing Company, 1994.

[STREAMS 90] "UNIX SVR4; programmer's guide : STREAMS", Prentice Hall, 1990.

[Wakeman 95] I. Wakeman, A. Ghosh, J. Crowcroft, V. Jacobson, S. Floyd, "Implementing real time packet forwarding policies using STREAMS", USENIX, janvier 1995.

6. Biographie

Vincent Roca est enseignant/chercheur contractuel à l'ENSIMAG (Grenoble). Ses thèmes de recherche concernent les techniques d'implantation hautes performances des systèmes de communication et le support du multimédia.

Christophe Diot est chercheur à l'INRIA Sophia Antipolis. Il s'intéresse plus particulièrement aux problèmes d'architecture des systèmes de communication hétérogènes tels qu'Internet, ainsi qu'à la garantie de service sur ce type de réseaux.