HAL
open science

Étude et réalisation d'une plateforme de communications intégrant le protocole pour le réseau à haut débit XTP

Michèle Boutet

To cite this version:

Michèle Boutet. Étude et réalisation d'une plateforme de communications intégrant le protocole pour le réseau à haut débit XTP. Réseaux et télécommunications [cs.NI]. 1993. dumas-00364933

HAL Id: dumas-00364933

https://dumas.ccsd.cnrs.fr/dumas-00364933v1

Submitted on 27 Feb 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

CONSERVATOIRE NATIONAL
DES ARTS ET METIERS
CENTRE AGREE DE GRENOBLE (C.U.E.F.A)

MEMOIRE

présenté en vue d'obtenir

LE DIPLOME D'INGENIEUR C.N.A.M.

en

INFORMATIQUE

par

Michèle BOUTET

Etude et réalisation d'une plateforme de communications intégrant le protocole pour réseau à haut débit XTP

Les travaux relatifs au présent mémoire ont été effectués dans le cadre de l'unité architecture de systèmes de communications de Bull sous la direction de Monsieur Michel HABERT.

Je remercie,

Monsieur Claude Kaiser, Professeur au Conservatoire National des Arts et Métiers, qui me fait l'honneur de présider le jury de ce mémoire.

Monsieur Alain Cazes, Maitre de Conférences au Conservatoire National des Arts et Métiers, d'avoir accepté de participer à ce jury.

Monsieur Jacques Courtin, Professeur à l'Université Pierre Mendès France de Grenoble, Responsable du Cycle Informatique du Conservatoire National des Arts et Métiers de Grenoble, qui m'a suivie et conseillée tout au long de ma formation.

Monsieur Michel Habert, Responsable de l'Unité Architecture de Systèmes de Communications, pour sa disponibilité et la qualité de son encadrement.

Monsieur Maurice Testa, Responsable de l'Equipe Protocoles Internet du Département Communications Unix, pour m'avoir accueillie dans son équipe.

Aimé Lerouzic, pour son soutien et son aide dans les différentes phases de ce travail.

Vincent Roca, pour sa collaboration amicale et fructueuse tout au long de ce travail.

Mohammed Hendaz et Kaïs Belgaïed pour leur participation à ce projet.

RESUME

Ce projet s'inscrit dans le cadre des réseaux hauts débits. Le but poursuivi est de réaliser une plateforme intégrant à la fois un protocole d'ancienne génération, TCP, et un nouveau protocole XTP ou eXpress Transport Protocol, qui a été conçu dès le départ pour répondre aux besoins actuels et futurs des applications, en termes de débit et de fonctionnalités.

Posséder une plateforme intégrant de façon similaire ces deux protocoles permettra ensuite de réaliser des expérimentations sur XTP et d'apprécier l'opportunité d'une éventuelle migration de TCP vers XTP.

Dans ce mémoire, le système de communications de la plateforme est d'abord introduit, en détaillant chacun de ses éléments. Puis, l'intégration du protocole XTP dans ce système de communications est présentée. Enfin, les protocoles sont comparés à la fois au niveau des performances et des fonctionnalités.

MOTS-CLES

réseaux, protocoles de communication, XTP, TCP/IP, étude de performances, intégration de protocoles, streams.

KEYWORDS

network, communication protocol, XTP, TCP/IP, performance studies, protocol integration, streams.

Chapitre 1 :
INTRODUCTION

1.1 L'unité de communications avancées 1–1

1.2 Situation technique du problème 1–1

1.3 Objectifs du projet 1–2

1.4 Travail réalisé autour du protocole XTP 1–3

1.5 Plan du mémoire 1–4

Chapitre 2 :

COMPOSITION DE LA PLATEFORME

2.1 Introduction 2-5

2.2 Le système de communication 2-5

2.2.1 Les protocoles 2-6

2.2.1.1 Transport des données : le protocole TCP 2-7

2.2.1.2 Adressage et routage : le protocole IP 2-12

2.2.1.3 Liaison des données : ETHERNET 2-16

2.2.2 Les STREAMS 2-16

2.2.3 Les interfaces 2-18

2.2.3.1 TPI et DLPI 2-18

2.2.3.2 La librairie XTI 2-18

2.2.4 Une application: le programme BENCH 2-21

2.3 Le protocole XTP 2-22

2.3.1 Le paquet XTP 2-23

2.3.2 connexion 2-24

2.3.3 routage 2-26

2.3.4 déconnexion 2-26

2.3.5 segmentation 2-28

2.3.6 contrôle d'erreurs 2-29

2.3.7 contrôle de flux 2-30

2.4 Les apports du protocole XTP 2-31

2.5 Conclusion 2-32

Chapitre 3 :
REALISATIONS

3.1 Introduction 3–35

3.2 Organisation logicielle et méthodologie 3–36

3.2.1 Production de programme 3–36

3.2.1.1 Arborescences parallèles 3–36

3.2.1.2 Organisation de l'arborescence 3–37

3.2.2 Exploitation des communications 3–39

3.2.2.1 Composants spécifiques de la partition système . 3–39

3.2.2.2 Séquencement des opérations du chargement d'Unix et des communications 3–39

3.2.3 Mise au point d'un driver du noyau 3–41

3.3 Implémentation choisie 3–41

3.3.1 Contrôle de flux des streams 3–42

3.3.2 La tête du stream 3–44

3.3.3 Le module timod 3–44

3.3.4 Le driver XTP 3–45

3.3.5 Organisation des buffers 3–47

3.3.6 Adressage 3–48

3.4 Stratégie adoptée 3–48

3.4.1 Première phase : mise en place d'un "squelette" . . . 3–48

3.4.2 Création du prototype 3–51

3.4.2.1 XTP sur le driver loopback. 3–52

3.4.2.2 XTP sur Ethernet : réseau privé 3–53

3.4.2.3 Résultats 3–54

3.4.3 Améliorations 3–55

3.4.3.1 Optimisation du flux de données 3–55

3.4.3.2 Politique d'acquittement 3–59

3.4.3.3 Résultats 3–61

3.4.3.4 Fonctionnalités restant à implémenter 3–62
3.4.4 Extension de la librairie XTI 3–62
3.4.4.1 Profil XTI pour XTP 3–62
3.4.4.2 Impact sur TPI 3–65
3.4.4.3 Faisabilité 3–65
3.5 Conclusion 3–65

Chapitre 4 :

ETUDE COMPARATIVE DE TCP ET XTP

4.1 Introduction 4–65

4.2 Fonctionnalités de la couche de transport 4–65

4.2.1 Gestion de la connexion 4–66

4.2.2 Segmentation et ré–assemblage 4–66

4.2.3 Contrôle des erreurs 4–66

4.2.4 Contrôle de flux 4–68

4.2.5 Implémentations des services transport 4–68

4.3 Mesures et analyse des résultats obtenus. 4–69

4.3.1 Analyse des résultats 4–70

4.3.2 Position relative des deux courbes 4–71

4.3.3 Mesures sur une configuration matérielle plus performante 4–71

4.4 Comparaison des protocoles 4–72

4.4.1 Source–destination 4–73

4.4.2 Organisation des traitements 4–74

4.4.3 Structure des paquets 4–74

4.4.4 Détection des erreurs 4–75

4.4.5 Correction des erreurs 4–76

4.4.6 Prévention des erreurs 4–77

4.4.7 Performance 4–79

4.4.8 Améliorations 4–80

4.5 Conclusion 4–81

Chapitre 5
CONCLUSIONS

5.1 **Motivations** 5–81

5.2 **Services et performances** 5–82

5.3 **Faut–il changer de protocole de transport ?** 5–82

5.4 **Perspectives** 5–83

BIBLIOGRAPHIE

Annexes :

1 – GLOSSAIRE

2 – PRESENTATION DE KRM

3 – AUTOMATE D'ETATS FINIS DE TCP

Chapitre 1

INTRODUCTION

1.1 L'unité de communications avancées

Le projet a été réalisé à Bull, au sein de l'unité horizontale architecture de systèmes de communications SPO/Unix Medium System Organisation, DCO.

Cette unité a été créée en août 1991 pour étudier et contribuer à l'architecture et au développement des quatre premiers niveaux du modèle de référence OSI de l'ISO. Plus particulièrement, des travaux sur des réseaux à haut débit sont menés pour tenir compte de l'évolution rapide et importante dans ce domaine.

1.2 Situation technique du problème

Au cours des dernières années, les besoins des utilisateurs en matière de réseaux se sont amplifiés et modifiés. Ces nouveaux besoins sont notamment liés à l'émergence des systèmes distribués basés sur le modèle client/serveur DCM.

Ces systèmes permettent le développement d'applications utilisant des appels de procédures distantes (RPC) ou le mode diffusion (multicast), mode qui permet d'envoyer un même message simultanément à plusieurs postes.

Ces systèmes permettent aussi le développement des applications temps réel qui vont pouvoir bénéficier des nouveaux services isochrones des réseaux.

Parallèlement, la technologie des transmissions a connu une évolution considérable. Dans le domaine des réseaux locaux (LAN) et des réseaux métropolitains (MAN), des débits supérieurs à 100 Mbits/s sont atteints.

Depuis 1988, les travaux du CCITT ont convergé vers le développement de spécifications pour le BISDN (réseau digital de services intégrés à large bande). Le BISDN supportera une gamme de services divers (voix, vidéo, données) via des connexions multimedia integrées accessibles par une interface utilisateur simple. Les services nécessitant des successions de débits de magnétudes supérieures à celles délivrées par ISDN (tels que les services relatifs à la vidéo) deviennent possibles avec BISDN. Il devra supporter à la fois des services interactifs et des services de distribution (bulk).

Ainsi, dans le cadre du BISDN, la technologie SDH/ATM (transmission temporelle asynchrone) permettra d'atteindre des débits de l'ordre de 600 Mbits/s dans les réseaux étendus (WAN) et locaux.

1.3 Objectifs du projet

Dans ce contexte, les principaux protocoles de transports actuels comme ISO TP4 et TCP peuvent devenir inadaptés. En effet, ces protocoles sont particulièrement lourds à implémenter et ils nécessitent le traitement de grandes quantités de données (multiples contrôles d'erreurs), ce qui alourdit leurs performances.

En fait, ils conviennent parfaitement aux réseaux actuels à fort taux d'erreurs et dont le débit varie entre 9600 bits/s et 10 Mbits/s, pour lesquels le temps passé à effectuer des contrôles reste marginal au vu du temps passé dans le réseau.

Or, les améliorations sensibles des couches basses en termes de performances (meilleurs débits) et de fiabilité (plus faibles taux d'erreurs) ont déplacé le "goulot d'étranglement" vers la couche Transport, qui se situe à une position clé entre les couches orientées transmission de l'information et les couches orientées traitement de l'information. C'est à ce niveau que se situe maintenant le nouveau goulot d'étranglement que constituent les temps de traitement des paquets dans les calculateurs, temps de traitement qui empêchent ainsi une utilisation optimale des bandes passantes disponibles.

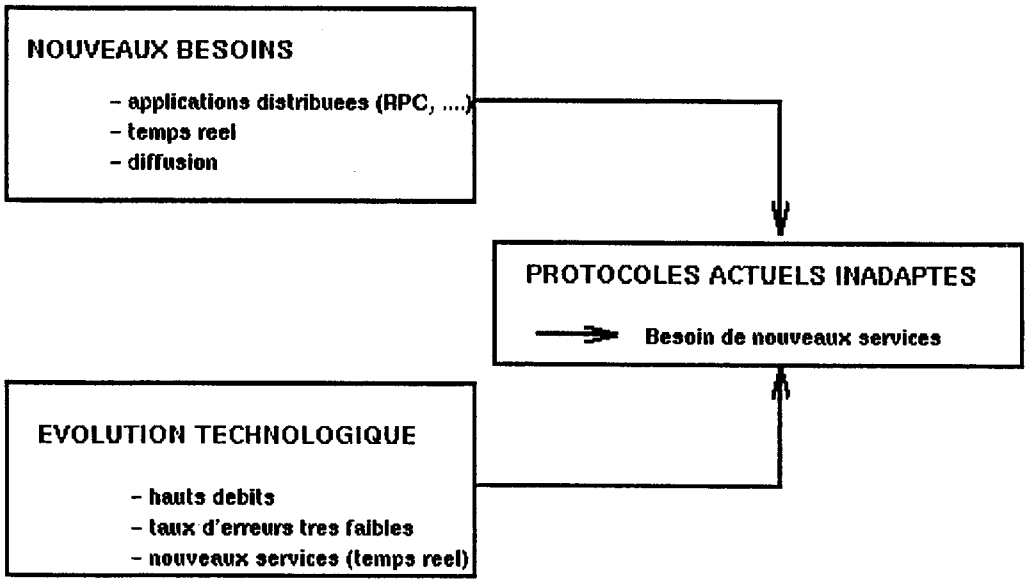


Fig. 1.1 nouveaux besoins et évolution technologique

Dès lors, il devient nécessaire de chercher à minimiser les délais engendrés par les protocoles de transport, en fournissant de nouveaux services et en améliorant les services existants, de façon à réduire dans les calculateurs les temps passés à la transmission de l'information.

1.4 Travail réalisé autour du protocole XTP

Dérivé du protocole temps réel de l'armée française GAM103, XTP a été repris par une compagnie américaine Protocol Engines Incorporated dont l'objectif est l'implémentation de XTP dans des circuits VLSI (circuits à très haute intégration).

Etant très récent (sa première implémentation en langage C date de 1988), les spécifications du protocole XTP ne sont pas encore définitives. Des études comportementales sont donc nécessaires afin d'ajuster les divers mécanismes mis en jeu.

Les performances obtenues dans une implantation donnée dépendant tout autant des qualités de l'architecture cible et de l'implémentation logicielle que des qualités intrinsèques du protocole, il est indispensable de mener parallèlement des études concernant les techniques d'implémentation et l'architecture des machines supportant ces implémentations.

C'est dans ce contexte que s'inscrit le projet : partant d'une implémentation logicielle standard du protocole XTP, nous devons réaliser une adaptation satisfaisant aux contraintes d'un sous–système de communication STREAMS d'une machine Unix.

1.5 Plan du mémoire

Ce mémoire est organisé en trois parties.

La première partie est consacrée à la présentation des différents composants logiciels de la plateforme.

La deuxième partie présente la stratégie adoptée pour la constitution de la plateforme ainsi que ses différentes étapes.

La troisième partie étudie la comparaison des performances obtenues sur XTP avec celles de TCP/IP.

Pour conclure, nous rappelons les résultats obtenus. Puis les perspectives sont ensuite présentées, ainsi que les points qui nécessiteront des expérimentations complémentaires.

Le monde des réseaux informatiques utilisant un vocabulaire spécifique, on trouvera parmi les annexes, un glossaire des principaux termes utilisés.

Chapitre 2

COMPOSITION DE LA PLATEFORME

2.1 Introduction

Dans les systèmes Unix ATT, les communications s'appuient sur des composants spécifiques (les protocoles) et sur une infrastructure (les STREAMS), l'assemblage de ces composants a conduit à la définition d'interfaces.

Ce chapître présente les différents éléments du système de communication initial ainsi que le protocole que nous y avons introduit. L'objectif est de donner des éléments suffisants pour la compréhension de l'élaboration de la plateforme et des problèmes rencontrés pour sa mise en œuvre.

2.2 Le système de communication

Dans une machine un système de communications est constitué de piles de protocoles. On distingue les communications horizontales de ces protocoles (entre plusieurs sous-systèmes) et les communications verticales (à l'intérieur d'un sous-système).

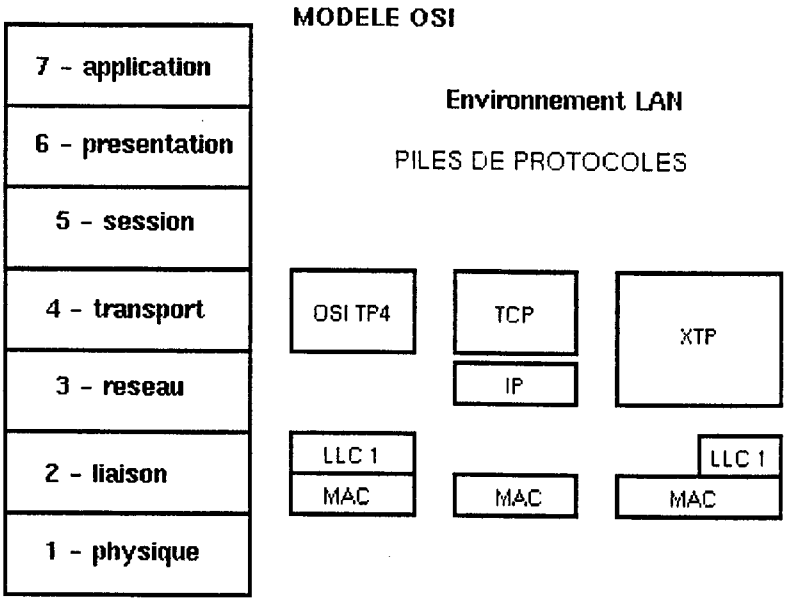


Fig. 2.1 piles des protocoles en environnement LAN

Par exemple, TCP, le protocole de transport (couche 4) d'une machine donnée dialogue avec les protocoles TCP des machines distantes, mais pour cela il communique avec IP (couche 3). A son tour IP, après avoir effectué sa part de traitements, encapsule les données venant de TCP et transmet le nouveau message à la couche voisine au niveau inférieur. Enfin la couche physique transmet le message résultant des encapsulations successives à la machine distante via le réseau. Sur la machine distante, les données sont "désencapsulées" en remontant de la couche physique vers la couche transport, TCP, qui est seul capable de décoder le message initial et d'effectuer le traitement approprié. Les autres couches ignorent le contenu du message initial. Elles ont seulement besoin de connaître son adresse de destination.

Ainsi, chaque couche doit pouvoir communiquer avec chacune de ses couches adjacentes, inférieures et supérieures, en complément de l'exécution de ses propres fonctions. La distinction entre ces deux familles de fonctions justifie l'organisation logicielle du système de communication (protocoles et infrastructure).

2.2.1 Les protocoles

La couche transport est chargée, une fois la liaison établie, d'effectuer les contrôles nécessaires au transport des informations de bout en bout. Il a pour rôle d'offrir des procédures de transmission bout à bout, c'est à dire pour lesquelles le réseau est totalement transparent.

Le réseau consiste, dans le cas le plus général, en une interconnexion de sous–réseaux, ces interconnexions étant réalisées grâce à des ponts ou des routeurs. La couche réseau est la seule à avoir une vision complète du réseau.

La couche liaison de données assure le transfert des données entre les nœuds successifs du réseau.

Dans la pile des protocoles DARPA (Department of Defense) la couche transport TCP est associée à IP qui réalise adressage et routage. Le support des transmissions que nous avons choisi d'utiliser est ETHERNET.

2.2.1.1 Transport des données : le protocole TCP

TCP est le plus complexe de tous les protocoles du monde INET. Il assure un service de bout en bout fiable en établissant un circuit virtuel pour le transport des données entre les applications. Il réalise :

- l'administration de la connexion : initialisation, maintenance et fermeture.
- le contrôle de l'intégralité des données transférées au moyen d'un échange d'accusés de réception (ACK),
- le contrôle de flot grâce à un mécanisme de fenêtres définissant dynamiquement la capacité des interlocuteurs.

Segments TCP et états de l'automate

<table>
<tr><td colspan="7">port source</td><td colspan="2">port destination</td></tr>
<tr><td colspan="9">numero de sequence du premier octet de donnees</td></tr>
<tr><td colspan="9">numero de sequence des donnees acquittees</td></tr>
<tr><td>offset donnees</td><td>reserve</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="2">fenetre</td></tr>
<tr><td colspan="7">checksum</td><td colspan="2">pointeur donnee urgente</td></tr>
<tr><td colspan="8">options</td><td>padding</td></tr>
<tr><td colspan="9">donnees</td></tr>
</table>

Fig. 2.2 format de l'en–tête TCP

Une connexion TCP peut être considérée comme un flot de données bidirectionnel respectant un certain séquencement circulant entre deux protocoles pairs. L'ouverture et la fermeture de la connexion sont des évènements explicites délimitant une séquence à l'intérieur de laquelle les données transitent à intervalles irréguliers et par paquets de taille variable. Leur position dans le flot est définie par un numéro de séquence. Les acquittements portent le numéro de séquence de la prochaine donnée à recevoir. Un segment peut ne pas contenir de donnée mais il contient toujours un numéro de séquence et l'acquittement de tous les segments contigüs reçus.

Pour le contrôle de flot, chaque segment véhiculant un acquittement, contient aussi la valeur d'une fenêtre qui indique le nouveau nombre d'octets acceptables en réception par l'émetteur du segment (à partir du numéro de séquence spécifié dans le champ "numéro de séquence des données acquittées").

Etats finis de TCP

Conceptuellement, TCP utilise un automate d'états finis pour contrôler toutes les interactions. Les transitions correspondent aux transferts de segments.

De plus, le traitement de sortie utilise lui–même un automate d'états finis (4 états) que l'on peut assimiler à des micro–transitions à l'intérieur d'un même état de l'automate de TCP. On trouvera en annexe l'automate d'états finis de TCP.

Ouverture de connexion

Elle nécessite l'échange de trois paquets :

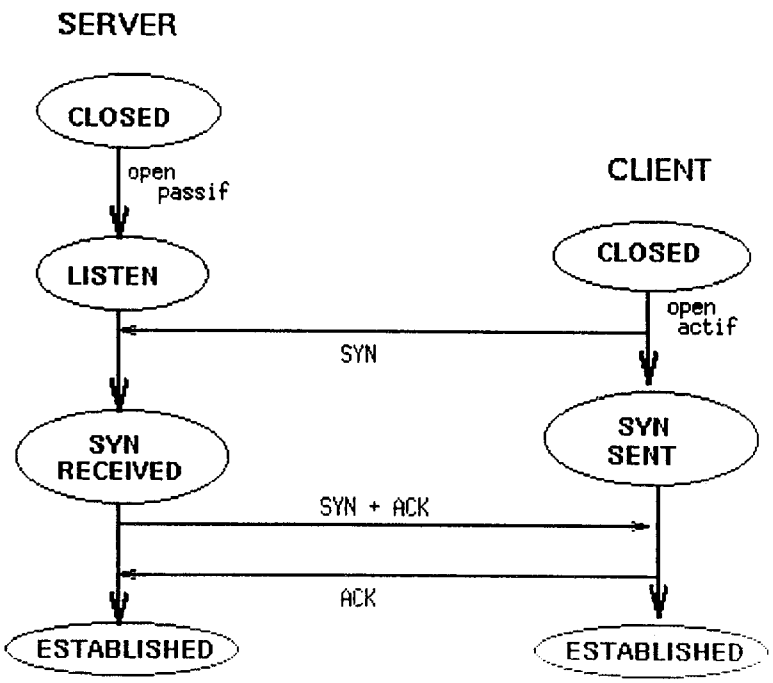


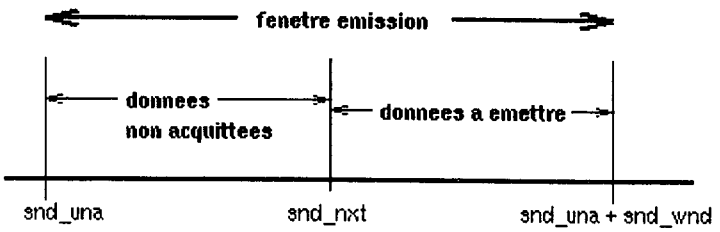
Fig. 2.3 séquence d'ouverture de connexion

Transferts des données

Les valeurs numéro de séquence des données acquittées et fenêtre permettent de définir 2 espaces de travail (l'un pour l'émission l'autre pour la réception).

En réception, TCP diminue la taille de sa fenêtre quand il reçoit des données et l'augmente quand il les délivre à son processus utilisateur. En émission, il conserve une copie des segments qu'il envoit jusqu'à ce qu'il en reçoive un acquittement. Si aucun ACK ne lui est parvenu passé un certain délai, il réitère son envoi.

Les segments dupliqués sont rejetés à la réception mais acquittés. Si des segments arrivent dans le désordre, le récepteur les conserve jusqu'à ce qu'il ait reçu les segments manquants.

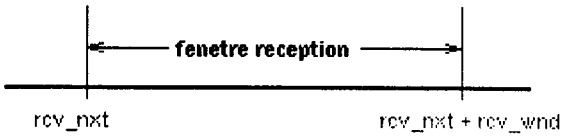


ESPACE DE TRAVAIL EMISSION

snd_una : numero de sequence premiere donnee emise non acquittee

snd_nxt : numero de sequence prochaine donnee a emettre

snd_wnd : fenetre emission



ESPACE DE TRAVAIL RECEPTION

rcv_nxt : numero de sequence attendu

rcv_wnd : fenetre reception

Fig. 2.4 espaces de travail

Fin de connexion

Chacun des interlocuteurs peut terminer une émission de données en positionnant l'indicateur FIN dans son dernier paquet émis et continuer de recevoir des données; à la réception d'un FIN (qu'il acquitte) la connexion est fermée.

Quand une connexion est perdue à la suite d'un abandon ou d'un timeout sur l'un des deux interlocuteurs, dès qu'il y a émission de données du côté actif, le récepteur envoit un segment RST avec pour numéro de séquence la valeur du segment reçu, signalant ainsi que la connexion est partiellement fermée et qu'il faut la ré–initialiser.

Transmission control block (TCB)

Chaque connexion maintient un jeu important de variables d'états dans un bloc de contrôle (TCB) : état de la connexion, timers, options, indicateurs divers, une file réservée aux données reçues dans le désordre et quelques variables de numéro de séquence. Ces variables permettent de délimiter l'espace de travail pour l'émission et la réception.

Processus timer (contrôle des délais)

TCP utilise quatre timers: deux timers pour l'émission, un pour la surveillance de la connexion et le dernier pour clore la connexion.

Timer de retransmission : il est armé quand on émet des données et arrété quand elles sont acquittées. Si ce timer expire, TCP ré–envoit les données non acquittées. La valeur de ce timer est calculée à partir du "round trip time" (moyenne calculée sur le temps nécessaire pour recevoir l'acquittement de segments individuels) ce paramètre permet d'adapter le timer de retransmission au réseau utilisé.

Timer "persiste" : c'est le deuxième timer pour l'émission. Il est utilisé pour détecter la perte d'autres types de segments (mise à jour des fenêtres) qui pourrait provoquer l'engorgement d'une connexion.

Le timer de surveillance contrôle les connexions au repos (détection de crash ou de time out des clients afin de libérer le server). Quand ce timer expire TCP envoit un paquet "keep alive"; s'il reçoit un segment RST, il peut fermer la connexion.

Enfin le dernier timer appelé 2MSL (deux fois la durée de vie maxima d'un segment) est utilisé en fin de connexion après l'envoi du dernier segment FIN.

Processus de traitement des segments en entrée

A la réception d'un segment, TCP analyse son en–tête pour déterminer les actions à réaliser.

– Recherche d'une TCB associée à la connexion; s'il n'y en a pas il envoit un segment RST.

– Examen des paramètres locaux de fenêtrage par rapport au numéro de séquence du segment reçu afin d'écarter les données déjà reçues.

– Etude de l'indicateur ACK pour la mise à jour de sa file d'émission (retrait des données acquittées), et de sa fenêtre d'émission.

– Si l'indicateur URG est positionné, il compare "urgent pointer" courant avec le précédent. S'il est différent, TCP extrait l'octet ainsi identifié et le place dans le TCB.

– Examine si les données en cours sont contigües aux données précédentes. Dans ce cas elles sont immédiatement mises dans le buffer réception. Sinon les données sont stockées dans une file intermédaire et ne sont pas acquittées.

– Enfin TCP étudie si une fin de connexion est demandée (indicateur FIN).

Les données sont délivrées à l'utilisateur sur sa demande.

Processus de traitement des segments en sortie

Afin de minimiser le traffic sur le réseau, les algorithmes utilisés pour l'émission des segments constituent la partie la plus subtile de l'implémentation de TCP.

Le processus d'émission de TCP est appelé quand l'utilisateur a écrit de nouvelles données. Si le contrôle de flot le permet, TCP crée l'en–tête pour envoyer le segment immédiatement. Il alloue un buffer pour l'en–tête. Les données à envoyer sont alors ajoutées sous la forme d'une chaine de buffers. Le numéro de séquence est initialisé avec la valeur "snd_nxt" pris dans le TCB et le champ "acknowldgement number" prend la valeur de "rcv_nxt". Les indicateurs (FLAGS) sont initialisés à partir d'un tableau récapitulatif des différents états de la connexion. Le champ fenêtre est calculé à partir de l'espace disponible dans le buffer de réception. Si une donnée urgente doit être envoyée, les champs correspondants sont positionnés. Enfin TCP initialise le checksum et met à jour les variables snd_nxt et snd_max pour cette connexion, dans le TCB.

Le processus d'émission de TCP optimise les transferts sur le réseau en interdisant des fenêtres trop petites : pour des valeurs inférieures à la taille d'un paquet et au quart du buffer réception, TCP indique une valeur de fenêtre nulle. De plus TCP limite l'envoi de petits paquets : il autorise l'émission du premier octet seul, calcule le round trip time pour émettre les suivants. Ce qui a pour effet de laisser s'accumuler les données courtes dans une file, le temps du round trip time et de pouvoir les émettre toutes en un seul segment.

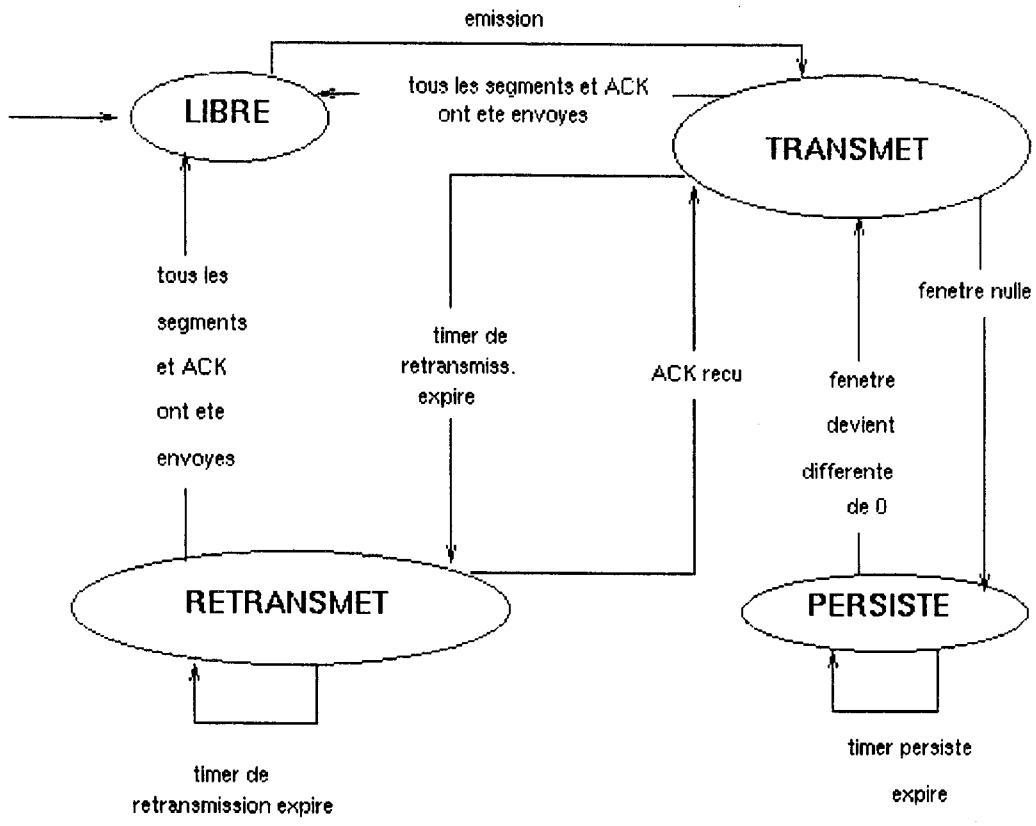


Fig. 2.5 états d'une transmission

De plus pour éviter la congestion du réseau, TCP utilise une fenêtre de congestion calculée dynamiquement et gérée séparément. Ainsi le nombre de segments en attente d'émission varie en fonction de la limite acceptable par le réseau.

2.2.1.2 Adressage et routage : le protocole IP

IP fournit les services de transaction internet pour la couche transport. Son rôle essentiel est d'acheminer des "datagrammes" (blocs de données comportant un en–tête avec des informations d'acheminement) d'une machine à une autre .

Le protocole IP assure la fragmentation et le ré–assemblage des blocs d'unités de données pour s'affranchir des limites imposées par la couche liaison de données. Il assure le routage vers la machine destinataire du datagramme, et traduit l'adresse Internet (adresse logique) fournie par le transport en adresse Ethernet (adresse physique). A la réception d'un datagramme, il le transmet au protocole destinataire (démultiplexage vers TCP–UDP).

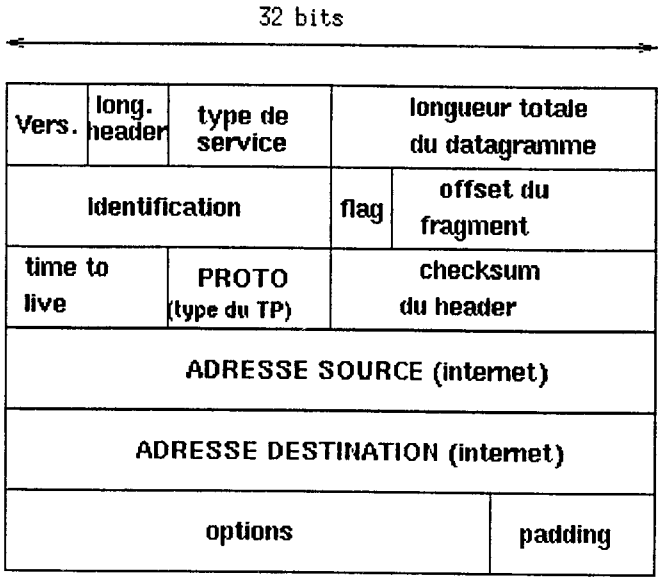


Fig. 2.6 En–tête du protocole IP

IP manipule des données (paquets de la couche transport), il crée ou exploite un entête qui contient des informations spécifiques à la couche réseau.

Organisation logicielle de IP

Intuitivement on peut considérer deux grandes fonctions:

– en entrée : utilisation du champ "PROTO" de l'en–tête pour déterminer le protocole destinataire.

– en sortie : utilisation d'une table de routage pour transférer le datagramme sur le réseau.

L'existence de gateways (passerelle entre deux réseaux), où il y a interaction entre les entrées et les sorties conduit à une définition plus générale basée sur trois techniques d'organisation:

– uniformité du routage et des queues : les datagrammes à traiter transitent par des files d'attente identiques quelque soit leur source (réseau ou protocole transport de la machine locale). Le traitement du routage procède par extraction d'un datagramme d'une file d'attente et application d'un algorithme de routage sans se préoccuper de la source.

– indépendance du processus IP

– interface local ou pseudo–interface: elle a la même structure que les interfaces physiques mais il correspond au protocole du transport.

L'algorithme de IP définit le routage de chaque datagramme et le passe à l'interface adéquat, s'il s'agit du pseudo–interface celui–ci réalise le démultiplexage vers les protocoles de transport.

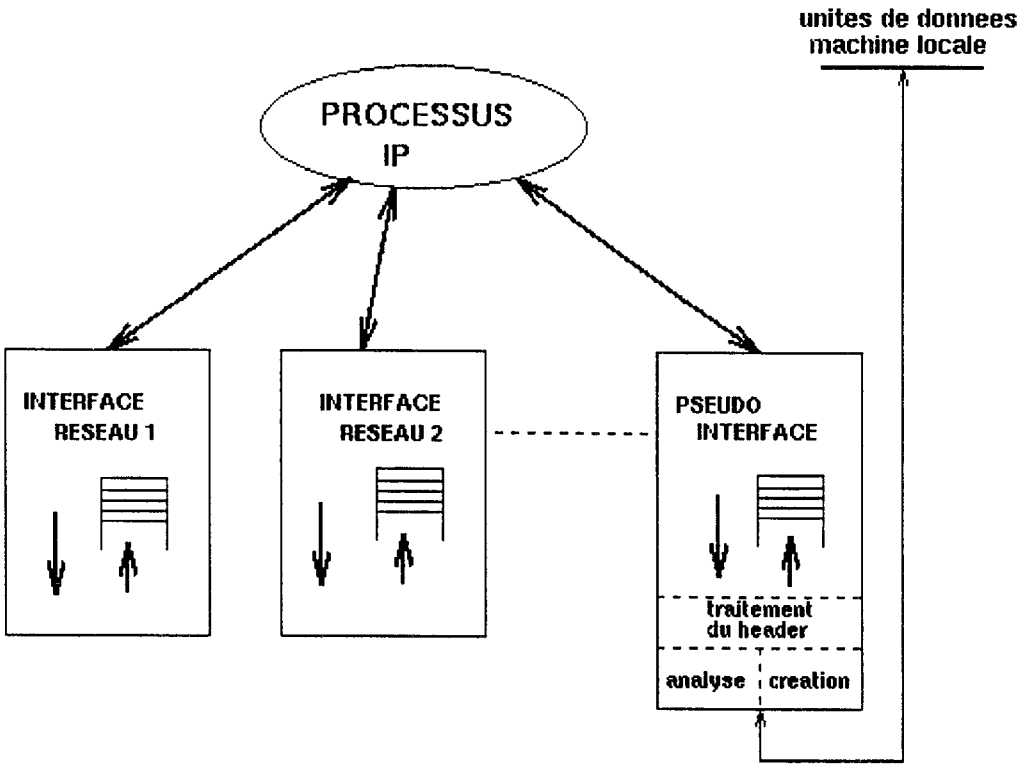


Fig. 2.7 interfaces et pseudo–interface

La priorité entre les entrées et les sorties est établie selon la méthode round–robin (on sert une première fois chacune des queues en attente avant de commencer à servir une deuxième fois).

Si toutes les queues sont vides, IP est en sommeil dans une procédure d'attente, il sera réveillé dès qu'un datagramme arrive dans une queue. Le processus IP:

- vérifie la validité du datagramme (version, classes d'adresses, checksum).
- calcule le routage (adresse du prochain gateway pour atteindre l'adresse destinataire).
- remplit le champ adresse source pour les datagrammes issues de l'interface locale.
- fragmente (si c'est nécessaire) les datagrammes à envoyer sur le réseau; réassemble les datagrammes reçus.
- recalcule le checksum de l'entête

Routage

Le routage est organisé autour d'une table et d'un algorithme, il regroupe deux types de traitement:

– le calcul de la route pour l'émission d'un datagramme. L'algorithme doit être rapide car utilisé très souvent. Les performances d'un gateway sont directement liées à l'optimisation de cet algorithme.

– la maintenance de la table (ajouts, suppressions, modifications de routes). Les opérations de mise à jour sont ponctuelles, donc moins exigeantes au niveau optimisation.

L'utilisation d'une "hash table" et des fonctions associées optimise le temps de parcours de la table de routage. Les accès à la table sont effectués en exclusion mutuelle.

Chaque entrée de la table contient un pointeur sur une liste chainée de structures contenant les informations de routage (pour une destination : adresse IP du destinataire, masque approprié, adresse du prochain gateway pour cette destination, interface réseau à utiliser pour accéder au gateway).

La sélection d'une route est obtenue en recherchant une coïncidence entre l'adresse destinataire du datagramme et une adresse dans la table par l'utilisation de masques.

Fragmentation et ré–assemblage

L'opération de fragmentation intervient après le routage. Elle est nécessaire quand la longueur du datagramme à émettre dépasse la taille maximum d'une unité de données de l'interface réseau.

IP découpe le datagramme original en multiples fragments, duplique l'en–tête pour chacun d'eux, positionne les indicateurs de fragmentation (fragment bit, more fragment, offset) et copie une partie des données.

Le ré–assemblage (opération inverse) reconstitue le datagramme initial à partir des multiples fragments reçus. Il opère en deux phases : réception des fragments jusqu'à ce que le datagramme soit complet puis reconstitution.

IP doit pouvoir traiter les fragments qui arrivent dans le désordre, en retard, et même mélangés avec des fragments d'autres datagrammes. Son implémentation est basée sur une table de structures de données permettant de situer rapidement un groupe de fragments, de localiser une insertion, de tester si le datagramme est complet.

Il utilise l'en–tête du fragment reçu pour identifier son groupe, puis il cherche l'entrée du groupe dans sa table, il teste si tous les fragments peuvent être ré–assemblés, si ce n'est pas le cas, il additionne le fragment à la liste chainée des fragments.

Quand le datagramme est complet, IP reconstitue le datagramme par une recopie des données de chaque fragments (position définie par le champ offset) et la construction de l'en–tête. Le datagramme peut alors être mis dans la file d'attente de la pseudo–interface.

De plus, IP assure la maintenance de sa table : il teste périodiquement ses listes de fragments et écarte les datagrammes incomplets ayant dépassé un certain délai (time to live).

2.2.1.3 Liaison des données : ETHERNET

Le niveau MAC (Medium Access Control) a en charge la gestion du support physique (Ethernet) :

- activation et désactivation de connexion physique.
- transmission d'unités de données du service physique.
- gestion des erreurs.

Il fournit l'adresse physique aux niveaux supérieurs et filtre les interruptions (en réception et en émission).

8	6	6	2	46 .. 1500	4 octets
PREAMBULE	DESTINAT	SOURCE	TYPE	DATA	FCS

TYPE = 0x0800 : vers TCP/IP
0x817D : vers XTP

Fig. 2.8 format de la trame Ethernet

Le champ préambule correspond à une phase de synchronisation. Les champs destination et source contiennent les adresses physiques (adresses Ethernet). Le champ type définit le protocole destinataire.

En émission, les requêtes sont non–blocantes (retour dès que le paquet est dans la file d'attente du boitier). Il n'y a pas de copie intermédiaire des données. Le driver analyse la structure pour connaitre le nombre de fragments à transférer et leur adresse. Si il y a moins de 64 octets à transmettre, le traitement de la requête prend en charge le remplissage.

En réception, le driver attend jusqu'à la réception du dernier paquet avant de prévenir le protocole destinataire.

2.2.2 Les STREAMS

Avec UNIX System V version 3, en 1986, AT&T a introduit un type de technologie d'entrées–sorties standard, les STREAMS.

Il s'agit d'un mécanisme noyau qui permet le développement de services répartis et de drivers pour la communication de données. Il définit des interfaces standard pour les entrées–sorties de caractères au niveau noyau et entre le noyau et le niveau utilisateur par la communication de messages.

Un stream est un chemin de communication bidirectionnel entre un driver dans le noyau et un processus utilisateur.

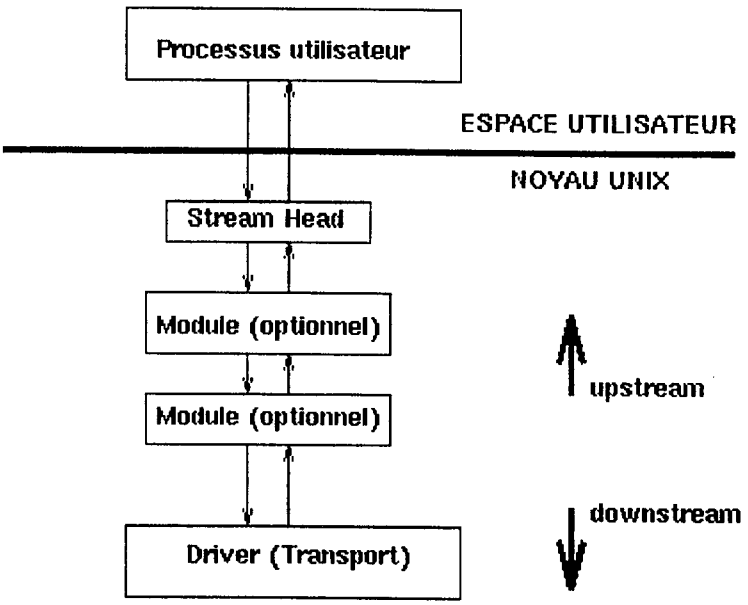


Fig. 2.9 schéma d'architecture d'un stream

Un stream est composé de trois parties : une tête de stream (Stream Head), un ou plusieurs modules (optionnels) et un driver (Stream End). La tête de stream fournit l'interface entre le stream et le processus utilisateur. Elle traite les appels système de l'utilisateur relatifs au stream et exécute le transfert bidirectionnel des données et des informations, entre l'application (dans l'espace utilisateur) et les modules (dans l'espace noyau).

Un module exécute les transformations intermédiaires sur les messages circulant entre la tête de stream et le driver. Il peut y avoir zéro ou plusieurs modules dans le stream suivant la nature du driver. Chaque module est constitué d'une paire de queues, une pour le côté lecture et une pour le côté écriture du module. Chaque queue contient une liste de messages en attente d'être traités.

Le driver transforme les données et les informations de contrôle entre les messages stream et leur représentation externe. Les données sont passées entre le driver et la tête de stream sous forme de messages. Les messages qui circulent de la tête de stream vers le driver sont dits transitant downstream et ceux qui circulent en sens inverse sont dits transitant upstream.

Ainsi l'infrastructure STREAMS reflète la nature ouverte des protocoles de communication les plus courants en assurant la modularité et la possibilité d'accès multiple. Elle constitue ainsi un support bien adapté pour l'implémentation de ces protocoles.

2.2.3 Les interfaces

Associées au mécanisme des STREAMS, les interfaces permettent la modularité du système de communication. Elles se situent à plusieurs niveaux :

- dans le noyau pour TPI et DLPI, respectivement l'une comme interface haute de l'entité de transport et l'autre à la frontière de la couche liaison de données.
- dans l'espace utilisateur pour la librairie XTI.

2.2.3.1 TPI et DLPI

TPI (Transport Provider Interface) et DLPI (Data Link Provider Interface) sont deux interfaces standards développées par AT&T Bell Laboratories. Ces interfaces sont orientées message, dans l'environnement STREAMS d'Unix.

Elles sont constituées d'un ensemble de primitives véhiculées par des messages Streams (La terminologie officielle retient le terme de "primitives" pour ces interfaces, mais en fait il s'agit de structures en langage C).

TPI définit une interface de transport (niveau noyau). Les primitives provenant de l'utilisateur (downstreams) correspondent à des requêtes (ici XTI) ou à des réponses aux événements envoyés par le transport. Les autres, provenant du transport, correspondent à des confirmations de requêtes ou à des indications d'événements reçus par celui–ci.

A l'autre extrémité, l'interface DLPI définit un ensemble de primitives et des règles d'utilisation permettant l'accès au service de liaison de données sans aucune connaissance du protocole utilisé. Les primitives de gestion locale gèrent dynamiquement le point de communication entre un fournisseur du service de liaison et un utilisateur (Data Link Service Access Point). encadrant les primitives de transfert de données.

2.2.3.2 La librairie XTI

XTI (X/Open Transport Interface) est une interface de transport conforme à l'environnement CAE (Common Applications Environment) défini par l'X/Open.

Afin de permettre la portabilité des applications, XTI définit une interface de service transport qui est indépendante des protocoles de transport utilisés. Ainsi XTI peut être utilisée avec les protocoles ISO TP4, TCP, UDP, Netbios.

XTI est implémentée sous la forme d'une librairie de fonctions C sous Unix. De façon génèrale, ces fonctions permettent :

- de se donner un point d'entrée au service transport,
- d'ouvrir et de fermer une connexion au transport,
- d'envoyer et de recevoir des données,
- d'obtenir des informations sur la connexion (paramètres, états,...).

Un descripteur de fichier spécial (fd) permet à l'utilisateur de s'identifier par rapport au transport (vue depuis l'utilisateur). Ainsi la première fonction à utiliser est t_open(), qui rend un descripteur de fichier spécial. Ce fd sera utilisé comme argument dans toutes les fonctions appelées ultérieurement : il identifie le point d'entrée logique au service transport.

XTI propose deux modes de service : connexion et datagramme. Le mode connexion permet de transférer des données après avoir établi une connexion durable et fiable. Ce mode permet de négocier les paramètres et les options qui gouverneront la phase de transfert des données. Un lien logique est établi entre les unités de données transmises. En revanche, le mode sans connexion est un service orienté message. Les unités de données (datagrammes) contiennent à la fois l'adresse destination et les données elles–mêmes. Il n'y a aucun lien logique entre deux datagrammes successifs; du reste, ils n'arriveront peut–être pas à destination dans l'ordre où ils ont été émis.

La figure 2.10 illustre l'appel des fonctions de la librairie XTI dans les deux modes de connexion.

Plusieurs appels successifs à la fonction t_open() permettent d'obtenir plusieurs fd. De cette façon, un utilisateur peut gérer plusieurs connexions de transport à la fois. De même, plusieurs processus peuvent se partager le même fd, à condition qu'ils synchronisent leurs actions de manière à ne pas utiliser une fonction qui ne corresponde pas à l'état dans lequel se trouve leur connexion de transport.

Une caractéristique inhérente à la couche Transport étant qu'elle cache les détails physiques du médium utilisé, l'interface XTI quant à elle, permet une indépendance des applications vis–à–vis des protocoles et du réseau.

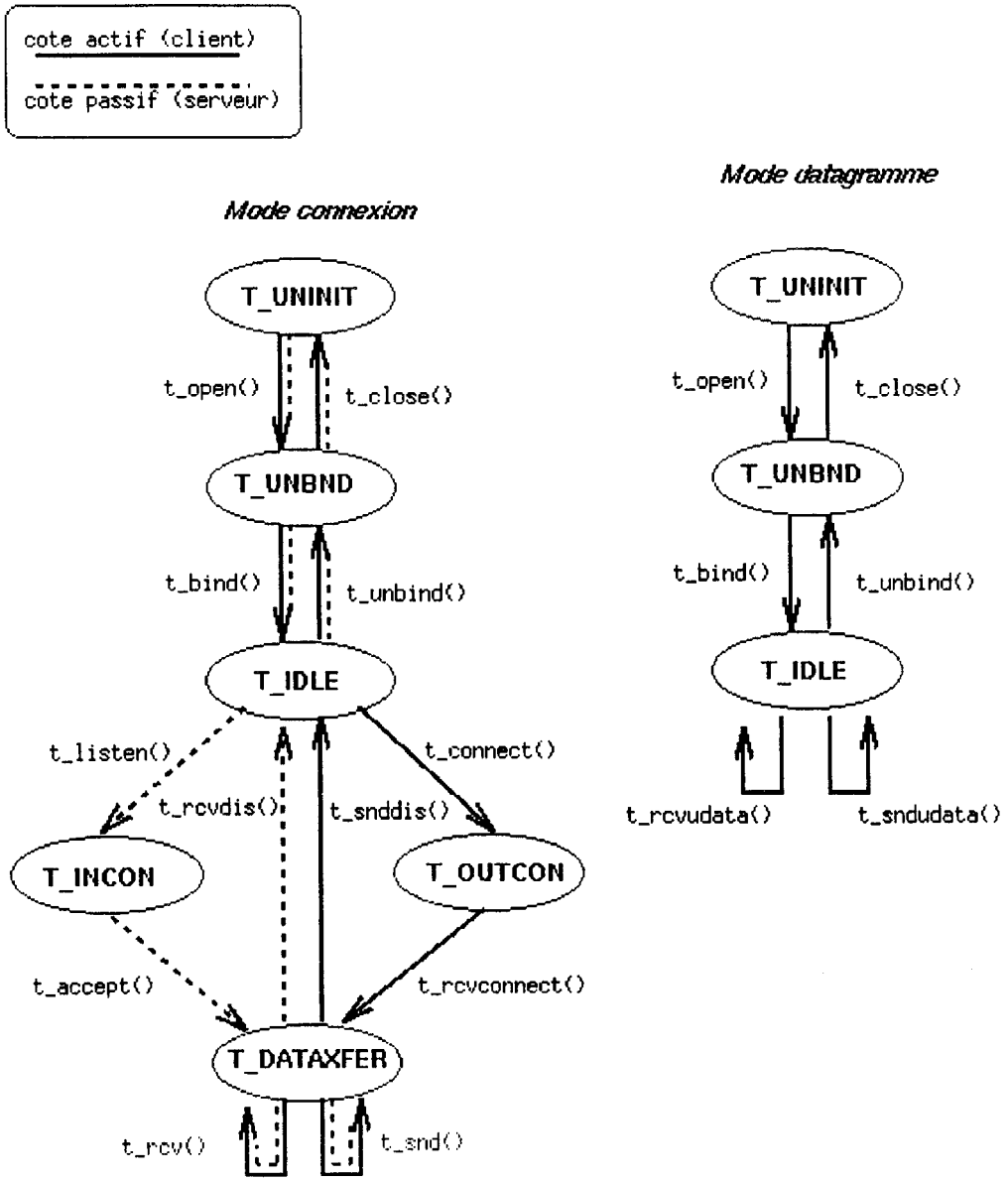


Fig. 2.10 Mode connexion et mode datagramme

2.2.4 Une application: le programme BENCH

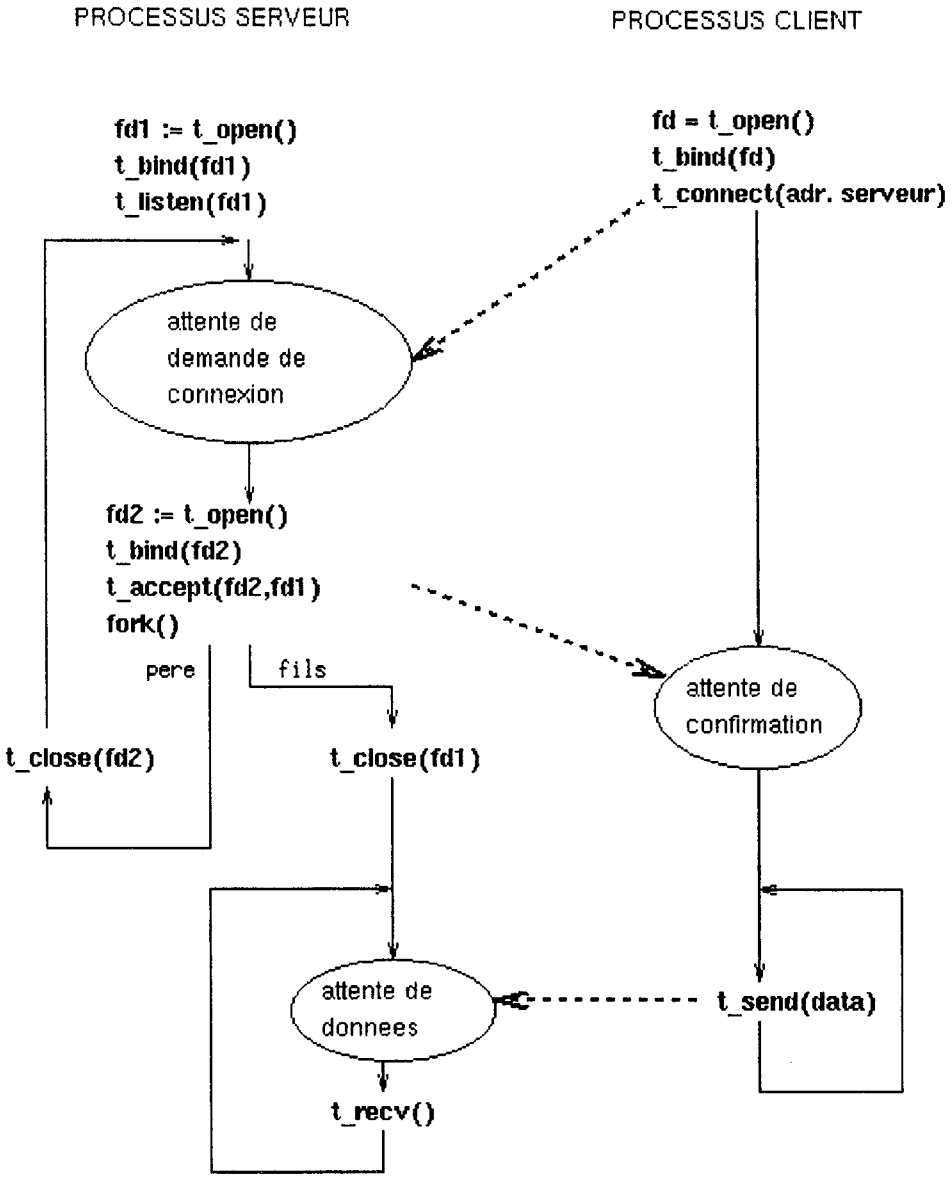


Fig. 2.11 le modèle client–serveur appliqué au programme BENCH

Les applications classiques de TCP sont basées sur le modèle client–serveur. Le principe de base est qu'un processus peut être en attente d'indication de connexion entrante sur un fd et accepter la connexion sur un fd différent. Ceci facilite l'écriture d'une application serveur où le serveur attend toutes les indications de connexion pour un TSAP donné (point d'accès au transport), accepte la connexion sur un nouveau fd et lance le processus fils [fork()] qui servira les requêtes relatives à cette connexion.

BENCH est une application développée par Bull pour le test du fonctionnement et la mesure des performances de TCP/IP. Elle accède aux services du transport par des appels à la librairie XTI.

Du côté passif, le serveur ouvre un stream (t_open), s'attache à un TSAP (t_bind), et enfin se met en attente de demandes de connexion (t_listen). Le nombre maximum de connexions acceptables peut être spécifié dans un paramètre de t_bind.

De son côté le processus client ouvre un stream et acquiert un TSAP (t_open et t_bind), puis il envoie une demande de connexion au serveur (t_connect).

Quand une demande de connexion arrive, le serveur est averti et doit renvoyer une confirmation de connexion par la fonction XTI : t_accept. Pour TCP comme pour XTP, cette confirmation n'a aucun effet sur l'état de la connexion qui est déjà établie. Elle permet au serveur de passer la connexion à un processus fils. Les opérations suivantes relatives à cette connexion (émission et réception), seront effectuées avec un autre fd [fdi].

Ainsi le serveur reste en attente d'une nouvelle demande de connexion. Le processus fils qu'il a créé traite les requêtes du processus client.

2.3 Le protocole XTP

XTP s'inscrit dans le cadre des efforts développés pour répondre à la fois aux nouveaux besoins des utilisateurs en matière de réseaux et aux évolutions des technologies de transmission.

C'est un protocole en cours d'achèvement, c'est à dire que même si le volume des modifications d'une version à l'autre tend à diminuer, il ne s'agit pas encore d'un protocole stabilisé. Ainsi, alors qu'il était initialement conçu pour remplacer les protocoles TCP/IP, il a subi dans la version 3.6 d'importants changements visant à le faire converger vers les normes ISO.

Il s'agit d'un protocole de transfert allégé, c'est à dire qu'il regroupe en une couche des fonctionnalités des couches réseau et transport du modèle OSI. Ce regroupement permet la suppression de redondances, notamment au niveau de la segmentation et du contrôle de flux. L'implantation de certaines fonctionnalités sur des circuits intégrés spécifiques (ASIC) permet des gains de performances. D'autre part ce protocole a été conçu pour supporter une partie des traitements en parallèle. Des implémentations du protocole ont notamment été réalisées sur transputers [ROCA].

Enfin, il faut insister sur le fait que si le protocole est défini, ce n'est pas le cas encore pour le service fourni, pour lequel seules quelques indications sont données.

XTP a un fonctionnement de type commutation de circuits, il permet d'établir un circuit virtuel bidirectionnel entre deux systèmes. Dans ce mode de fonctionnement, le paquet d'initialisation de connexion trace un chemin à travers les différents nœuds du réseau, les autres paquets relatifs à cette connexion emprunteront tous ce même chemin. XTP peut fonctionner en mode connexion ou en mode datagramme fiable (connexion à très courte durée de vie).

L'état de l'entité XTP de chaque système est appelé contexte. Le modèle de transmission est un flux d'octets auquel on a ajouté la possibilité d'insérer de façon transparente des "frontières" du côté émetteur, frontières qui sont fidèlement reproduites à l'arrivée, du côté receveur.

Il s'agit de présenter ici brièvement les principaux mécanismes dont dispose le protocole XTP.

2.3.1 Le paquet XTP

Le paquet est l'unité de transfert défini par le contrôle d'accès au medium. XTP utilise deux formats de paquet : un pour le contrôle et un pour les données (paquet d'information).

Les deux formats ont en commun un segment de tête (header) et un segment de fin (trailer) chacun de longueur constante. Les segments d'information et de contrôle sont de longueur variable.

Le segment de tête spécifie le type du paquet et identifie la portion du flux de données présente dans le segment d'information. Il contient les options choisies pour la transmission. Le trailer contient des indications pour le contrôle d'erreurs, il identifie la proportion du flux de données délivrées à l'application réceptrice, et enfin il contient les indicateurs qui contrôle les changements d'état (déconnexion, demande d'acquittement,...).

Le segment d'information contient évidemment les données utilisateur à transférer mais il est aussi utilisé pour communiquer des informations particulières quand c'est nécessaire (adresses en début de connexion,...).

Le segment de contrôle délivre les valeurs des paramètres relatifs aux différents contrôles (flux, erreurs,..).

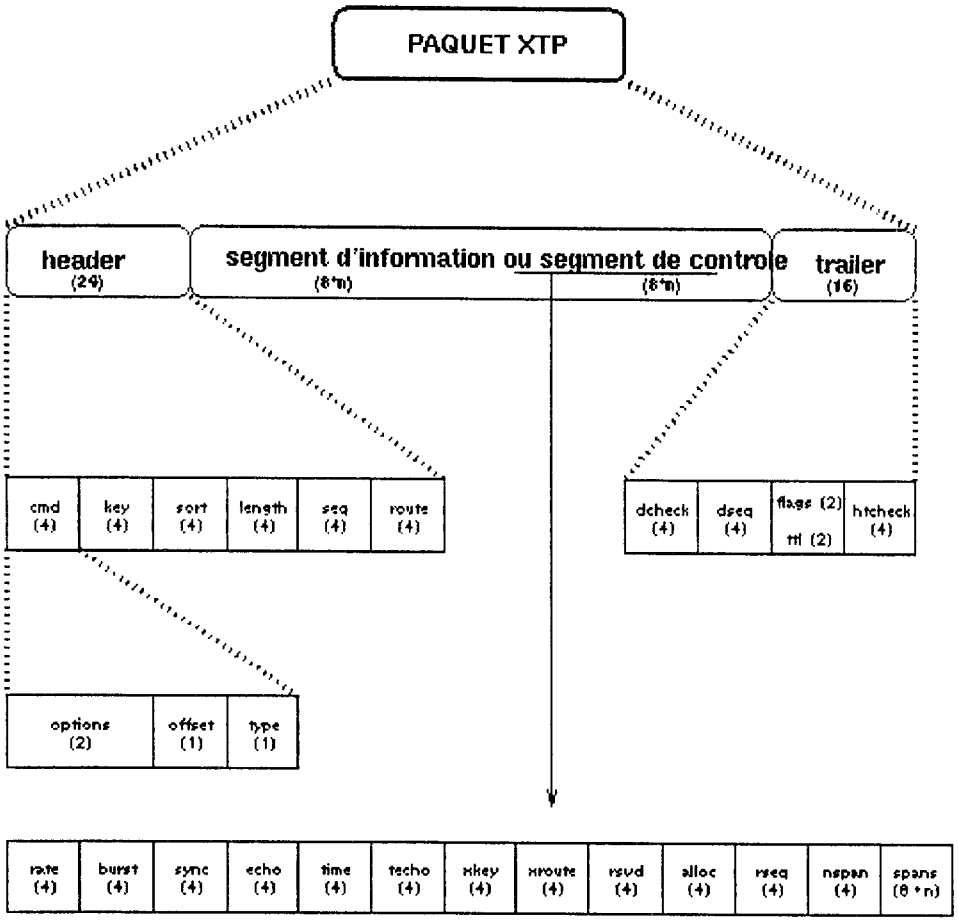


Fig. 2.12 format des paquets XTP

2.3.2 connexion

XTP propose les modes de connexion implicite et explicite.

Le mécanisme de connexion explicite requiert une confirmation de connexion avant l'envoi des données. Il permet de négocier la qualité du service pour la connexion. Avec XTP, deux messages suffisent (demande et réponse).

A l'opposé, le mode implicite correspond à l'envoi d'un paquet d'initialisation de connexion (pouvant éventuellement contenir des données utilisateur). L'entité de transport considère la connexion comme établie et peut émettre des paquets de données sans attendre de recevoir la confirmation de connexion.

XTP permet aussi les modes confirmé et non confirmé.

Le mode non confirmé correspond à un acceptation "automatique" des demandes de connexion. L'utilisateur indique a priori à son entité XTP les connexions qu'il est prêt à accepter, en précisant les critères de sélection. L'entité XTP passe en état d'écoute (listen–state). Elle traitera les requêtes de connexion qui arriveront sans en référer à l'utilisateur, auquel elle enverra seulement une indication d'acceptation de connexion.

A l'opposé, l'utilisateur peut choisir le mode confirmé. Dans ce cas, les critères passés à l'entité XTP servent seulement à un premier tri, la décision finale revenant à l'utilisateur.

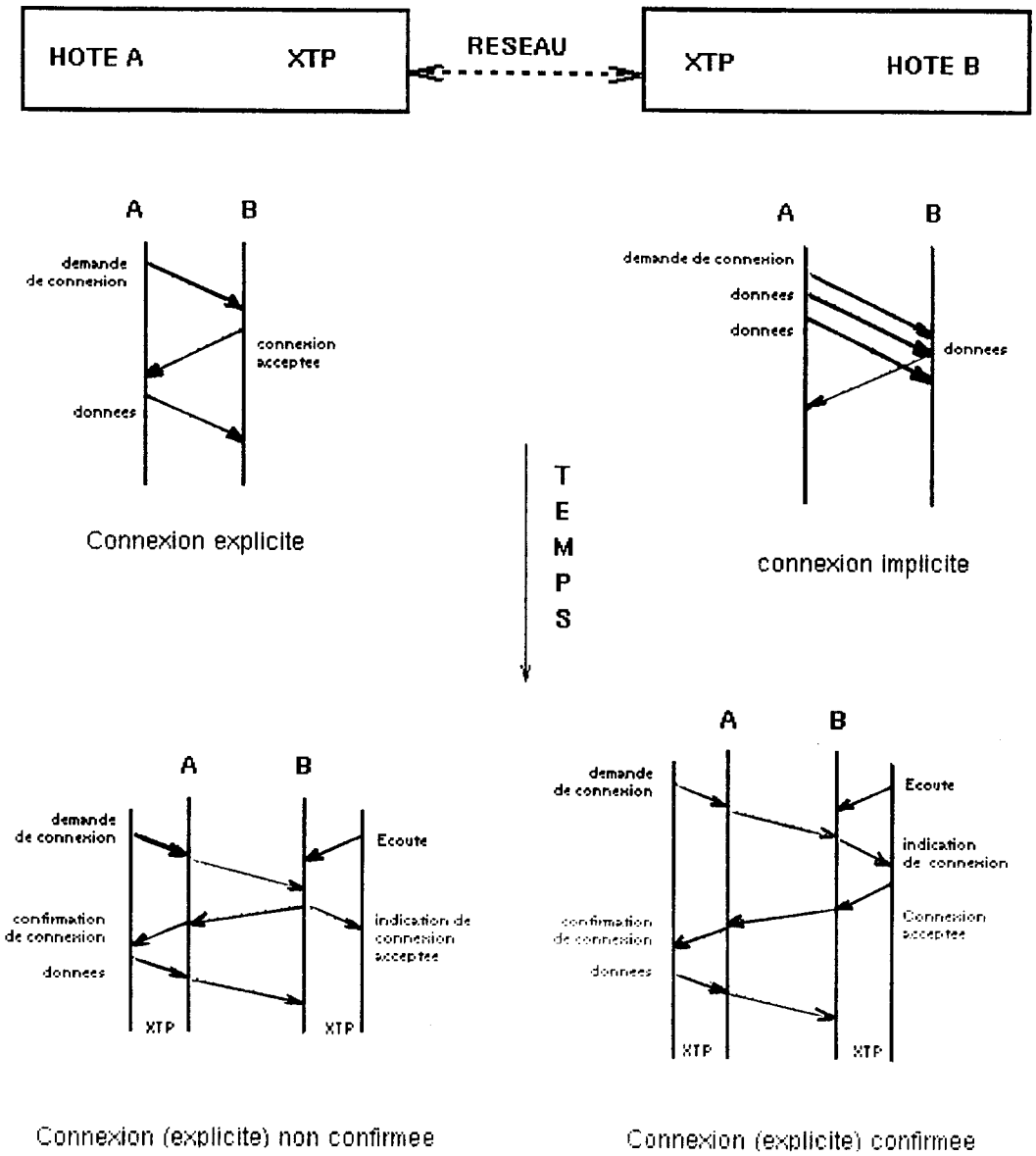
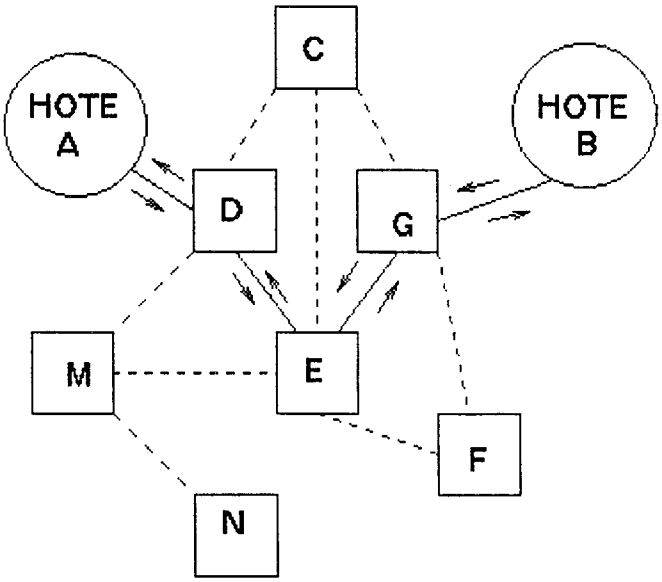


Fig. 2.13 modes de connexion

2.3.3 routage



A-D-E-G-B represente un chemin

Fig. 2.14 routage d'XTP

XTP étant un protocole de transfert, il intègre une partie des mécanismes de la couche réseau et en particulier, le routage. Il a un fonctionnement de type commutation de circuit : le paquet d'initialisation de connexion trace un "chemin" à travers le réseau, les autres paquets émis dans le cadre de cette connexion empruntent ce même chemin.

Le protocole XTP exploite deux champs de l'en–tête du paquet ("route" et "key") pour réaliser cette opération. Les adresses destination et source ne sont indiquées que dans le paquet d'initialisation de la connexion.

Grâce à des mécanismes comme l'échange de route, le partage de route et l'échange de clé, XTP assure un routage de proche en proche optimisé qui simplifie les opérations sur les tables de routage.

2.3.4 déconnexion

XTP dispose des mécanismes de déconnexion ordonnée ou forcée.

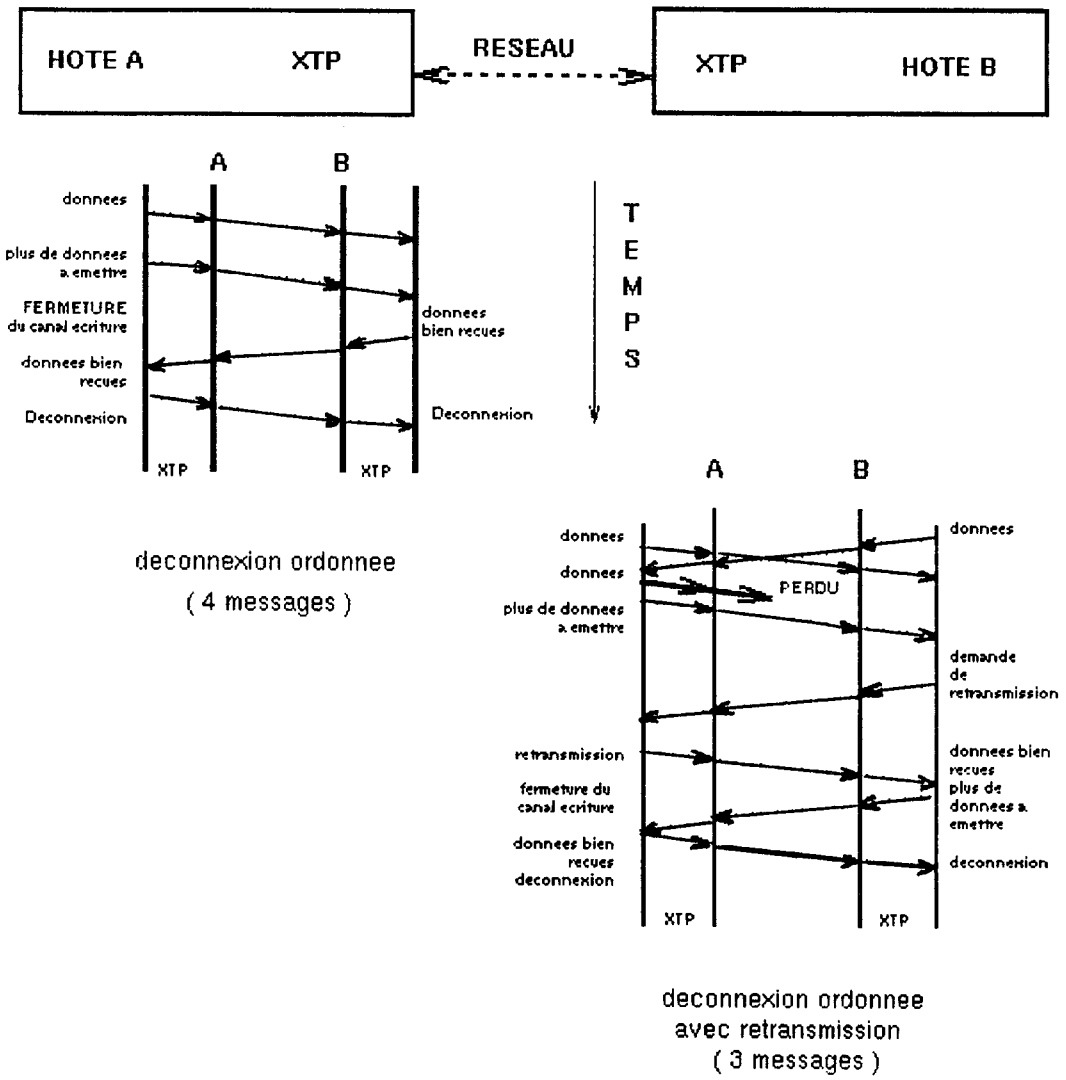


Fig. 2.15 déconnexions ordonnées

Dans une déconnexion ordonnée, il y a accord des deux entités connectées, ce qui permet de demander des retransmissions et garantit une transmission fiable et correcte des données. XTP offre la possibilité, comme TCP, de séparer la fermeture des canaux d'écriture et de lecture. La déconnexion ordonnée implique l'échange de trois ou quatre messages.

Par contre, la déconnexion forcée ne garantit pas le transfert correct des données. L'effet du message de déconnexion est immédiat, et il n'y a pas de négociation. Ce mécanisme, basé sur l'envoi d'un message, est comparable à celui qu'on trouve dans ISO TP4 : deux messages indication et acquittement.

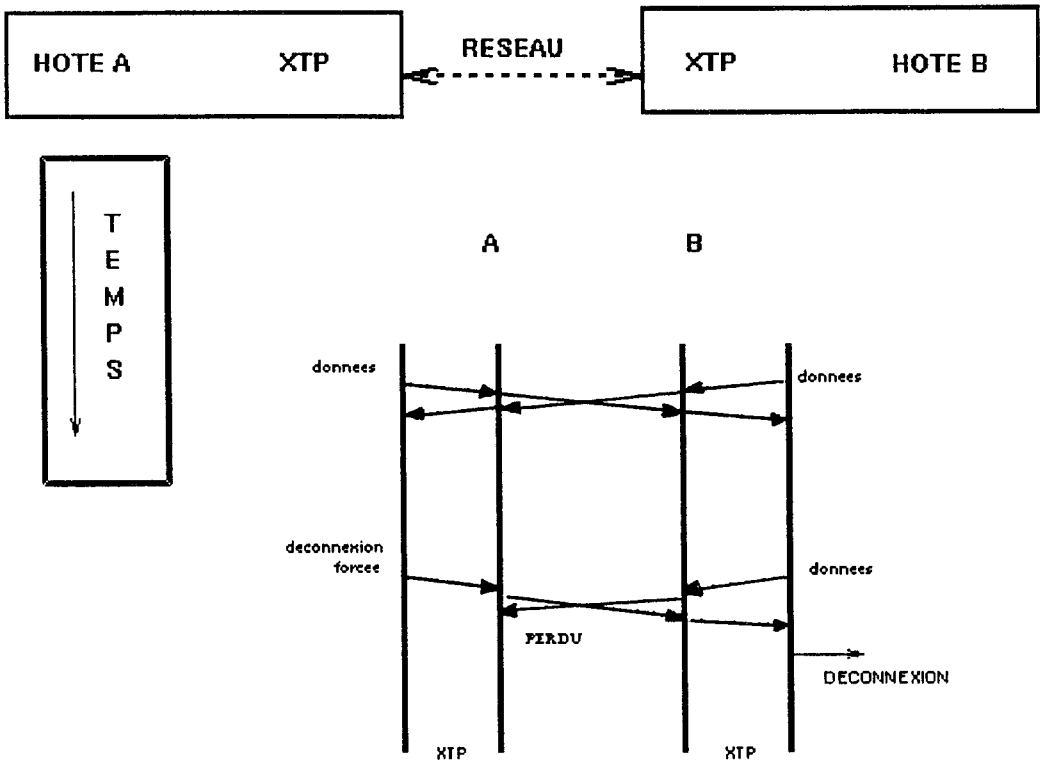


Fig. 2.16 déconnexion forcée

Comme pour la connexion, XTP propose les modes de déconnexion ordonnée confirmée ou non confirmée. L'intérêt du mécanisme de déconnexion confirmée est loin d'être évident, en raison du fonctionnement même du protocole XTP. Il apparait que ce mécanisme revient à demander à l'entité réceptrice d'entériner l'arrêt de réception sans qu'une réelle alternative soit possible (demande de retransmission par exemple).

2.3.5 segmentation

XTP dispose du mécanisme de segmentation / ré–assemblage au niveau transport. Ce mécanisme permet d'accepter des messages utilisateurs de longueur quelconque (par exemple un long fichier texte), et de découper ce message en messages plus petits pour respecter les limitations de taille imposées par le réseau. Ainsi XTP évite la redondance du mécanisme de fragmentation du niveau réseau.

De plus XTP ne dispose pas du mécanisme de groupage des TSDU (unite de donnée du service transport) au niveau du transport. En effet, ce mécanisme sert à minimiser les pertes de temps dans le réseau. Or maintenant, avec l'évolution considérable des technologies de transmission, tant au niveau des performances qu'au niveau de la fiabilité, les pertes de temps se situent dans les calculateurs. Le groupage n'est donc plus utile.

2.3.6 contrôle d'erreurs

De même que TCP et TP4, XTP doit détecter les erreurs et retransmettre les données en erreur ou perdues. Avec un fonctionnement de type commutation de circuits, XTP garantit l'arrivée des paquets dans l'ordre.

Il utilise deux checksums (calcul d'un polynôme de contrôle) pour vérifier l'intégrité des données, l'un pour le segment d'information (débrayable) et l'autre pour les autres segments. XTP place les checksums dans le trailer afin d'obtenir la simultanéité de leur calcul avec l'émission ou la réception. Un checksum en erreur entraine le rejet du paquet.

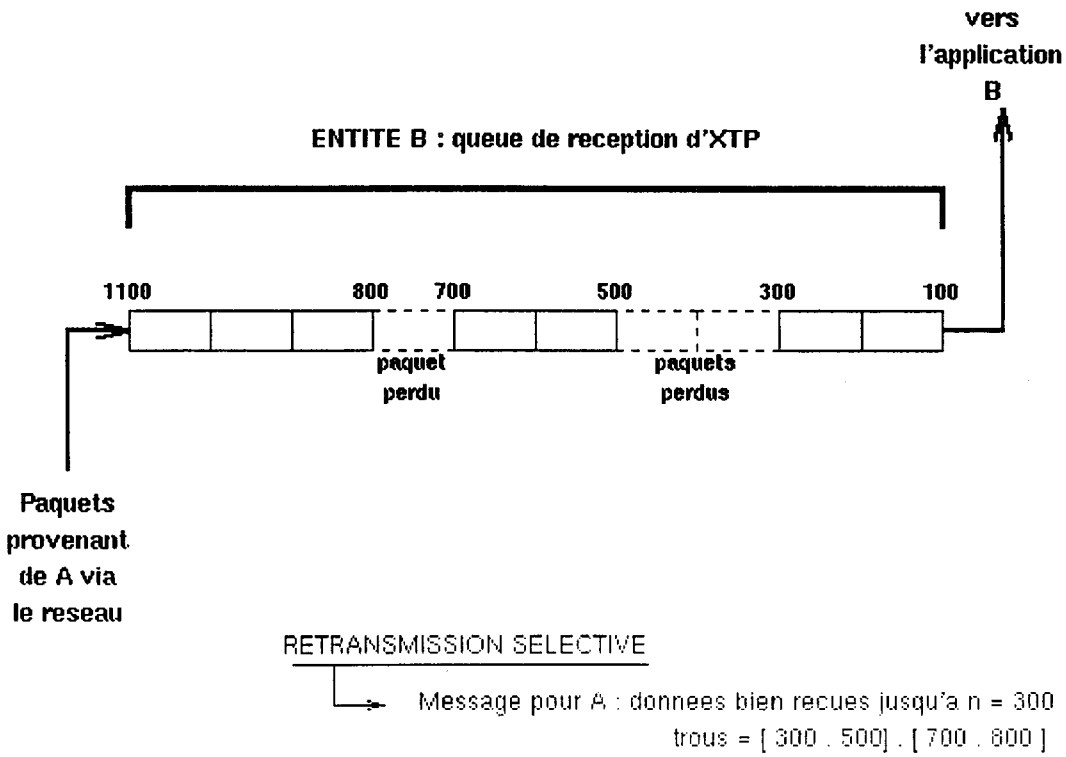


Fig. 2.17 contrôle d'erreurs d'XTP

La détection des données perdues est basée sur le comptage des octets émis (numéro de séquence). XTP dispose de deux mécanismes d'acquittement : cumulatif (positif) et sélectif (négatif).

Par l'acquittement cumulatif, le récepteur indique le numéro d'octets pour lequel tous les paquets concernant des données ayant un numéro de séquence inférieur ont été bien reçus. Alors que l'acquittement sélectif indique les numéros de séquence des paquets perdus ce qui permet une retransmission sélective des données.

Enfin la politique d'acquittement est sous la responsabilité de l'entité émettrice: dans le cas de fonctionnement normal, l'entité XTP réceptrice ne renvoie un acquittement que si et seulement si ça lui est explicitement demandé. Ce qui permet de minimiser les traitements à l'intérieur du protocole ainsi que l'encombrement du réseau.

2.3.7 contrôle de flux

Le contrôle de flux est basé sur la numérotation des données. Il utilise le mécanisme classique de "fenêtre glissante". L'entité réceptrice indique à l'entité émettrice deux nombres :

- le plus grand numéro de données pour lequel toutes les données ayant un numéro inférieur ont été correctement reçues. Il constitue le bord inférieur de la fenêtre.
- le numéro maximum de données à ne pas dépasser en émission de données. Il correspond au bord supérieur de la fenêtre.

Les données émises ou retransmises devront avoir des numéros de séquence compris entre ces deux nombres. La différence entre ces deux nombres donne la taille allouée au buffer en réception. La mise à jour de la valeur de ces deux nombres provoque le glissement de la fenêtre.

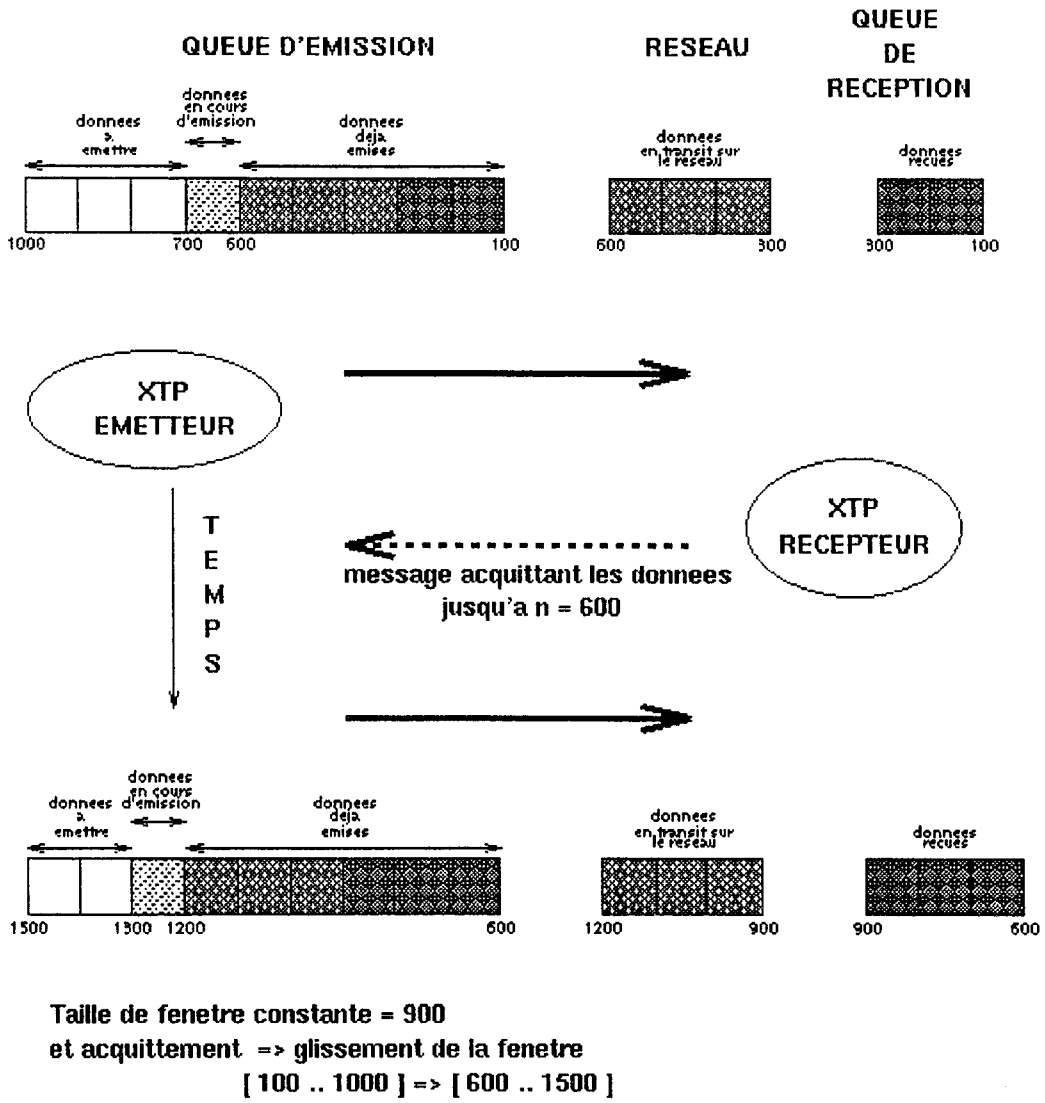


Fig. 2.18 contrôle de flux d'XTP

2.4 Les apports du protocole XTP

L'évolution des applications comme celle des supports de communication devrait influencer les modèles de protocoles de transport contemporains. Avec le développement des systèmes distribués, l'utilisation des communications sur les réseaux glisse vers l'exploitation de nouveaux mécanismes : échanges à deux messages (RPC), opérations en mode diffusion (multicast) où un message peut être envoyé à plusieurs postes distants, etc...L'originalité du protocole XTP est que ses mécanismes apparaissent comme étant adaptés aussi bien aux nouveaux supports de communication qu'aux nouvelles applications.

Pour une meilleure adaptation aux applications, les concepteurs ont concentré leurs efforts sur des mécanismes de connexion allégée (implicite) et de multicast. Alors que TCP nécessite l'échange de trois paquets pour ouvrir une connexion, XTP a recours à la connexion implicite où l'ouverture de la connexion apparaît comme un effet de bord du premier paquet reçu (FIRST) contenant éventuellement des données utilisateur. Ce mécanisme permet une réalisation efficace des échanges à deux messages. De plus, la fermeture de connexion ne nécessite aucun paquet de contrôle pour fermer la connexion. Enfin, la possibilité d'invalider les contrôles d'erreur parait intéressante pour certaines applications (transfert de voix numérisée, par exemple).

SERVICES	**types d'applications**
services avec connexion	
transfert fiable de messages	transport d'une suite de pages
transfert fiable de données	transfert de fichier
transfert sans contrôle d'erreur	voix numérisée
services sans connexion	
transfert de datagramme sans acquittement	messagerie électronique
transfert de datagramme avec acquittement	messagerie électronique avec accusé de réception
requêtes / réponses	consultation de base de données

Le but de rendre XTP mieux adapté que TP4 et TCP aux possibilités des nouveaux supports de communication (hauts débits et faible taux d'erreur) a influencé les mécanismes mis en œuvre dans les parties routage et transport de ce protocole. La connexion implicite associée à la possibilité d'un transfert de données immédiat (sans attendre de confirmation de connexion) peut augmenter de façon significative les performances d'un transfert de données.

Par ces caractéristiques, XTP apparait attractif et prometteur, bien adapté aux nouveaux besoins. Au cours de l'étude comparative des protocoles présentée dans un chapître suivant, nous présenterons plus en détail les différences et les innovations apportées par XTP.

2.5 Conclusion

XTP par sa définition semble présenter des avantages indéniables en terme de services offerts. Le but de cette étude est d'évaluer sur le plan des performances le gain obtenu avec XTP par rapport à TCP.

Les contraintes sont de trois types .

– 1 – Contraintes liées au protocole :

KRM (Kernel Reference Model) est l'implémentation logicielle du protocole XTP que nous avons choisi d'intégrer. Elle dispose de mécanismes spécifiques associés à une gestion mémoire qui lui est propre, l'ensemble étant incompatible avec la plateforme initiale. L'écriture d'interfaces haute et basse est nécessaire pour le portage ainsi que la suppression de certains mécanismes (les uns en raison du portage, les autres en raison du mode de fonctionnement choisi).

– 2 – Contraintes liées à la réalisation :

Le projet est divisé en 4 parties (portage, intégration, définition du service d'accès à XTP, améliorations).

– 3 – Contraintes liées à l'étude des performances

La mise en œuvre du protocole XTP ne doit en rien altérer le fonctionnement de TCP, les deux protocoles doivent pouvoir être utilisés indifféremment dans des conditions identiques.

L'ensemble des contraintes exposées alliées à celles propres à la structure d'accueil matérielle et logicielle vont être à la base de la définition de l'organisation du projet.

INSTITUT IMAG
Informatique, Mathématiques Appliquées de Grenoble
CNRS-INPG-USMG
MÉDIATHÈQUE
B.P. 53 X
38041 GRENOBLE CEDEX
FRANCE
Tél. 76.51.46.36

Chapitre 3

REALISATIONS

3.1 Introduction

La réalisation de ce projet s'est articulée autour de trois grands axes : une organisation logicielle associée à une méthodologie, la définition d'une implémentation pour XTP et enfin une stratégie pour réaliser notre plateforme de communication.

La structure d'accueil s'est constituée à partir du système utilisé pour la production de programme des communications sous Unix. Le protocole XTP devait prendre place dans cette organisation sans en altérer l'homogénéité et en respectant sa cohérence.

D'autre part, l'implémentation choisie pour l'intégration du protocole XTP est un ensemble complexe et très peu documenté. Là encore, il est apparu essentiel de préserver l'homogénéité du code du protocole, et de faire porter les efforts sur l'écriture des interfaces, ce qui a conduit à définir la première implémentation de XTP sous STREAMS.

Enfin il fallait opter pour une stratégie qui prenne en compte les deux premiers points, afin d'atteindre l'objectif tout en respectant les contraintes d'un calendrier.

Ce chapitre présente les trois axes de la réalisation de cette plateforme et conclut sur une première évaluation.

3.2 Organisation logicielle et méthodologie

L'intégration du protocole XTP a nécessité une étape préliminaire d'exploration de l'existant, notamment pour ce qui est organisation logicielle et exploitation des communications dans le système unix.

Au cours de cette étape, il est apparu important de distinguer très nettement l'aspect production de programme, où on était utilisateur du système unix pour la création d'un produit complexe, de l'aspect mise au point, où nous étions un des composants à l'intérieur du noyau.

3.2.1 Production de programme

Sous la racine de notre système de production de programme, se trouvent les racines des arborescences des n produits fabriqués (symbolisées par /PRODUIT). Ici le produit généré est Unix, l'emploi du terme générique "produit" permet de différencier l'Unix en cours d'exploitation sur la machine de production de programme de l'Unix en cours de développement.

Le principe de la production de programme pour un produit donné est régi par la nécessité de permettre à plusieurs opérateurs d'intervenir simultanément sur un même produit. Ceci implique une architecture logicielle associée à une méthode dans la chronologie des différentes opérations réalisées par ces opérateurs.

3.2.1.1 Arborescences parallèles

Pour un produit donné, trois arborescences parallèles sont prévues : arborescences de référence, de travail et d'intégration. Cependant l'espace mémoire occupé par une arborescence ainsi que sa complexité, interdisent les duplications brutales pour chaque arborescence de travail.

Il est donc préférable d'exploiter le mécanisme des liaisons de fichier offert par unix et de lui associer une chronologie dans les opérations à effectuer :

1 – ***mktree*** : pour générer l'arborescence de travail à partir de l'arborescence référence (crée par des liens).

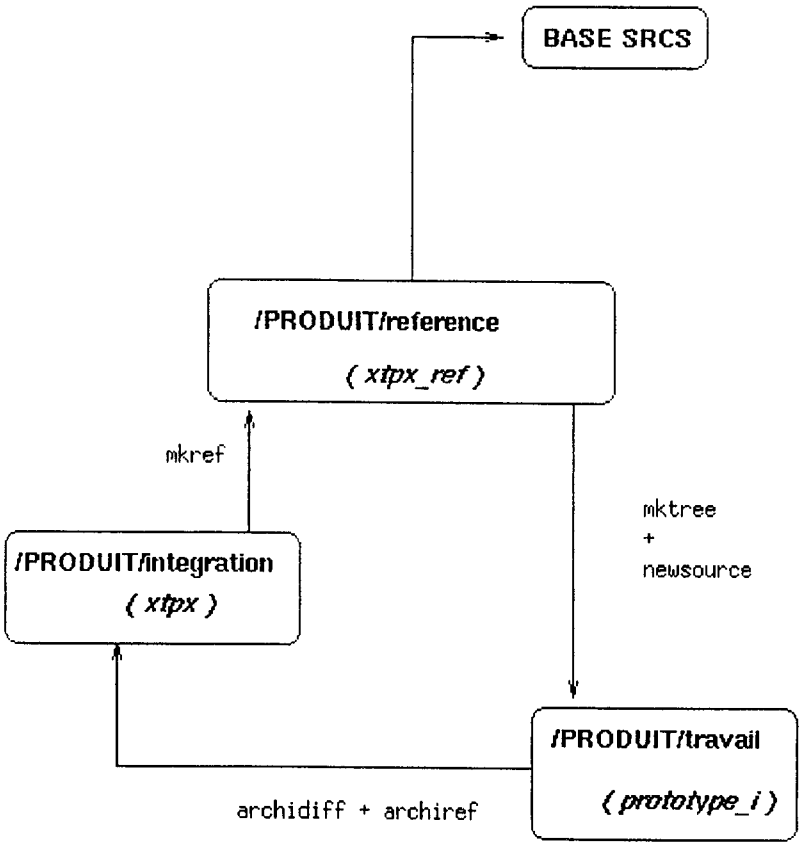
2 – ***newsource*** : pour personnaliser les fichiers sur lesquels on souhaite intervenir (casser les liens avec la référence).

3 – effectuer les modifications. Compiler et valider (debug).

4 – ***archidiff*** : opération préliminaire avant d'intégrer le code modifié, nécessaire pour s'assurer qu'on introduit les modifications d'un seul opérateur à la fois.

5 – ***archiref*** : si la commande précédente l'autorise, archiref permet d'intégrer les modifications dans l'arborescence d'intégration.

6 – ***mkref*** : pour transférer le code de l'arborescence d'intégration dans l'arborescence de référence.



L'archivage dans la base SRCS concerne le produit fini, validé et livré. Cette opération est à la charge de l'administrateur du système.

3.2.1.2 Organisation de l'arborescence

Tous les fichiers sources du système de production de programme, pour les communications, sont répartis sur 3 répertoires principaux:

–kernel/ios: contient le code du noyau pour la "partie entrées/sorties" (réparti en plusieurs sous–chapitres).

–cmd : regroupe les commandes utilisateur.

–lib : contient les sources de la librairie XTI.

A la compilation, les codes générés à partir des fichiers sources contenus dans ces répertoires sont rassemblés dans un autre répertoire **target** qui représente la machine destinatrice.

kernel/ios/inet : regroupe les différents interfaces et protocoles du transport à la couche physique (tcp, udp, arp, ip, dlpi, deth,...) constituant la suite de protocoles internet.

cmd : Cette partie réalise l'interface avec les commandes utilisateur c'est à dire quelques unes des commandes que l'on trouvera disponibles au niveau /usr/bin d'une machine installée (par exemple : ftp, rcp, rshell, rlogin,....).

Ces différents fichiers ont en charge : la gestion des paramètres de la commande, l'ouverture de streams et de fichiers, les appels au noyau, la gestion des erreurs, la libération des streams et des fichiers qui ont été ouverts par cette commande.

lib : regroupe les sources de la librairies XTI, ils sont compilés et archivés dans une librairie.

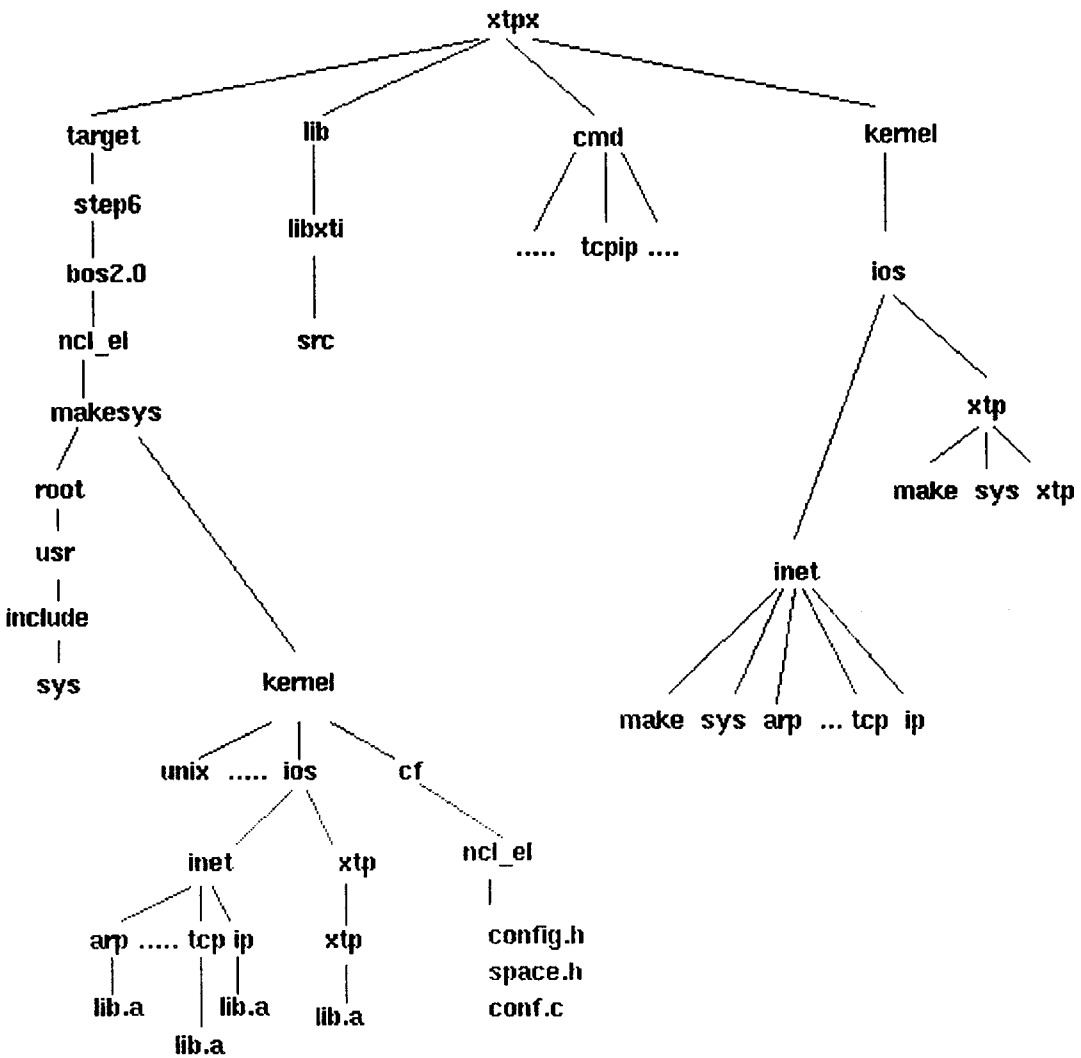


Fig. 3.1 organisation d'une arborescence

Ainsi, l'intégration d'un nouveau protocole impose de connaitre l'organisation logicielle d'une arborescence dans son ensemble :

– pour pouvoir conserver l'homogénéité du système dans le choix de la nouvelle organisation. Nous avons donc choisi de placer XTP au même niveau que la suite de protocoles INET et d'utiliser des mécanismes identiques.

– pour pouvoir faire les modifications nécessaires sur les différents fichiers existants, au niveau commandes, librairies ou fichiers de configurations afin d'apporter l'extension voulue au système précédent.

3.2.2 Exploitation des communications

L'exploitation des communications dans le système est définie au niveau de l'administration. Elle s'appuie sur une architecture comprenant des composants spécifiques et sur un séquencement dans les événements.

3.2.2.1 Composants spécifiques de la partition système

/etc	programmes et tables pour l'administration : fichiers de commandes en shell qui spécifient la configuration du système, les services offerts,...
/dev	fichiers spéciaux définissant tous les "devices" du système. Ils permettent de leur associer un identificateur et de les manipuler comme un fichier.
/unix	système d'exploitation comprenant le code du noyau ainsi que celui des différents drivers.
/usr/bin	commandes spécifiques : ftp, ... , bench et benchd

3.2.2.2 Séquencement des opérations du chargement d'Unix et des communications

Sous Unix, la plupart des programmes serveur des applications standards exploitant les communications sont lancés par un super serveur, "Internet daemon" (/etc/inetd) qui agit en serveur maitre pour les programmes spécifiés dans son fichier de configuration : /etc/inetd.conf. Il est à l'écoute des demandes de services pour ces serveurs et lance le programme approprié quand une requête arrive.

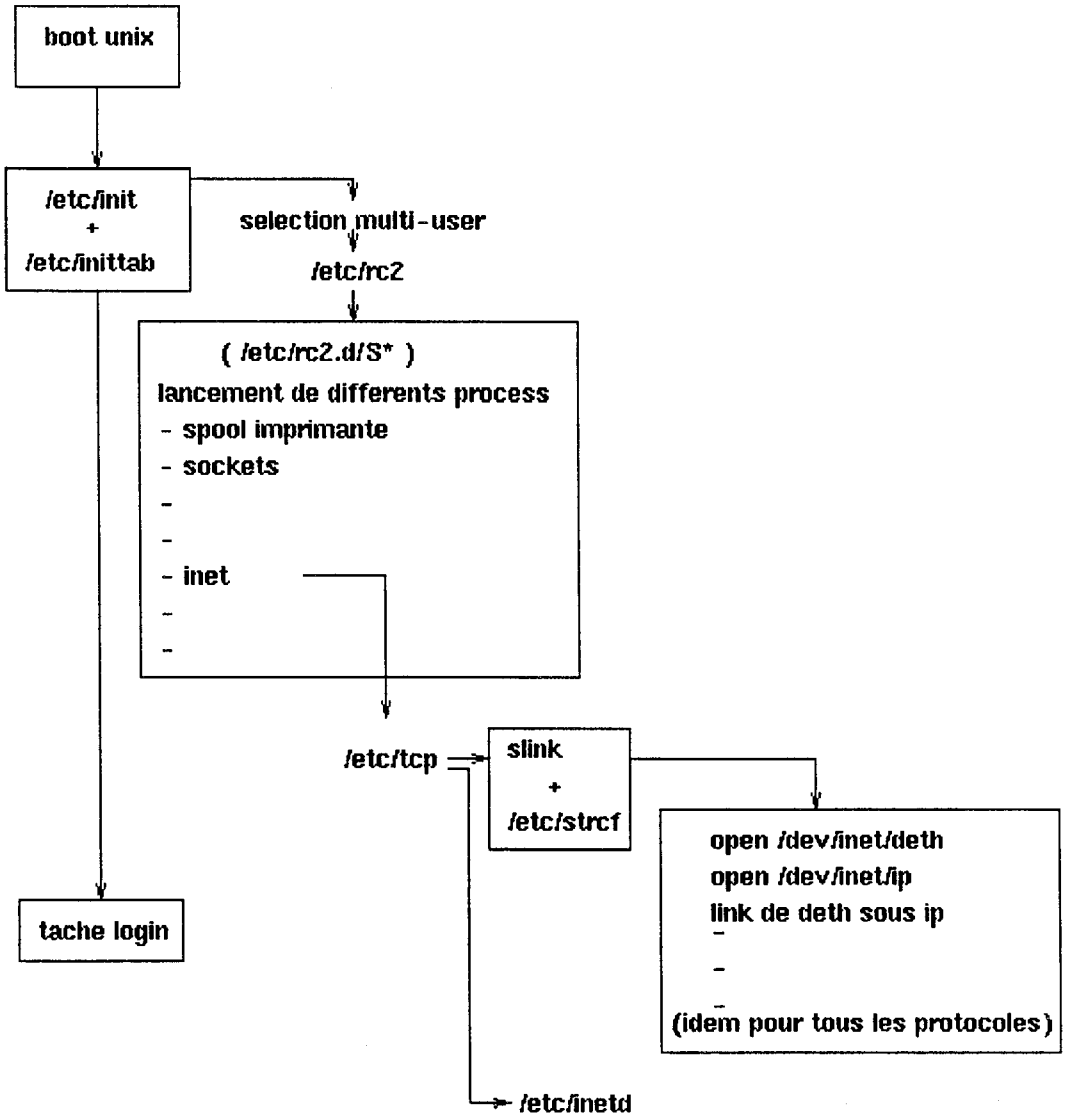


Fig. 3.2 chronologie des opérations

A l'issue de cette séquence d'opérations les drivers streams sont ouverts et organisés, de la couche transport à la couche physique .

Les différents "blocs" (tcp, ip, deth...) peuvent communiquer entre eux, sans référencer explicitement leur interlocuteur, en "postant" des messages dans la queue du stream. L'acheminement du message est pris en charge par le mécanisme des streams.

L'utilisation de cette suite de protocoles est alors disponible pour les utilisateurs. Elle est réalisée par l'appel de fonctions de XTI, qui ouvrent un stream entre l'utilisateur et TCP.

3.2.3 Mise au point d'un driver du noyau

En résumé, les communications sous Unix mettent en œuvre des mécanismes relativement complexes. Or la difficulté de la mise au point d'un driver est accrue par le fait qu'il faille intervenir à plusieurs niveaux : directement dans le noyau, au niveau de la configuration de la machine, et dans l'espace utilisateur. Une des conséquences immédiates de cette catractéristique a été la nécessité de disposer de deux "univers" : l'un pour le développement des programmes et l'autre pour la mise au point de ces programmes. Ceci nous a conduit à définir une méthodologie, où les opérations sont réparties suivant deux types : celles qui ne sont faites qu'une seule fois (configuration et intégration) et celles qui doivent être répétées dans tout le processus de mise au point du driver (debug).

Cette distinction a permis de définir la première étape de la réalisation, où trois processus ont été menés en parallèle : l'un d'eux consacré à l'étude des modifications de la librairie XTI, un autre à la spécification des interfaces du protocole XTP sous streams et le troisième à la mise en place d'un environnement de travail pour l'ensemble du projet (production de programme et machines de mise au point). La deuxième étape de la réalisation a été consacrée aux améliorations à apporter à notre prototype. Dans les paragraphes suivants, nous détaillerons chacun de ces aspects de la réalisation du projet.

3.3 Implémentation choisie

Le premier choix que nous ayons eu à faire concernait le code implémentant le protocole XTP, que nous allions intégrer. KRM a été retenu parce qu'il émane de la société même qui a défini le protocole XTP et qu'il nous était accessible gratuitement. En effet, dans le cadre d'un partenariat avec la société Bull, le Laboratoire de Génie Informatique (LGI) a prêté l'implémentation KRM dont il disposait, à des fins expérimentales. Il s'agit de **KRM 1.6** correspondant à la version 3.5 du protocole XTP. Cette implémentation avait déjà fait l'objet de travaux de portage en environnement transputer au sein du LGI [ROCA 91].

L'implémentation que nous avons choisie pour intégrer XTP découle d'une implémentation de TCP/IP sur STREAMS. Le code du protocole TCP/IP a été remplacé par le code de KRM.

TCP/IP est implémenté avec deux drivers multiplexés : TCP et IP. En-dessous de IP, le module **ARPPROC** est "poussé" : il est utilisé pour convertir les adresses Internet en adresses physiques.

Le driver XTP est un driver STREAMS multiplexé supportant le mécanisme dit "clone", il est chargé de l'administration du protocole. De même que pour TCP, le module **TIMOD** est "poussé" au–dessus du driver XTP, il assure un léger interfaçage, décrit plus tard, entre la tête du stream et le driver.

En–dessous de lui se trouve le driver DLPI, lui–même au–dessus des drivers du réseau. Au travers de DLPI, XTP utilise directement les services du contrôleur d'accès au medium (IEEE 802.x).

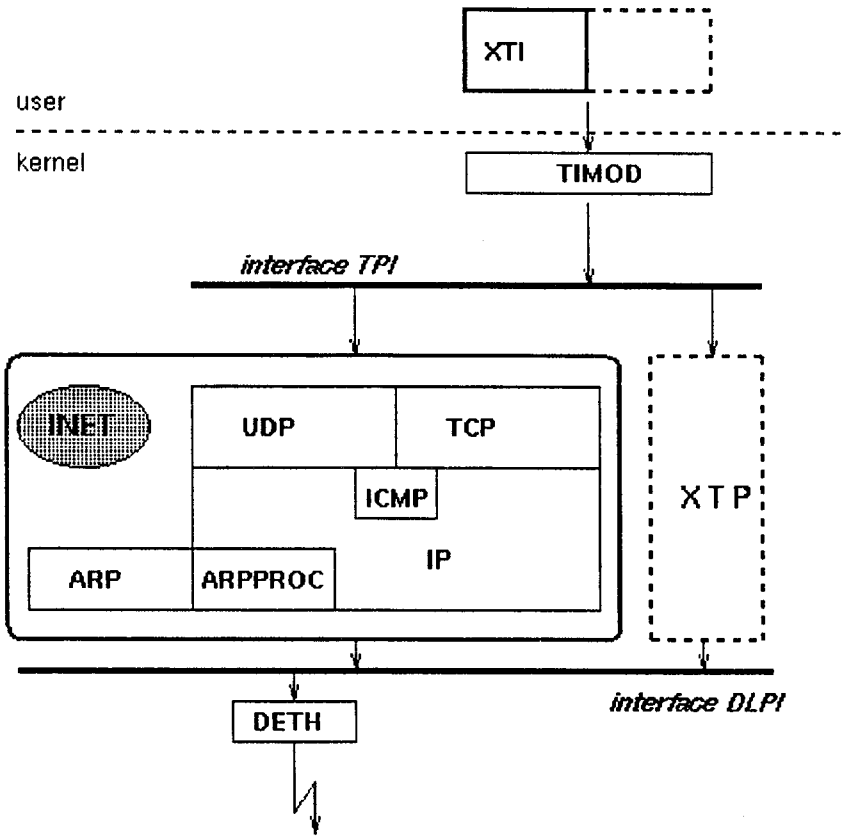


Fig. 3.3 organisation des modules et drivers

3.3.1 Contrôle de flux des streams

Chaque module d'un stream assurant le contrôle de flux, tel que le driver XTP, est constitué d'une paire de structures **queue** (une pour la lecture, une pour l'écriture). De plus, un driver multiplexeur nécessite une paire de structures **queue**

au niveau supérieur (upper) et une autre au niveau inférieur (lower). Ces structures sont allouées et initialisées quand un stream est créé ou quand un module est "poussé" sur un stream.

Ces structures permettent à l'infrastructure des STREAMS d'assurer le contrôle de flux. Par exemple, dans la configuration des STREAMS, le driver XTP étant placé au–dessus du driver DLPI, la queue d'écriture inférieure de XTP (lower write queue) est "reliée" à la queue d'écriture supérieure du driver DLPI (upper write queue). Le mécanisme des STREAMS autorisera donc le passage d'un message de XTP vers DLPI (émission de données) seulement si le driver DLPI est en mesure de prendre en charge ce message.

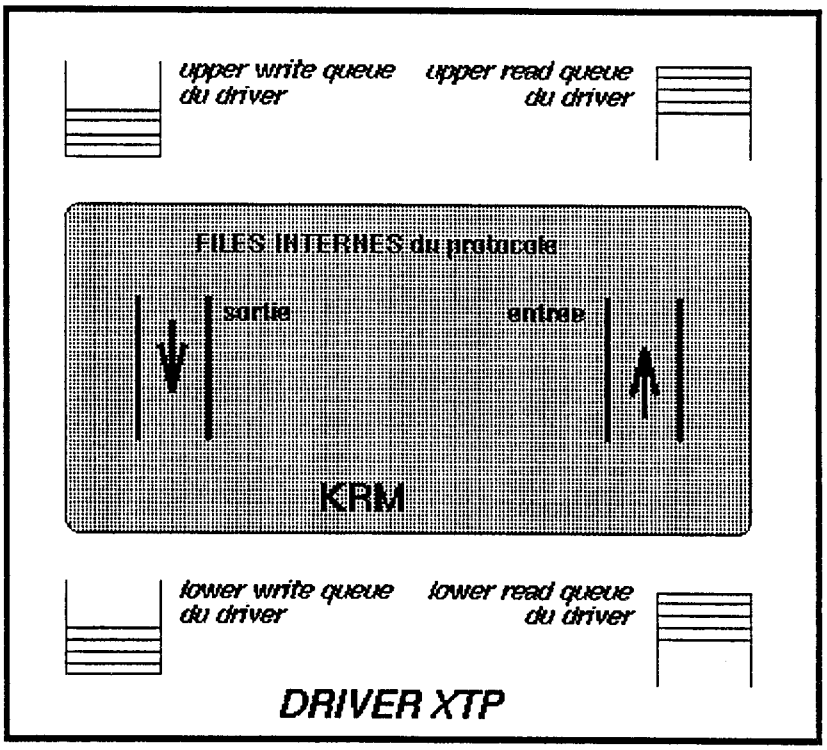


Fig. 3.4 structure du driver stream XTP

Les procédures associées à une queue sont les routines qui traitent les messages empruntant cette queue. Il en résulte généralement un message envoyé dans la même direction. Une queue doit contenir une procédure **put** (obligatoire) et une **procédure de service** associée (facultative).

La procédure *put* reçoit les messages de la queue précédente. L'appel de cette procédure dans la direction adéquate est le seul moyen de passer des messages entre les modules (module, driver et stream head).

En plus de la procédure *put*, les STREAMS autorisent une procédure de service, associée aux traitements différés. La procédure *put* est toujours appelée en premier, depuis la queue du stream précédent. Après avoir exécuté sa part de traitement des messages, elle s'arrange pour que la procédure de service soit appelée en transmettant le message à la fonction *putq*. Cette fonction réalise deux choses : elle place le message dans la file des messages en attente de la queue puis elle met la queue dans la file des queues en attente de l'ordonnanceur des streams (scheduler). Après le retour de la fonction *putq* dans la procédure *put*, la procédure de service est devenue "activable". Plus tard, elle sera appelée automatiquement par le scheduler.

En résumé, il y a trois possibilités pour traiter et passer les messages entre modules:

– *putnext* pour passer un message à la queue suivante qui le traite immédiatement.

– *putq* : la routine du driver activée par *putnext* ne traite pas immédiatement le message et l'insère dans la file des messages en attente de la queue du module destination. Le traitement du message est différé, la procédure de service sera appelée par le scheduler des streams.

– *putq* dans la file d'attente de la queue du module source quand la fonction *canput*, qui teste s'il y a de la place disponible dans la queue du module destination, refuse le message.

3.3.2 La tête du stream

Les requêtes doivent être mises en file d'attente en raison du fait que le noyau n'est pas toujours disponible pour les traiter au moment où elles arrivent.

La file de lecture contient les messages à délivrer à l'utilisateur, alors que la file d'écriture transporte les requêtes à envoyer dans le sens downstream vers le driver XTP. Les messages de contrôle utilisent les mêmes queues. Ils concernent les commandes IOCTL et quelques messages d'acquittement.

Les principales primitives utilisées pour les messages sont *putmsg* et *getmsg*.

3.3.3 Le module timod

Son rôle est d'assurer un interfaçage entre la tête de stream et le driver XTP. Dans la direction downstream, il complète le processus de génération des primitives TPI destinées au driver XTP alors que dans la direction upstream, il complète quelques réponses TPI vers l'application.

Timod contient uniquement les routines pour ouvrir, fermer et configurer le module, à côté des routines *timod_wput* et *timod_rput* qui assurent la translation des données. Il n'y a pas de procédure de service, les messages traversent le module timod sans délai. Il est transparent pour les données.

3.3.4 Le driver XTP

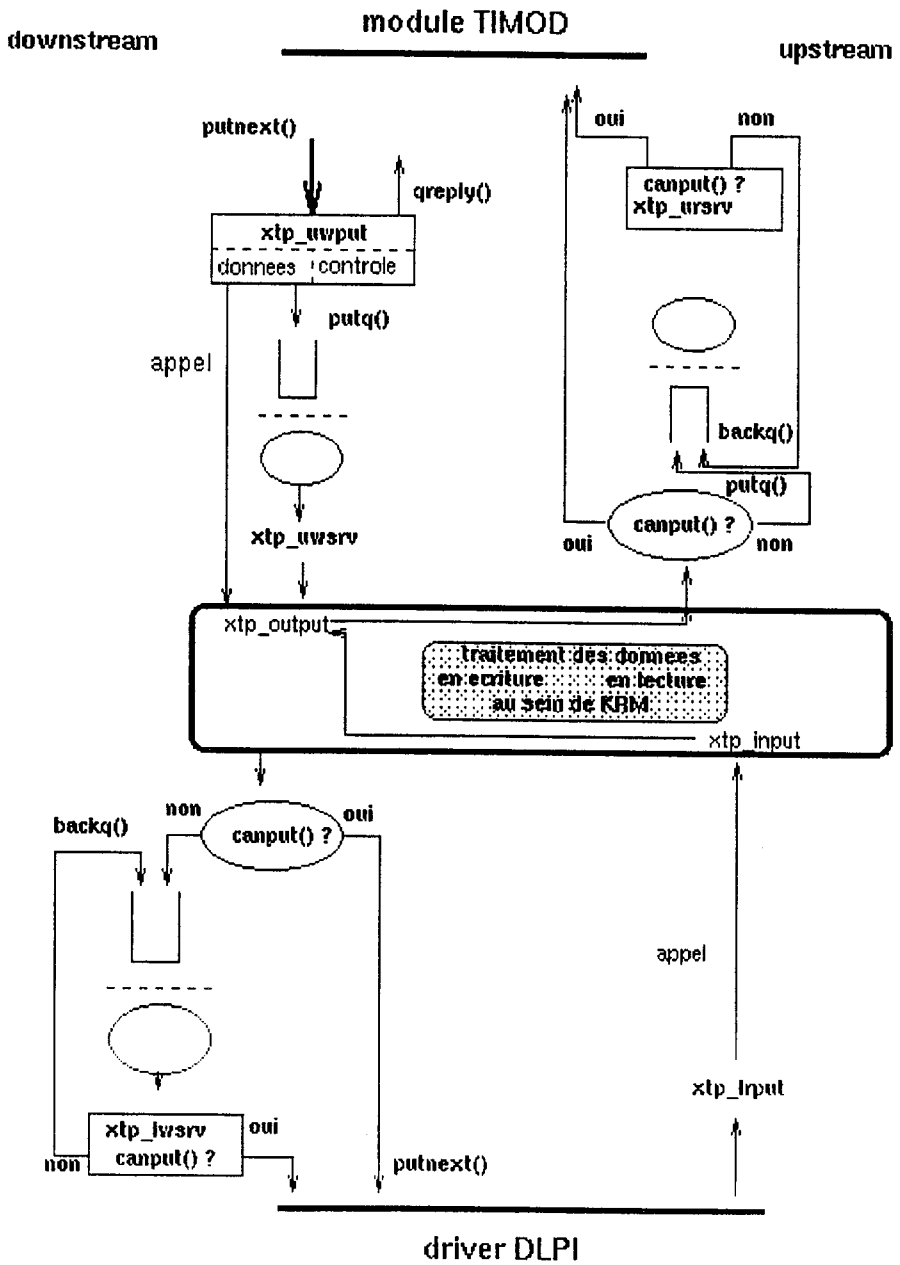


Fig. 3.5 implémentation de XTP sous STREAMS

En écriture (downstream)

Du côté écriture, le driver XTP ne fait rien à moins que quelque chose lui arrive via sa queue d'écriture, ce qui correspond à un des évènements suivants :

– une routine a exécuté un *putnext* en écriture dans la tête de stream, ce qui correspond à un appel par pointeur de la routine put de timod.

– la routine *put* de TIMOD a traité un message qu'elle avait reçu et appelle à son tour *putnext* en écriture; ce qui se traduit en un appel par pointeur de la routine *put* du driver XTP.

Il est important de noter que la routine de la tête de stream ayant initialisé cette séquence, est bloquée dans l'attente du retour de son appel *putnext*.

Au niveau du driver XTP, les évènements suivants se produisent:

La fonction *xtpuwput* réalise quelques contrôles et différentes opérations sur le message qu'elle vient de recevoir. Puis elle teste s'il s'agit d'un message de données ou d'un message de contrôle. Les messages de contrôle sont traités sans retard et les réponses sont envoyées par *qreply*. Pour les messages de données, *xtpuwput* teste si le protocole peut prendre en charge le message.

Le protocole maintient pour chaque connexion, une file d'émission qu'il fait évoluer au fur et à mesure des demandes d'envoi de données et de leur acquittement. Quand la file d'émission est saturée, la fonction *xtpuwput* se contente de mettre les messages qui lui arrivent dans la file d'attente du driver (*putq*), alors que si le message peut prendre place dans la file d'émission, il est traité immédiatement. De plus, si l'état de la connexion le permet, un paquet XTP est préparé à partir de ce message, et le driver XTP l'envoit au driver DLPI.

Au retour de la fonction *xtpuwput*, la routine de la tête de stream à l'origine de l'appel, reprend le contrôle.

L'appel de la routine standard *putq* a pour but de différer le traitement d'un message : une fois le message placé dans la file d'attente du driver XTP, la procédure de service du driver est activable pour une exécution ultérieure. Plusieurs messages peuvent s'accumuler ainsi dans la file d'attente du driver. Quand elle est appelée par le scheduler des streams, la procédure de service *xtpuwsrv* retire chaque message de la file d'attente du driver, un à un. Comme la fonction *xtpuwput*, si le protocole ne peut pas prendre en charge le message, la fonction *xtpuwsrv* remet le message à sa place dans la file d'attente du driver, par la fonction standard des STREAMS *putbq*.

Ainsi cette séquence d'opérations se répète jusqu'à ce qu'il n'y ait plus de message dans la file d'attente du driver XTP, dans le sens downstream.

En lecture (upstream)

Dès qu'un paquet est reçu par le driver XTP en provenance de la couche liaison de données, la fonction *xtplrput* est appelée depuis la routine *putnext* du driver DLPI. Aucun contrôle de flux n'est effectué au niveau de cette routine, elle appelle directement la fonction d'entrée du protocole.

Cette fonction analyse l'entête du paquet. S'il s'agit d'un paquet de données arrivées à destination, les données sont mises dans la queue de lecture du protocole. Si les données reçues sont contiguës, elles peuvent être délivrées à l'utilisateur et retirées de la queue de lecture de XTP. A ce niveau le contrôle de flux des streams intervient; les données qui ne peuvent pas être passées au module TIMOD sont stockées dans la file d'attente du driver XTP, en lecture.

Contrôle de flux

Les STREAMS fournissent un contrôle de flux vertical alors que le protocole XTP assure un contrôle horizontal.

Verticalement, le résultat positif d'un *canput* permet le transit des messages entre les modules sans attente. Dans de tels cas, il n'y a pas de perte de temps dans la gestion interne des files d'attente et dans les traitements différés des messages.

Ceci est fortement conditionné par le contrôle de flux horizontal, notamment par le choix de la taille des fenêtres et de la fréquence des acquittements : la fenêtre ne doit jamais être fermée et il ne doit pas y avoir d'attente d'acquittement. Quand le récepteur est plus lent que l'émetteur, le contrôle de flux vertical sera effectif du côté de l'émetteur.

3.3.5 Organisation des buffers

Dans l'implémentation que nous avons choisi de réaliser, les données ne doivent être copiées qu'une seule fois pour passer de l'espace utilisateur à l'espace noyau, et vice versa. Elles doivent traverser l'espace noyau dans les deux sens sans copies superflues et pénalisantes du point de vue du temps d'exécution.

Le mécanisme des STREAMS maintient un ensemble de stockage privé pour ses messages. Une procédure (*allocb*) peut demander l'allocation d'un message d'une taille précise. Un message stream consiste en un ou plusieurs blocs de messages liés entre eux. La première cause des blocs multiples dans un seul message est due au mécanisme des STREAMS lui–même qui limite la taille des blocs à quatre kOctets; l'envoi de 4097 octets nécessite donc deux blocs.

L'intérêt majeur des blocs multiples dans un seul message est la possibilité de modularité : il est ainsi permis de choisir un bloc et de le chaîner avec un autre bloc quelconque. On créé ainsi une entité logique moyennant très peu de modifications physiques. Il s'agit d'un mécanisme très utile pour la création d'un paquet : il permet l'adjonction d'informations (header et trailer) sans recopier les données.

3.3.6 Adressage

XTP supporte plusieurs modes d'adressage, notamment l'adressage Internet que nous avons choisi d'employer dans notre implémentation; d'une part, afin de pouvoir utiliser les mêmes fonctions du serveur de nom que les applications sur TCP, et d'autre part, afin de pouvoir appeler, depuis le driver XTP, des fonctions communes à TCP–IP notamment pour trouver la route et l'adresse réseau lors de l'envoi d'un paquet FIRST.

3.4 Stratégie adoptée

La méthodologie que nous avons adoptée pour la création de cette plateforme de communication est basée sur le principe d'un processus qui capitalise. Nous avons choisi de procéder par étapes à partir d'un noyau standard, y intégrer un protocole XTP inactivable d'abord, puis le rendre utilisable ensuite, performant enfin. L'étude des extensions à apporter à la librairie XTI a été menée en parallèle avec les travaux sur le protocole XTP.

3.4.1 Première phase : mise en place d'un "squelette"

Nous avons choisi de passer par cette première étape en raison de la complexité des moyens que nous devions mettre en œuvre pour intégrer un nouveau protocole. Le but poursuivi était de mettre en place toute l'infrastructure nécessaire au développement et à la mise au point du projet ainsi que d'en valider le fonctionnement, sans se préoccuper outre–mesure du contenu précis du code que nous intégrions, ce dernier point faisant l'objet du reste de l'étude.

Structure d'accueil

L'implémentation choisie pour intégrer XTP découlant d'une implémentation de TCP/IP sur streams, nous avons donc procédé par symétrie dans l'arborescence des sources BOS 2.0. Un répertoire XTP a été créé suivant les mêmes critères que celui utilisé pour TCP/IP, seuls les fichiers sources ont été remplacés par ceux de KRM.

Choix d'un mode de fonctionnement

La première difficulté à laquelle nous nous sommes heurtés résidait dans le fait que KRM propose un grand nombre d'options diverses, dépendantes les unes des autres dans certains cas, ce qui donne certainement beaucoup de souplesse au code mais le pénalise au niveau de sa lisibilité, ceci étant encore accentué par l'absence de documentation.

Nous ne détaillerons pas ici les différentes options. Précisons cependant que nous avons opté pour un mode de fonctionnement (STANDALONE) qui semblait prévu pour des machines ne supportant pas de système de communications. Dans ce mode, KRM définit lui–même ses propres structures de données, organise lui–même la gestion de sa mémoire et utilise une palette restreinte de fonctions du noyau. Ainsi le code de KRM est relativement autonome, plus adapté à une intégration dans notre système unix.

Compilation et édition de liens

Le but poursuivi était de créer par nous–mêmes un noyau unix incluant un protocole XTP. Pour cela la première opération nécessaire était de déclarer le driver XTP dans la table des devices du système d'exploitation (cdevsw du système unix). Le numéro d'entrée dans la cdevsw ainsi attribué est appelé le majeur (128 pour xtp), il sera utilisé ensuite pour la création du fichier spécial "/dev/xtp". L'ensemble permet au système d'exploitation d'établir la corrélation entre l'identification logique (open("/dev/xtp") par exemple) employée par l'utilisateur et la description physique du driver xtp, telle qu'elle apparait dans la structure streamtab xtp_info, présente à l'emplacement 128 de la cdevsw du système unix que nous avons généré. En fait, les structures streamtab présentes dans la cdevsw effectuent la déclaration implicite des différentes fonctions des modules et drivers intégrés au noyau.

Comme dans beaucoup d'intégrations, l'introduction d'un ensemble de fichiers dans le système existant a amené son cortège de définitions multiples ou manquantes. De plus, pour l'absence de certaines définitions, le manque de documentation sur KRM nous a conduit à adopter une résolution par intuition.

Dans un tout premier temps nous avons utilisé le mécanisme de génération automatique des Makefile identique à celui de TCP/IP. Il s'agit d'un ensemble de commandes qui analyse la composition d'un répertoire, établit la liste des fichiers sources, détermine leurs dépendances et fixe la liste des objets à archiver dans la librairie de ce répertoire.

Néammoins, ce procédé s'est avéré trop lourd pour cette phase d'intégration, puisqu'à chaque modification dans la composition des "include", il fallait relancer toute la procédure. Or, cette première phase était constituée essentiellement de tatonnements dans l'organisation des "include" de KRM avant de pouvoir compiler l'ensemble des fichiers et réaliser l'édition de liens générant le noyau unix. Nous avons donc dû procéder manuellement en compilant "pas à pas" chacun des fichiers composant les 17 000 lignes de langage C que nous intégrions.

A l'issue de cette suite d'opérations le système de production de programme était mis en place pour la partie noyau, les procédures automatiques étaient utilisables.

Activation du code : application et interface

Le code de l'application BENCH utilisant TCP/IP et UDP a été porté sur XTP sans modification. Quant à la librairie XTI, nous avons dû modifier un seul fichier : la fonction t_open qui effectue un contrôle sur la chaine de caractères passée en argument de l'appel de la fonction. Il fallait donc lui faire reconnaitre "/dev/xtp".

A ce niveau, les modifications ont été négligeables en nombre de lignes de code.

Installation d'une machine de test

La machine de mise au point est un DPX2 équipé d'une carte mère MTB3 avec un microprocesseur 68030 de Motorola et d'un coupleur Ethernet connecté au réseau local du centre Bull d'Echirolles. Le système d'exploitation que nous avons installé est un BOS2 standard donc sans driver DLPI. La phase de préparation de la machine de test constituait un préliminaire indispensable à toute opération de debug du protocole XTP.

Afin d'utiliser les drivers DLPI et XTP, nous avons introduit les fichiers spéciaux nécessaires dans le file system de la machine ("/dev/dlpi" et "/dev/xtp") grâce à la commande unix mknod, avec les numéros de majeur définis précédemment lors de la compilation et l'édition de liens : mknod /dev/xtp c 29 128 (les paramètres "c 29" indiquent qu'il s'agit d'un driver de type caractère supportant le mode clone).

De plus, il a été nécessaire de modifier le fichier strcf utilisé par la commande slink qui est lancée à l'initialisation des communications après le chargement du système; slink a en charge de lier les différents modules et drivers STREAMS entre eux. STRCF décrit la configuration des streams, nous y avons introduit l'ouverture d'un stream XTP, stream multiplexeur placé au–dessus du driver loopback et du driver DLPI lui–même au–dessus de l'interface Ethernet.

Enfin, nous avons installé l'application BENCH prévue pour XTP.

A l'issue de cette séquence notre machine de test pouvait recevoir le noyau unix avec les drivers DLPI et XTP. La mise au point du driver XTP était dorénavant possible, par strcf pour la configuration du stream XTP puis par BENCH pour le protocole lui–même.

Définition d'une procédure de mise au point

Les environnements production de programme et mise au point étant initialisés, nous pouvions désormais nous consacrer au code du protocole.

Après toutes les opérations de codage, compilation et édition de liens, le noyau généré était transféré sur la machine de mise au point grâce au réseau quand c'était possible, par support magnétique sinon. Il fallait alors recharger (reboot) le système pour mettre au point notre code au moyen de KDB (Kernel DeBuger), après avoir lancé l'application BENCH.

L'ensemble de ces opérations constituait dorénavant la séquence de base pour toute modification du code, aussi minime soit–elle.

Résultat

En anticipant par rapport à une phase d'intégration placée habituellement après le codage, cette étape nous a permis de résoudre un maximum des problèmes apparemment secondaires dans notre étude et de consacrer l'essentiel du temps à la partie prépondérante de notre projet : l'adaptation de KRM et l'étude de ses performances.

A l'issue de cette étape nous disposions d'un arbre de référence, point de départ des quatre arborescences de travail sur lesquelles nous avons pu réaliser les différentes évolutions du prototype.

3.4.2 Création du prototype

La création du premier prototype s'est faite au sein de l'arborescence de référence qui venait d'être créée. Aucune des trois interfaces proposées par KRM n'était compatible avec les STREAMS. Il s'agissait donc de construire l'interface en s'inspirant des trois précédentes pour ce qui concerne les fonctions à activer à l'intérieur du protocole, en respectant les exigences des STREAMS d'une part et des interfaces TPI et DLPI d'autre part.

Les routines de gestion du stream (xtp_open et xtp_close) initialisent et libèrent les structures privées nécessaires à l'établissement des connexions futures et au fonctionnement du protocole. Pour le traitement des requêtes TPI et DLPI les traitements à faire sont de deux types : les opérations de contrôle et les opérations de transfert de données.

Les opérations de contrôle sont faites directement autour des structures STREAMS : décodage des messages véhiculant la primitive, appel des procédures KRM et création d'un message de requête et/ou de réponse. Pour les opérations de transfert de données, le principe diffère au niveau du traitement des données. En effet l'hétérogénéité des structures STREAMS et KRM nécessite un travail important d'adaptation. La solution retenue a été, dans un premier temps, de recopier purement et simplement les données dans les structures adaptées, à chaque transition STREAMS/KRM et KRM/STREAMS. Sachant que ce procédé était très pénalisant au niveau des performances, il ne pouvait être que temporaire.

L'optimisation de l'utilisation de la mémoire pour le flux des données ainsi que l'homogénéisation des structures ont fait l'objet d'une étude ultérieure, ce travail ayant nécessité une étude approfondie du chemin des données à l'intérieur du protocole.

3.4.2.1 XTP sur le driver loopback

KRM offre neuf modes d'adressage dont l'adressage Internet et l'adressage direct. Nous avions choisi l'adressage Internet de KRM, cependant l'implémentation des protocoles Inet que nous utilisons nous a imposé de faire un compromis entre les deux modes d'adressage : direct et Internet.

Dans l'environnement initial où l'adressage Internet est employé, un module ARPPROC (module au sens STREAMS) assure la traduction des adresses Internet en adresses Ethernet pour les datagrammes qui le traversent; pour cela, il exploite la table du protocole ARP et une fonction arpresolve.

Une des particularités du protocole XTP est qu'il maintient lui–même sa propre table de routage, qu'il initialise une fois pour toutes au cours de la poignée de main d'ouverture de la connexion. Il a besoin à ce moment–là de connaitre l'adresse Internet et l'adresse Ethernet correspondante. Or le module ARPPROC, tel qu'il est implémenté, ne permet pas d'avoir cette information au niveau supérieur.

Une des solutions possibles était l'extension du module ARPPROC pour lui faire "remonter" l'information vers XTP, ce qui risquait de nous entrainer plus loin que ce que nous avions prévu et de nous faire manquer la première échéance de notre calendrier.

Une autre solution consistait à choisir un mode d'adressage de KRM plus simple et de prévoir son adaptation pour la phase suivante. L'adressage direct correspondait bien à cette exigence. Initialement prévu pour des applications nécessitant un circuit virtuel permanent ou pour des systèmes de communication dont la topologie est fixée, il présentait l'avantage d'être utilisable immédiatement pour une version de KRM en mode rebouclé, tout en offrant la possibilité d'être utilisé dans des conditions plus réelles, moyennant quelques modifications.

Ainsi, à la date prévue, la première ébauche du protocole XTP fonctionnait en exploitant les mécanismes de l'adressage direct de KRM pour accéder à une destination par défaut, celle du driver loopback. L'étape suivante nous imposait d'adapter KRM pour remplacer cette adresse fixe par une adresse Internet, afin d'observer le protocole en situation réelle.

3.4.2.2 XTP sur Ethernet : réseau privé

Adressage

Une adresse directe (DADDR) se compose d'un champ daddr_key sur quatre octets contenant un identificateur choisi par l'application, associé à un champ daddr_mac, adresse Ethernet sur six octets. Les procédures de routage sont simplifiées et ne permettent de dialoguer qu'avec des machines situées sur le même réseau local.

Dans un premier temps et tant que nous étions en mode rebouclé, nous avions conservé cet adressage en l'état. Nous avons ensuite effectué quelques modifications afin de permettre à l'application d'utiliser des adresses Internet de la même façon qu'elle le ferait si elle dialoguait au moyen d'une configuration TCP/IP. Le passage entre adresses Internet et Ethernet se fait alors en consultant la table de translation du protocole ARP.

Dans une configuration TCP/IP où le module ARPPROC est inséré en–dessous du driver IP, la conversion adresse Internet/Ethernet est effectuée pour chaque paquet. Avec le protocole XTP, ce n'est plus nécessaire, l'adresse Ethernet étant mémorisée dans les structures de routage. Ainsi, lors de la création d'une route, nous effectuons une recherche dans la table ARP et par la suite, nous réutilisons cette adresse; il n'y a plus lieu d'insérer le module ARP en–dessous de XTP.

Routage

A chaque interface réseau, XTP associe une structure de données (sxtp_provider) initialisée par le système lors de son lancement. Au sein de cette structure sont regroupées toutes les caractéristiques physiques de l'interface (tailles maximale/minimale de paquets, adresse au sein du réseau) et logiques (adresse internet, nom logique de l'interface du type "de0", "lo0"...). Toutes ces structures sont regroupées au sein d'une table.

A chaque connexion ouverte, il est défini une route au sein du ou des réseaux ainsi que dans les stations extrêmes. Afin de gérer les transferts au sein de cette route, XTP créé pour chaque station impliquée (y compris les éventuels routeurs intermédiaires) une structure (xtp_rtentry) dans laquelle sont regroupées les informations permettant au protocole d'échanger des données sur le réseau dans les meilleures conditions possibles (contrôle de flux, de débit, identificateurs utilisés pour le routage des trames...).

Chaque connexion utilisant un réseau, XTP créé également un chainage entre structure de routage et structure d'interface.

Dans un premier temps, travaillant en mode rebouclé, nous n'avons ouvert et integré au–dessous d'XTP que l'interface loopback. Ainsi, tout paquet à transmettre doit impérativement passer par l'unique interface disponible. Aucune recherche n'est nécessaire, nous utilisons le premier et unique élément de la table des interfaces.

Dans un deuxième temps, en présence à la fois de l'interface loopbak et de l'interface Ethernet, il a été nécessaire d'effectuer un choix, pour chaque connexion. Lors de sa demande d'ouverture de connexion, l'application doit fournir l'adresse de son correspondant. Cette adresse est tout d'abord comparée à l'adresse internet du driver loopback, adresse standard fixée sur toutes les machines à 127.0.0.1.

S'il y a identité, nous lions la structure de routage avec la structure décrivant le driver loopback (reconnu par son nom logique "lo0"), sinon nous parcourons la table des interfaces et nous lions la structure de routage avec la structure décrivant la première interface qui ne soit pas le driver loopback (nom logique différent de "lo0").

3.4.2.3 Résultats

Au terme de l'étape de création du prototype, nous sommes parvenus à intégrer le protocole XTP dans l'environnement STREAMS, en l'adaptant aux interfaces TPI et DLPI de façon transparente pour les applications; le programme BENCH, qui tourne au–dessus de TCP/IP, a été porté sans modification.

Pour ces premières mesures nous avons paramétré BENCH avec une taille de tampon (Transport Service Data Unit) de un kOctet. En mode rebouclé, les premières performances que nous avons pu mesurer pour la transmission de ces TSDU étaient de185 kOctets par seconde (234 kOctets/s pour TCP/IP). Le passage sur le réseau Ethernet a encore réduit ces performances (145 kOctets/s pour la même taille de TSDU). Cette chutte a trois raisons principales :

– la différence de profondeur du code traversé par les données

– la recopie des données sur le support physique alors que le driver loopback se contente d'un passage de pointeurs.

– les délais dus à la gestion du support physique alors que pour le driver loopback, un seul processus prend en charge le transfert des données. Sur Ethernet, le temps de réception des acquittements est mêlé avec les émissions de données. En mode loopback, il s'agit de deux processus s'exécutant sur un même processeur, émission des données et acquittements seront donc traités séquentiellement et comptabilisés séparément.

L'étude théorique des spécifications de XTP laissait espérer des résultats contraires. Néammoins, la jeunesse de notre prototype ajoutée aux options que nous avons prises pour une première implémentation peuvent expliquer cette constatation.

En effet, KRM manipule des structures de données hétérogènes à celle des STREAMS, ce qui entraine des pertes de temps dans les différentes opérations de formatage des données à chaque transition STREAMS / KRM (recopies de buffers). De plus, KRM dispose de ses propres queues de stockage dont la gestion vient encore alourdir le chemin des données, en raison des redondances existant entre les mécanimes fournis par les STREAMS et ceux de KRM, notamment pour le contrôle de flux.

En outre, pour la stratégie des demandes de paquets de contrôle, l'option prise pour tester KRM est celle donnée par défaut, à savoir un paquet de contrôle est envoyé systématiquement pour acquitter les données reçues et délivrées à l'utilisateur.

Tous ces facteurs, liés à l'implémentation, constituent un handicap pour apprécier rigoureusement les performances du protocole XTP par rapport à celles de TCP/IP. Les étapes suivantes ont pour objet d'améliorer cette implémentation de XTP. Elles portent essentiellement sur les points suivants : la minimisation des nombreuses recopies dans le chemin des données et l'écriture d'un algorithme définissant une stratégie optimisée pour les demandes d'acquittements.

3.4.3 Améliorations

Cette phase du projet a été consacrée à la recherche de solutions pour augmenter les performances du prototype que nous avions réalisé.

La nécessité d'homogénéiser les structures STREAMS/KRM s'est imposée tout de suite comme une priorité. Il était inconcevable de faire des mesures comparatives entre XTP et TCP/IP avec autant de recopies des tampons de données.

La politique d'acquittement utilisée par défaut dans KRM consiste à envoyer un accusé de réception pour chaque TSDU (unité de données du service transport) reçue. Or, des acquittements trop fréquents peuvent être une surcharge, il était donc important de rechercher une solution offrant la sécurité et améliorant la rapidité.

3.4.3.1 Optimisation du flux de données

Cette étape nécessitait de définir une structure de données permettant la traversée du driver XTP de part en part, sans aucune recopie physique des données, comme cela existe dans TCP notamment.

Une fois résolu ce problème d'hétérogénéité dans les structures, il était intéressant de rechercher les redondances dans les traitements afin de les éviter, ce qui nous a conduit essentiellement à supprimer les structures Control Bloc de KRM qui réalisaient un contrôle de flux similaire à celui pratiqué par les STREAMS.

Nouvelle structure de données

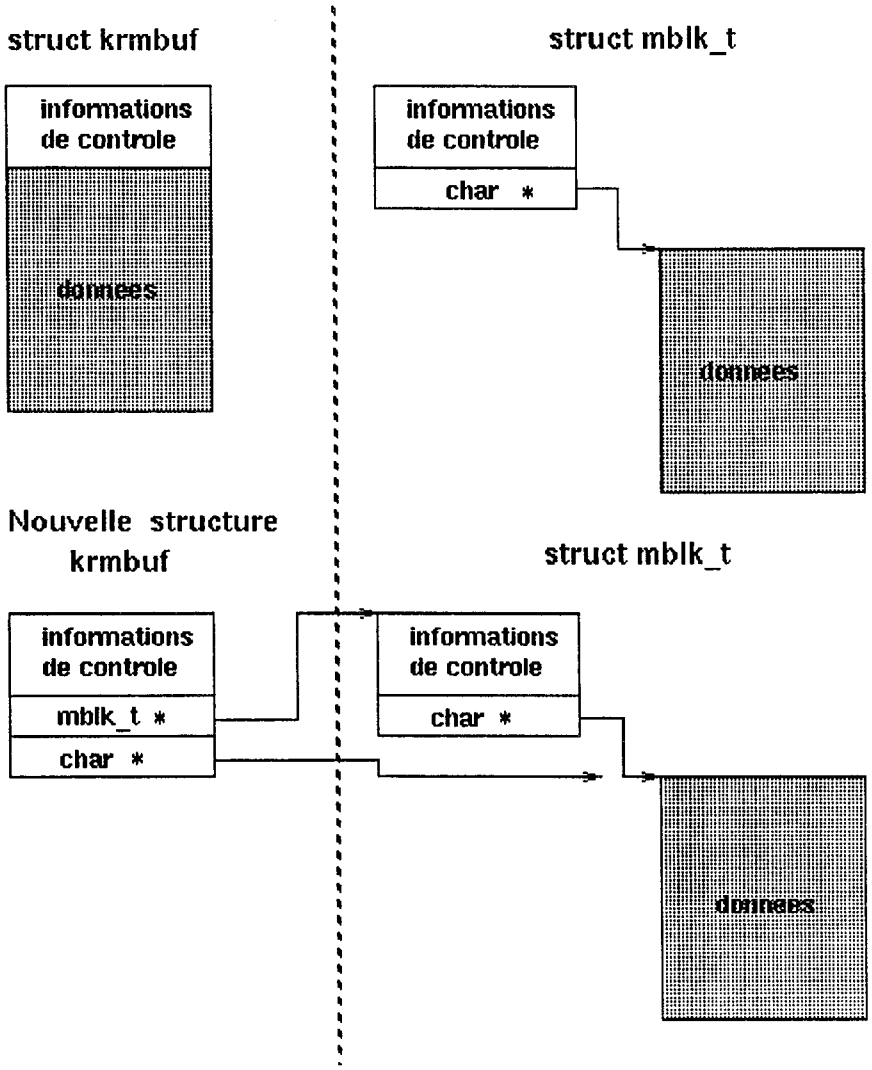


Fig. 3.6 structures de données

KRM maintient un ensemble de tampons (le pool de krmbufs) qu'il gère de façon autonome. Le krmbuf représente l'unité de base de cette gestion. Utilisé principalement pour le stockage des données, il est "retiré" du pool au moyen de la fonction *krmbuf_get* et restitué par la fonction *krmbuf_free*.

Une des solutions envisagées était de remplacer cette structure krmbuf par la structure des STREAMS mblk_t. En effet, il semblait possible d'établir une corrélation entre la plupart des champs de ces deux structures. Néammoins, un problème se posait pour un champ de krmbuf contenant des informations de contrôle pour lequel nous n'avons pas trouvé d'équivalence ni de moyen de remplacement dans les structures STREAMS. De plus, cette solution était très coûteuse en temps de travail, puisqu'elle nécessitait de reprendre l'intégralité du codage de KRM.

Le modèle que nous avons retenu est une solution de compromis. Nous avons légèrement modifié la structure krmbuf et établi une convention : à chaque structure mblk_t qui arrive au driver est associée une structure krmbuf (lien par pointeur), et à l'intérieur de KRM, à chaque krmbuf attribué est alloué une structure mblk_t. Ainsi toutes les procédures de KRM peuvent s'exécuter, elles retrouvent les informations nécessaires dans les mêmes champs.

De plus il n'y a plus de recopie physique des données. En effet, la fonction *dupb* des STREAMS permet de créer plusieurs blocs de messages qui pointent physiquement sur les mêmes données. L'environnement STREAM assure la gestion de ce mécanisme, bien adapté aux protocoles de transport qui doivent conserver "un exemplaire" des données qu'ils envoient jusqu'à leur acquittement.

Suppression de la structure bloc de contrôle

Les structures de données CB (Control Bloc) assure l'interfaçage entre KRM et l'application utilisatrice, elles étaient donc redondantes avec l'environnement STREAMS.

Dans le sens upstream, l'écriture de l'interface haute du driver avait tout naturellement supprimé leur utilisation, en raison de leur situation en fin de traitement des données.

Dans le sens downstream, le mécanisme était légèrement plus complexe. Les opérations de contrôle de flux des STREAMS ajoutées à celles du protocole imposaient d'attendre la résolution du problème de la recopie des données avant de supprimer cette structure. Jusque là le risque subsistait de rencontrer la situation où KRM ne pourrait accepter qu'une partie des données d'une requête TPI, alors que du point de vue des STREAMS la requête était satisfaite.

Implémentation retenue

Dans le sens downstream, au fur et à mesure de leur arrivée et en fonction du contrôle de flux de KRM (suivant le principe d'un buffer élastique), le lien mblk_t/krmbuf est créé et la structure obtenue est chaînée dans la file de sortie du protocole.

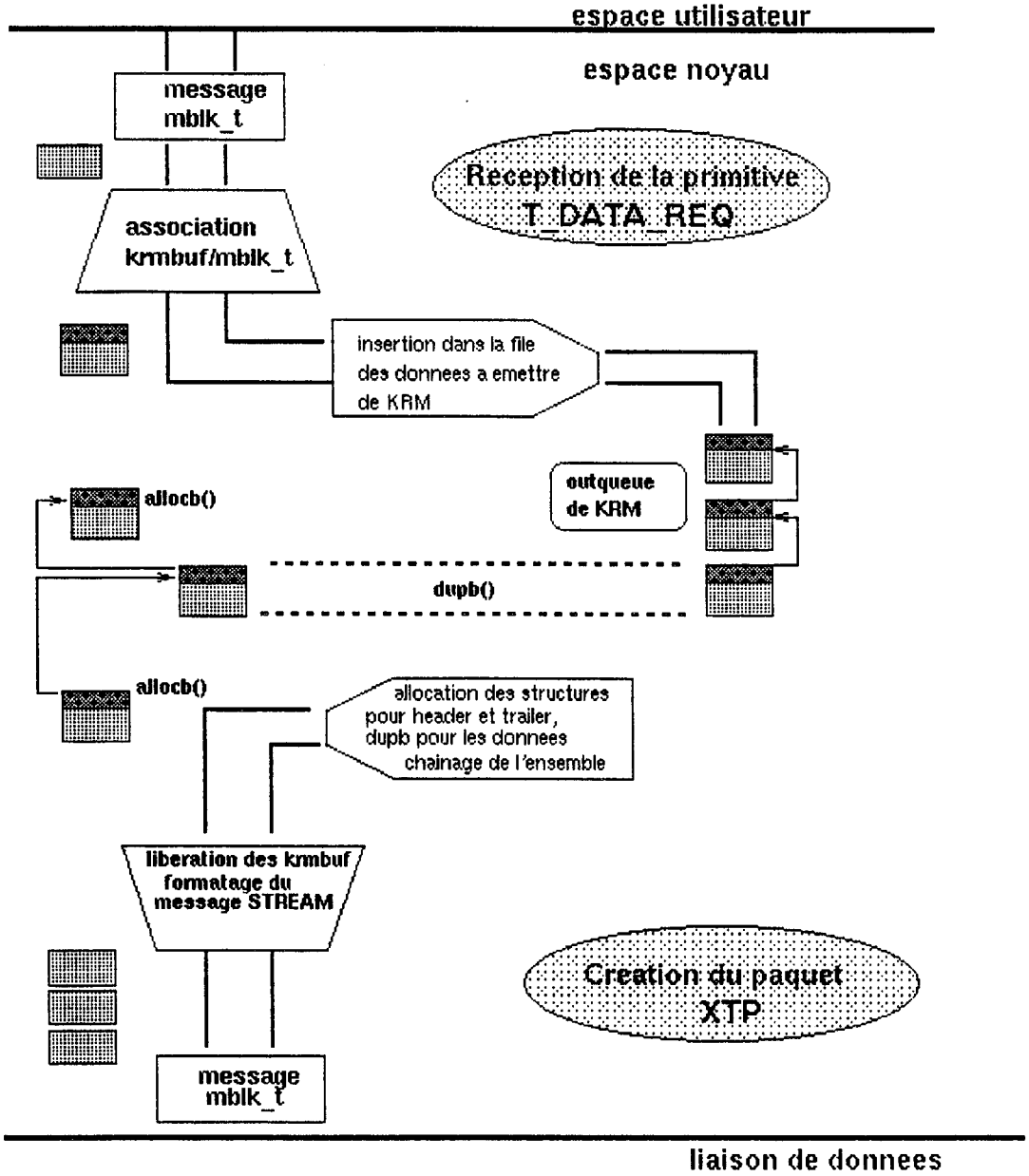


Fig. 3.7 flux des données en émission

Dès que c'est possible (états des interfaces, du réseau, de la fenêtre,....) le protocole crée un paquet pour émettre les données :

– allocation d'une structure mblk_t/krmbuf pour le header

– dupb sur le(s) bloc(s) de données à émettre (structure mblk_t/krmbuf), stockée(s) dans la file de sortie du protocole.

– allocation d'une structure mblk_t/krmbuf pour le trailer

– chainage de l'ensemble de ces structures constituant le paquet

– avant d'envoyer à DLPI le message STREAM ainsi constitué, suppression des structures krmbuf devenues superflues.

Dans le sens upstream, le mécanisme est similaire. Pour les autres types de paquet, tel que le paquet de contrôle, le mécanisme diffère en ce sens que le paquet est créé de toutes pièces dans une seule structure mblk_t/krmbuf.

Cette technique a pour avantage majeur d'éviter toutes recopies physiques des données, elle a cependant un léger inconvénient : le paquet XTP nécessite au moins trois messages STREAMS lors de sa création. Néammoins le gain qu'elle amène dans le temps de traversée du protocole améliore sensiblement les performances (voir figure 3.8).

3.4.3.2 Politique d'acquittement

XTP offre la possibilité d'adapter l'envoi des acquittements en fonction des besoins des applications. Il est apparu intéressant de tirer parti de cette facilité dans le but d'améliorer les performances. Les acquittements sont véhiculés par les paquets de contrôle. Ils ont pour rôle d'une part de permettre à l'émetteur de mettre à jour sa fenêtre d'émission et d'autre part de "purger" sa file d'émission. XTP offre la possibilité d'adapter l'envoi des acquittements en fonction des besoins des applications. Il est apparu intéressant d'étudier cette facilité dans le but d'améliorer les performances.

Pour la fenêtre d'émission, le seul acquittement nécessaire à la poursuite de l'émission sans interruption est celui qui arrive à l'émetteur juste avant qu'il ne soit bloqué par le contrôle de flux.

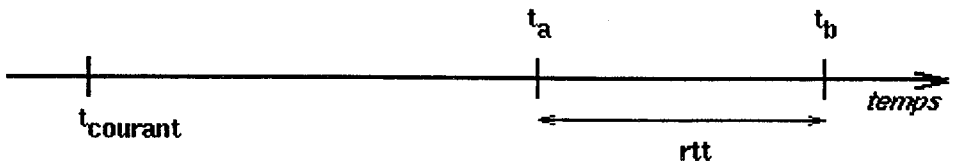
Pour le contrôle des erreurs, la garantie pour l'émetteur que ses données sont parvenues dans l'ordre et sans erreur n'est pas compromise par l'omission de quelques acquittements pour deux raisons :

– la sémantique de ces acquittements. Le récepteur affecte au champ *rseq* du segment de contrôle le numéro de séquence du prochain octet qu'il attend signalant ainsi qu'il a reçu sans erreur toutes les données jusqu'à *rseq –1* . Il n'a donc pas besoin de tous les acquittements intermédiaires. On se trouve en fait dans un cas analogue à la perte non fatale d'un acquittement décrite par La Baguette.

– la perte de paquets n'est pas particulièrement grave grâce à la stratégie de retransmission sélective de XTP. Ceci n'est pas vrai avec un protocole à retransmission cumulative. En effet diminuer les acquittements risque en cas de perte de données de retarder la détection de l'erreur et donc d'augmenter le volume des données à retransmettre, ce qui aurait des conséquences fatales sur le débit.

mise en œuvre

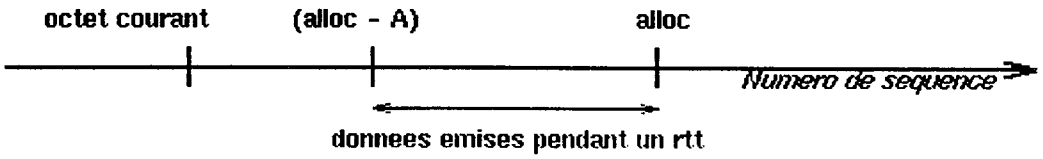
La première remarque est que les acquittements doivent être commandés par l'émetteur de façon à décharger le récepteur de la tâche de décision.



Il faut prévoir l'instant t_b où l'émetteur aura transmis le dernier paquet de sa fenêtre et attendra l'arrivée d'un acquittement pour se débloquer, calculer l'instant ($t_a = t_b$ – rtt) et demander un paquet de contrôle au moyen de la dernière TPDU (unité de données du protocole de transport) envoyée avant l'instant t_a. (le rtt est le temps séparant l'émission d'une TPDU de la réception de son acquittement).

La détermination de cet instant n'est pas facile : le rtt dépend fortement du réseau, la prévision de l'instant t_b est étroitement liée au débit, la précision de l'horloge étant de 20 ms sous unix.

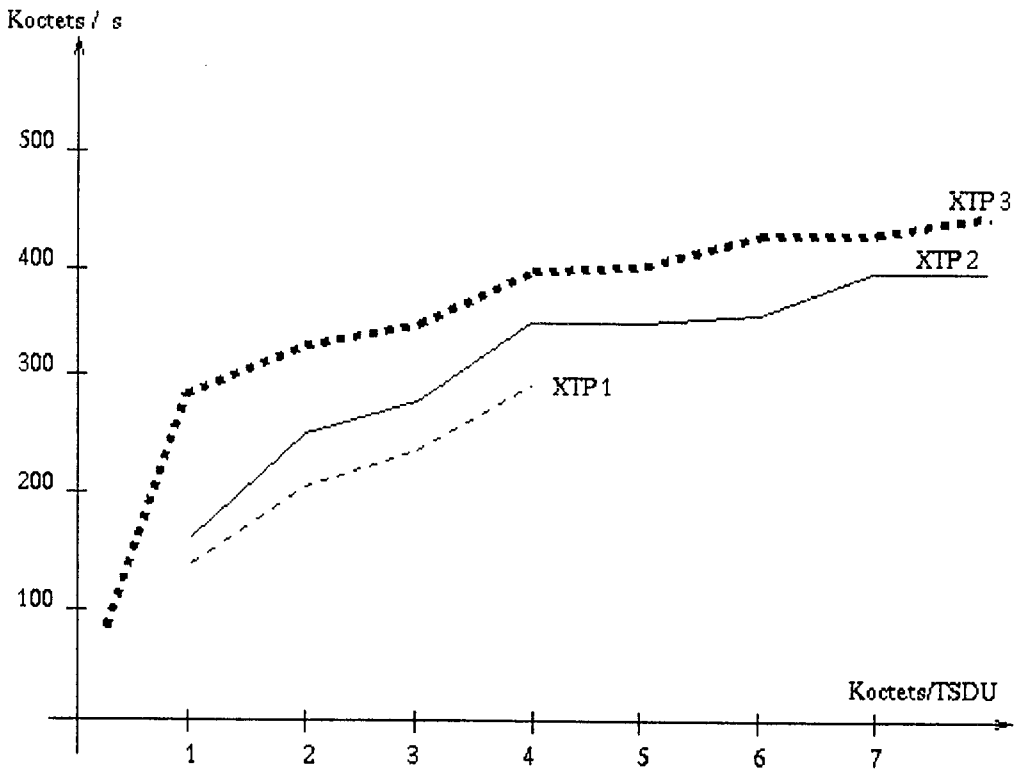
Pour pallier à la difficulté de mesurer avec précision le temps, nous avons choisi de prévoir l'instant t_a à partir du numéro de séquence courant, *seq*, dans le flux d'émission. On demandera un acquittement seulement au moyen des paquets tels que (*seq + A = alloc*), où le paramètre A représente la quantité de données qu'il est possible d'envoyer pendant le rtt.



Le paramètre A dépend du rapport entre le temps total de transmission et celui de propagation sur le réseau.

Le temps de propagation étant négligeable sur une courte distance, le paramètre A est majoré par deux fois la taille d'une TPDU. Si le temps de propagation devient grand, un calcul préliminaire pendant la phase d'ouverture de la connexion permettra d'estimer la valeur du paramètre A en fonction de la première mesure du rtt.

3.4.3.3 Résultats



XTP 1 : sur Ethernet
XTP 2 : suppression des recopies
XTP 3 : strategie d'acquittement

Fig. 3.8 améliorations apportées au débit de XTP

XTP assure un débit plus élevé surtout lorsqu'il s'agit de TSDU de grande taille. Pour une même quantité de données à envoyer, le nombre d'appels système est d'autant plus petit que les TSDU sont grandes. De plus, le nombre d'informations réellement transmises sur le réseau grandit par rapport aux données utiles quand les TSDU diminuent.

Chaque émission (ou réception) nécessitait deux recopies des données. La suppression de ces recopies a apporté un gain notable dans le débit. Il croît avec la taille des TSDU. En effet, pour des TDSU de faible taille, la proportion de temps passé à recopier les tampons est d'une importance moindre par rapport au temps de traitement total à l'intérieur de KRM. Pour la taille maximale de TSDU que nous avons utilisée (huit kOctets), l'homogénéisation du flux de données nous a donc permis de gagner, au maximum, le temps de copie de seize kOctets par TSDU.

De même, la stratégie d'acquittement que nous avons implémentée a sensiblement amélioré les performances du prototype mettant en œuvre les acquittements par défaut.

3.4.3.4 Fonctionnalités restant à implémenter

Nous avons délibérément ignoré certaines parties de KRM, qui n'étaient pas directement indispensables au fonctionnement du protocole dans les conditions où nous l'avons utilisé. Le but poursuivi étant une comparaison des résultats de XTP et de TCP/IP, nous n'avons pas mis en œuvre le mode Multicast ni activé le timer que le protocole XTP utilise seulement pour la surveillance de la connexion.

3.4.4 Extension de la librairie XTI

Parallèlement à l'élaboration de la plateforme, une étude a été menée autour des interfaces hautes du service transport. Cette étude a pour but de donner aux utilisateurs un moyen d'accéder aux nouvelles fonctionnalités du transport, introduites par XTP. Il s'agit pour nous de donner ici l'approche globale des problèmes rencontrés et des solutions retenues.

3.4.4.1 Profil XTI pour XTP

Approche retenue

L'approche retenue a été de tirer pleinement parti des possibilités offertes par le protocole et, dans la mesure du possible, de modifier le moins possible les interfaces existantes, XTI et TPI.

D'une façon générale, l'objectif a été de préserver la compatibilité ascendante avec la version initiale de XTI. Ainsi pour les nouveaux services la solution choisie consiste à créer de nouvelles fonctions, plutôt que modifier les fonctions existantes. Cela présente les avantages suivants :

- plus de clarté (éviter les confusions sémantiques)
- moindre modification des fonctions existantes (maintien de la compatibilité ascendante)
- implémentation rapide (réutilisation du code existant)

D'autre part, les modifications pour XTP susceptibles d'être généralisées à d'autres protocoles ont été introduites au niveau des paramètres d'appel des fonctions XTI. Les autres fonctions, plus spécifiques à XTP ont été introduites au niveau des options.

Lacunes et solutions proposées

Avant d'étudier les différents points qui nécessitent une modification de XTI, rappelons qu'avant tout envoi de données, l'utilisateur doit se lier à une entité de transport; il reçoit alors des informations sur cette entité. Elles décrivent les fonctions de XTI supportées par l'entité de transport et précisent le domaine de certains paramètres.

Nouveaux services

La révision 3.6 de XTP définit six services différents : connexion, datagramme acquitté ou non, bulk, isochrone et transaction (voir glossaire en annexe). Deux problèmes se posent :

– XTI n'est pas prévu pour gérer un protocole disposant de plusieurs services. Jusqu'à présent, à un protocole était associé un service. Ainsi, pour les protocoles de la famille IP, TCP correspond au service connecté et UDP au service datagramme.

– Certains services définis pour XTP ne sont pas prévus dans XTI. Il s'agit en particulier des services datagramme acquitté et transaction.

La solution retenue consiste tout d'abord à regrouper les six services offerts par XTP, en trois services principaux avec des variantes. L'utilisateur choisit le service lorsqu'il se lie à l'entité transport; il choisit l'entité TCP, UDP ou l'une des trois variantes de l'entité XTP : XTP_CO, XTP_CL, XTP_RR.

le service connecté (XTP_CO pour Connection Oriented)

avec les modes : connexion simple
bulk
isochrone

Ce regroupement permet de passer facilement, dans le cadre d'une connexion existante, d'un mode de connexion simple aux modes bulk ou isochrone et vice versa, facilité de fonctionnement sur laquelle insistent les concepteurs de XTP [XTP Definition Revision 3.6]. Le choix du mode se fait par l'intermédiaire des options (paramètres totalement spécifiques au protocole utilisé) données avec certaines fonctions.

le service datagramme (XTP_CL pour ConnectionLess)

avec les modes : datagramme simple
datagramme acquitté

Le choix du mode se fait par un paramètre à chaque appel de la fonction d'envoi de datagramme. Une fonction de réception d'acquittement de datagramme est créée.

le service requête–réponse (XTP_RR pour Request Response)

Ce service est basé sur l'échange fiable d'une requête et d'une réponse qui ne se limite pas à un simple acquittement. La possibilité de demander l'acquittement d'une réponse est aussi proposée. Pour ce nouveau service, la solution correspondant à la création de six nouvelles fonctions a été retenue. Elle présente l'avantage d'une implémentation rapide avec plus de clarté et une moindre modification des fonctions existantes.

Multiples formats d'adresse

XTP supporte de multiples formats d'adresse (ISO, XNS, IEEE,...), ce qui n'existe pas dans les autres protocoles, et n'a donc pas été prévu dans XTI. A chaque fonction qui permet de passer une adresse, il est ajouté un paramètre précisant le type de l'adresse.

Connexion implicite

Le mécanisme de connexion implicite de XTP permet l'envoi de paquets de données sans attendre la confirmation de connexion, offrant ainsi un gain de performances notable pour les connexions de courte durée. Rien n'est prévu dans XTI pour gérer les connexions implicites. Le choix de ce mode se fera par un nouveau paramètre de la fonction de demande de connexion.

Connexion confirmée

XTP propose d'accepter "manuellement" une demande de connexion (comme ISO TP4) ou de ne pas en référer à l'utilisateur (comme TCP). Actuellement il n'est pas prévu qu'un protocole puisse offrir ces deux modes à la fois. Il a donc fallu autoriser le choix de ce mode. Celui–ci se fera par l'intermédiaire d'une option (les options sont des paramètres totalement spécifiques au protocole utilisé) transmise par une fonction particulière de XTI.

Données spéciales

Grâce au champ de données spéciales BTAG, XTP fournit aux applications un moyen d'insérer des informations de contrôle dans les données. Elles sont "vues" mais non interprétées par XTP. Ce mécanisme n'est pas géré par XTI. Un nouveau paramètre a dû être ajouté aux fonctions permettant d'envoyer des données.

Données dans les messages de déconnexion ordonnée

XTP permet d'envoyer des données dans les messages de déconnexion ordonnée, ce qui n'est pas prévu dans XTI. Deux nouveaux paramètres (un pour les données spéciales et un pour les données normales) ont été ajoutés aux fonctions d'envoi et de réception de déconnexion ordonnée.

3.4.4.2 Impact sur TPI

L'étude concerne essentiellement XTI car TPI transmet la plupart des données dans des tampons, sans les formater, de sorte que de nombreuses modifications au niveau de XTI ou de XTP restent transparentes pour TPI. En ce qui concerne les autres modifications, elles sont souvent "décalquées" de celles ajoutées au niveau de XTI.

L'étude effectuée implique la modification des primitives par le rajout de paramètres pour gérer les données spéciales d'une part et les multiples formats d'adresse d'autre part. De nouvelles primitives sont créées afin de traiter le service requête/réponse et le mode datagramme acquitté.

3.4.4.3 Faisabilité

Il ne nous a pas été possible d'implémenter l'extension de la librairie XTI, faute de temps. Cependant une première approximation quantitative du travail à faire a été effectuée. Les modifications concernent 21 fonctions XTI, 5 structures de données et 18 primitives TPI; elles sont assez simples et répétitives. Il faut ajouter 6 fonctions XTI et 7 primitives TPI.

Au total, les modifications et ajouts ont été estimés à 4 hommes/mois environ. En effet, même si la partie codage est relativement limitée et surtout assez répétitive, les phases de test risquent de s'avérer assez longues et délicates.

Ces modifications sont nécessaires pour une exploitation correcte du protocole. Que ce soit pour XTI ou TPI, elles représentent un faible volume de lignes C, même si elles affectent de nombreuses fonctions.

3.5 Conclusion

Au terme de cette réalisation, nous constatons que les apports successifs des différentes étapes nous ont permis d'atteindre le but poursuivi : une plateforme de communications avec les protocoles TCP/IP et XTP. Cependant, même si chaque étape a produit un protocole XTP plus performant que le précédent, nous n'avons pas obtenu les résultats que nous escomptions par rapport à TCP/IP.

L'implémentation utilisée pour TCP/IP est un produit validé, commercialisé et surtout optimisé. Elle est conçue pour s'intégrer totalement dans l'environnement STREAMS dont elle exploite à fond les mécanismes (contrôle de flux, ordonnancement des messages,...) et partage les structures de données (queues, messages,...).

D'autre part, la version de KRM que nous avons intégrée n'est pas la plus récente. En effet, nous avons commencé le projet avec le code de la version 1.6 de KRM et nous espérions pouvoir commuter sur la version suivante (annoncée comme étant plus performante) dès sa parution. Or, elle nous est parvenue alors que nous étions déjà en phase de codage. De plus, après avoir étudié KRM 1.7, nous avons constaté que son code était sensiblement différent de celui que nous utilisions. La commutation d'une version à l'autre allait demander un travail important; à ce stade du projet, cela risquait d'en compromettre gravement l'avancement.

Enfin, pour les résultats obtenus avec XTP, on ne peut pas se limiter à les qualifier en terme de rapidité. Les modifications définies dans l'étude de l'extension de la librairie XTI sont indispensables si l'on veut utiliser les mécanismes performants de XTP. Plus généralement, cette adaptation, en proposant un accès standard à ce nouveau protocole de transport, s'inscrit en fait dans le cadre d'un mouvement vers la standardisation, tendance importante de ces dernières années.

Chapitre 4

ETUDE COMPARATIVE TCP-IP/XTP

4.1 Introduction

Dans un premier temps nous présentons les différentes fonctionnalités que doit comporter la couche transport. Puis nous comparerons les résultats que nous avons obtenus dans nos expérimentations. Enfin nous étendrons cette étude comparative à la définition des services offerts par XTP, comparaison toute théorique puisque nous n'avons pas pu mettre en œuvre toutes les possibilités offertes par ce protocole.

4.2 Fonctionnalités de la couche de transport

Le transport opère directement au-dessus de la couche réseau. Les services fournis par cette couche varient très sensiblement [La Porta]. Dans un cas extrême, le procole de niveau trois peut n'assurer qu'un service datagramme sans protection contre le dé-séquencement des données, alors que dans le cas opposé, il peut offrir un service fiable de circuit virtuel garantissant la livraison des données dans l'ordre.

Les protocoles de transport doivent s'accommoder de ces disparités d'environnement et fournir néanmoins un service de transmission fiable à l'utilisateur.

4.2.1 Gestion de la connexion

L'efficacité des mécanismes d'ouverture et de fermeture de connexion est jugée sur deux critères : la rapidité et la sûreté [C. Diot]. Deux mécanismes sont principalement utilisés :

– la poignée de main qui nécessite l'échange préalable de paquets. Cette solution apporte la sécurité mais peut devenir lourde.

– l'ouverture implicite qui est plus rapide mais nécessite l'emploi de temporisateurs pour plus de sécurité.

Enfin un ensemble d'informations est conservé pour chaque connexion (contexte pour XTP ou TCB pour TCP). Il permet de conserver l'ensemble des conditions de l'ouverture de la connexion ainsi que l'état courant du transfert de données de la connexion. Chaque événement doit être affecté à une connexion avant de pouvoir être traité par le protocole.

4.2.2 Segmentation et ré–assemblage

Procédure classique de la couche transport, son rôle est de découper les TSDU transmises par l'utilisateur du service transport en TPDU de taille plus petite, acceptable par le réseau sans découpages complémentaires. Ainsi la taille de la segmentation est de 128 octets pour un réseau TRANSPAC, de 1500 octets pour un réseau Ethernet, et de 4000 octets pour un réseau FDDI.

Les protocoles supportant la notion de ré–assemblage des TSDU ajoutent une information de contrôle qui marque la dernière TPDU d'une TSDU, afin de la reconstituer en entier avant de la délivrer : pour XTP, il s'agit du bit EOM (end of message) dans le trailer et pour TP4, le bit EOT (end of transmission) dans l'en–tête.

TCP peut négocier une taille maximale de TPDU à l'ouverture de la connexion, mais la fragmentation imposée par le réseau est prise en charge par le protocole IP. C'est pourquoi TCP est couramment implémenté au–dessus de IP.

4.2.3 Contrôle des erreurs

Quatre types d'erreur peuvent affecter les transferts de données entre deux entités de transport opérant au–dessus d'un service réseau sans connexion : modification, perte, duplication et arrivée dans le désordre des données.

Détection de la modification des données

Les protocoles usuels calculent un checksum sur les octets d'un paquet afin de détecter une TPDU altérée. Dans les environnements LAN, il existe un CRC (circular redundancy check) ou une fonction équivalente de vérification de l'intégrité des données au niveau MAC. Néammoins ce checksum ne peut assurer une vérification à travers les éléments intermédiaires d'interconnexion dans le réseau [La Baguette]. Le checksum au niveau transport a pour rôle d'assurer l'intégrité point à point des informations.

La détection d'une TPDU erronée entraine son rejet. Le taux d'erreur (10^{-9} environ) rend ce procédé acceptable. Etant donné la fiabilité des nouveaux environnements (FDDI a un taux d'erreur de 10^{-12}) ainsi que les exigences de certaines applications, il apparaît intéressant de pouvoir éviter le calcul du checksum.

Perte des données, duplications, arrivées dans le désordre

Les phénomènes de perte de données peuvent être la conséquence d'un réseau défectueux ou de l'arrêt sur incident d'un routeur. Pour assurer le transport fiable des données malgré ces aléas, les protocoles utilisent des mécanismes d'accusé de réception et de retransmission ce qui peut provoquer des duplications quand un paquet a été simplement retardé (machine plus lente ou réseau encombré).

Dans certains cas des paquets ayant emprunté des chemins différents, peuvent même arriver dans le désordre. Les protocoles de transport doivent prendre en compte ce phénomène et rétablir le séquencement des données avant de les délivrer.

Méthode de contrôle d'erreurs

Pour déterminer les TPDU transmises avec succès les protocoles numérotent les données. L'entité de transport réceptrice retourne à l'émetteur des informations sur les TPDU reçues correctement. Ces acquittements permettent à l'émetteur de renvoyer éventuellement des données perdues.

Deux types d'acquittement sont associés avec deux stratégies de retransmission : les acquittements positifs associés à une stratégie de retransmission cumulative et les acquittements négatifs permettant une retransmission sélective des données perdues.

De plus les acquittements peuvent être générés systématiquement par le récepteur à chaque réception, ou demandés par l'émetteur (utilisation d'un temporisateur de surveillance).

4.2.4 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. Etant donné les débits élevés ciblés par les nouveaux protocoles, le contrôle de flux est plus que jamais nécessaire même si son fonctionnement reste à définir.

Le contrôle de flux n'est plus seulement chargé de gérer le flux de données traitées par l'entité réceptrice, mais surtout le taux de données transmises sur le réseau à chaque instant par l'émetteur. Son rôle est donc d'éviter la surcharge du système et la pénurie des ressources chez l'une des entités communicantes. Il doit intervenir, au niveau du transport, pour que des congestions ne se produisent pas en réception, durant le traitement des données, ou encore en émission [Gerla].

Deux mécanismes de contrôle de flux sont particulièrement connus: le contrôle de flux par fenêtre d'anticipation (sliding window où l'émetteur fait en sorte de ne pas saturer les capacités de stockage du récepteur) et le contrôle de débit (l'émetteur utilise une temporisation pour limiter ses envois).

Le contrôle de flux par fenêtre, bien qu'efficace est lent puisqu'il oblige le récepteur à émettre des acquittements qui encombrent le réseau. En cas de perte ou de ralentissement, cette technique devient vite pénalisante pour le débit du protocole. Le contrôle de débit semble meilleur et plus fiable puisqu'il évite la transmission des TPDU de contrôle. Cependant, il est délicat à mettre en œuvre : il nécessite une connaissance parfaite du réseau afin d'évaluer précisément la taille et le rythme des flots de données. Il utilise des paramètres qui peuvent éventuellement varier en cours d'activité, selon la configuration du réseau et le taux d'occupation de chaque noeud.

4.2.5 Implémentations des services transport

Les protocoles TCP et XTP réalisent les fonctions de la couche transport, cependant la définition qui en est donnée avec XTP est conditionnée par les exigences multiples d'une implémentation hardware. Sa philosophie de base est de maintenir aussi simples que possible tous les composants du chemin de données ayant un caractère déterminant dans le temps de traitement de l'information.

Cette caractéristique du protocole XTP nous fait espérer de meilleurs résultats en terme de débit, de la part de notre implémentation de XTP par rapport à l'implémentation de TCP.

4.3 Mesures et analyse des résultats obtenus

Nous cherchons à comparer les performances des deux protocoles TCP/IP et XTP en terme de débit (quantité de données envoyées / temps nécessaire à l'envoi). L'application utilisée est le programme BENCH, elle nous permet de mesurer le débit d'émission de données soumises au protocole de transport par TSDU de taille variable.

La configuration matérielle que nous avons utilisée est constituée à base de DPX2 reliés par un réseau local. La carte processeur MTB3 est équipée d'un microprocesseur Motorola 68030, elle est reliée au réseau grâce à une interface Ethernet DETH1, équipée d'un contrôleur de LAN 16 bits.

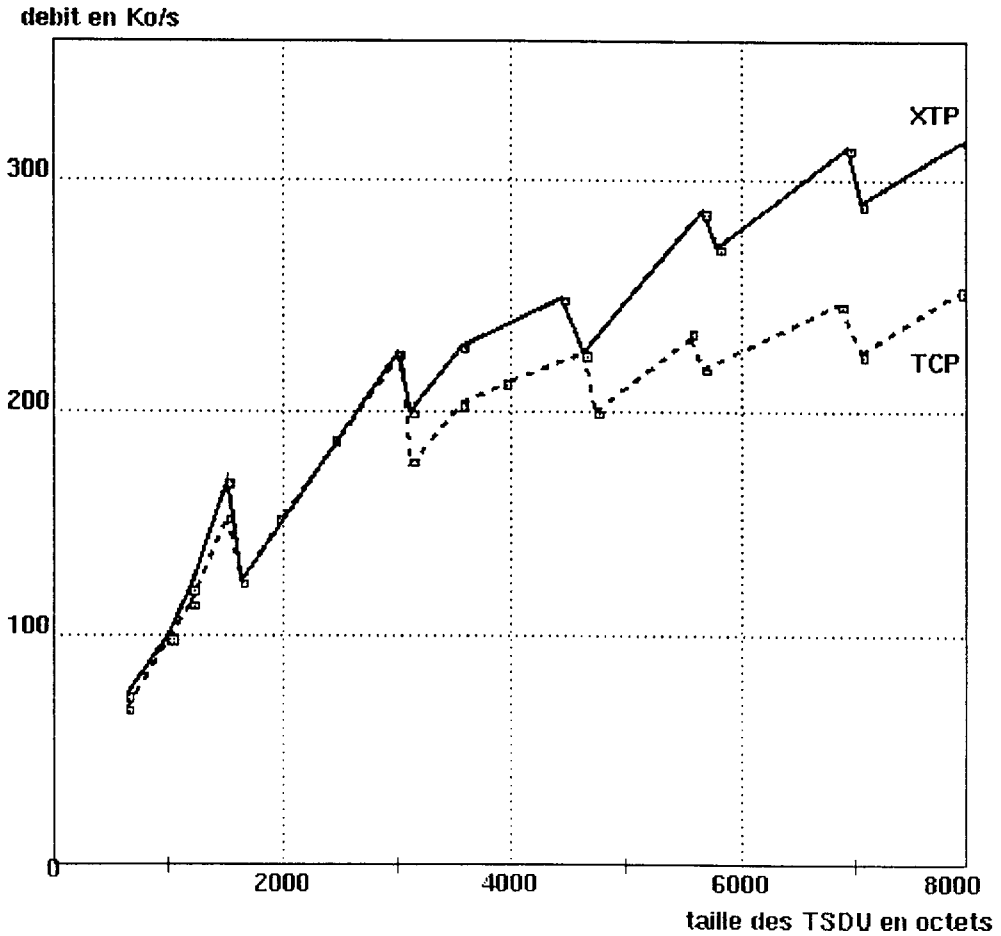


Fig. 4.1 débits relevés

4.3.1 Analyse des résultats

Les mesures que nous avons relevées amènent plusieurs remarques :

Courbes croissantes

Le débit est une fonction croissante. Sur un fragment de courbe la croissance peut avoir deux explications :

– le nombre des appels système (les appels à la librairie XTI, la consultation de l'état de l'interface au protocole, le formatage du message à soumettre au protocole,...); pour une même quantité de données à envoyer, ce nombre est d'autant plus petit que les TSDU sont grandes.

– la proportion des informations réellement transmises sur le réseau par rapport aux données utiles, c'est à dire les données que l'application utilisatrice demande d'envoyer. Pour une même quantité de données, plus la TSDU est petite, plus il y aura de paquets à envoyer, et donc d'informations de contrôle par paquet (encapsulation des données), et plus il y aura de paquets d'acquittement circulant sur le réseau.

Intuitivement, le débit est une fonction croissante (par morceaux) et majorée par le débit maximum du réseau (Ethernet : 10 Mbits/s). Ce débit ne peut donc pas croître indéfiniment.

Points de discontinuité

Ces points de chute de débit sont dus au phénomène de la segmentation liée à la taille maximale d'une trame Ethernet (1500 octets). Par exemple, une TSDU de 1452 octets tient après encapsulation de XTP, dans une seule trame Ethernet. Alors qu'une TSDU de 1453 octets est fragmentée en deux TPDU (une TPDU de 1452 octets et une de 1 octet) qui seront rassemblées à la réception.

Essayons d'estimer le coût global d'une TSDU, c'est à dire le temps de traitement de la TSDU à envoyer, et le temps utilisé pour traiter son acquittement.

– *Pour une TSDU non fragmentée* :

$$T_{traitement} = T_0 + C*Taille_{TSDU}$$

Le temps T_0 ne dépend pas de la taille de TSDU, il inclut le coût des appels système, des fonctions de formatage du paquet (sans le calcul du checksum), ainsi que le coût de l'acquittement.

Le temps $C*Taille_{TSDU}$ est le temps des traitements qui dépendent de la taille tel que le calcul de checksum. De plus, il convient de souligner ici, que même si nous avons supprimé les recopies mémoire au sein de KRM, il subsiste encore des recopies de données entre l'espace utilisateur et l'espace noyau, puis entre l'espace noyau et le medium. Le temps consommé par ces recopies est lui aussi dépendant de la taille des TSDU.

– *Pour une TSDU fragmentée :*

Au temps T_0 s'ajoutera le coût de la fragmentation :

$$T_{traitement} = T_0 + n*T_{fragmentation} + C*Taille_{TSDU}$$

Dans le cas général, le débit peut s'exprimer en première approximation par :

$$Débit(Taille_{TSDU}) = Taille_{TSDU} / (T_0 + n*T_{fragmentation} + C*Taille_{TSDU})$$

n étant le nombre de fragmentations. On obtient donc une fonction croissante par morceau.

4.3.2 Position relative des deux courbes

Pour les TSDU de petite taille, TCP est meilleur que XTP. Etant donné la taille réduite des TSDU, le coût des fonctions indépendantes de cette taille prédomine dans le coût total. Ces fonctions pourraient être plus coûteuses avec notre implémentation de XTP, en raison de l'adaptation des structures de données STREAMS/KRM, adaptations inexistantes dans l'implémentation de TCP/IP que nous avons utilisée.

Pour des TSDU de taille plus importante, le gain apporté par les tâches dépendantes de la taille des données, notamment pour le calcul des checksums compense le handicap dû au portage, et on remarque alors que XTP a en moyenne un débit de 21 % meilleur que TCP.

4.3.3 Mesures sur une configuration matérielle plus performante

Pour les mesures suivantes, nous avons modifié la configuration matérielle.

Les cartes processeur MTB3 ont été remplacées par des cartes MTB4 équipées de microprocesseur Motorola 68040. Il s'agit d'un processeur plus rapide (réduction du nombre de cycles d'horloge par instruction) et plus puissant (intégration de fonctionnalités supplémentaires).

L'interface Ethernet DETH1 a été remplacée par une carte DETH2. Il s'agit d'un matériel plus rapide, prévu pour réaliser des opérations sur des mots de 32 bits (16 bits sur DETH1).

La configuration ainsi obtenue est plus performante que la précédente sur le plan de la rapidité et de la puissance de calcul.

Sur cette configuration, nous constatons que les performances des deux protocoles sont très voisines pour des TSDU supérieures à 1000 octets.

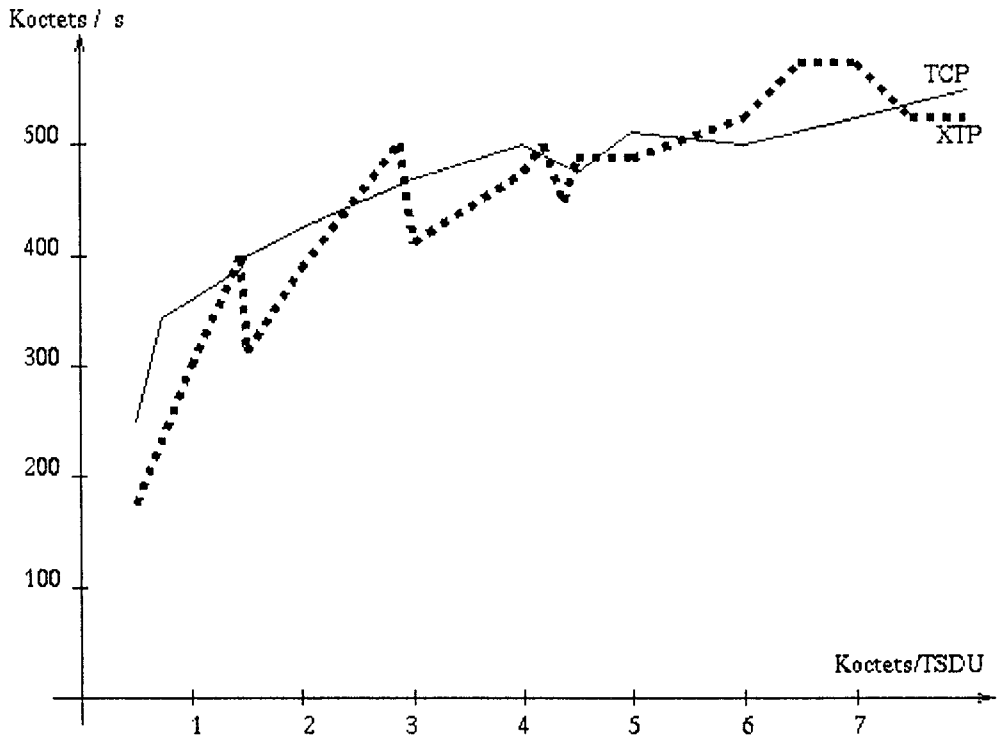


Fig. 4.2 débits comparés sur une configuration matérielle plus puissante

L'augmentation de la puissance de calcul des processeurs permet d'exécuter les algorithmes les plus complexes plus rapidement, ce qui avantage TCP. Cependant, la gestion du réseau ainsi que le réseau lui–même constituent une limite à l'augmentation du débit, ce qui désavantage XTP.

En effet, le temps gagné par XTP pendant le traitement des TSDU peut passer complètement inaperçu à cause des retards ou des attentes dus à la gestion du réseau. Dans ce cas, il est fort probable que le protocole XTP dépasserait TCP sur un réseau plus rapide, tel que FDDI.

4.4 Comparaison des protocoles

Ce paragraphe considére les différentes fonctionnalités de XTP et les compare à celles du protocole TCP. De la même façon, il examine le traitement des erreurs et évalue la performance pour ce qui est de la capacité de traitement et du mode burst. Cette étude porte sur des réseaux locaux et sans routeurs.

4.4.1 Source–destination

Diffusions : unicast et multicast

XTP comme TCP utilise le concept d'un circuit virtuel pour établir une connexion de bout en bout. XTP le réalise plus efficacement au moyen de trois paquets pour l'ouverture du circuit, le transfert d'un paquet de données et la fermeture de la connexion. TCP a besoin de sept paquets pour obtenir le même résultat.

IP peut être utilisé à la place de TCP pour envoyer des datagrammes sans établir de circuit. Avec XTP la connexion à trois paquets est assimilable à un mode datagramme fiable. Les données sont délivrées par le premier paquet aussi rapidement qu'un datagramme IP.

TCP aussi peut transporter des données dans le premier paquet, mais les données ne sont pas délivrées à l'application avant que le circuit soit confirmé par le troisième paquet. Dans TP4, les données utilisateur présentes dans le premier paquet sont limitées à 32 octets.

De plus, le fonctionnement de type commutation de circuit du protocole XTP garantit l'arrivée des paquets dans l'ordre. Alors que dans TCP/IP, le calcul de la route à chaque paquet émis peut entrainer un déséquencement des paquets à la réception.

Le multicast n'est pas disponible sur TCP. Avec IP, il est possible d'envoyer un datagramme à une classe d'adresse. Cependant, le concept de flux de données n'existant pas dans IP, les erreurs comme les discontinuités ne sont pas détectées encore moins corrigées.

XTP assure un mode multicast fiable, avec détection et correction des erreurs (retransmission). Suivant le nombre de stations concernées, le mode multicast de XTP permet d'augmenter les performances par rapport au mode unicast appliqué à l'émission des mêmes données vers les mêmes récepteurs.

Concentration

Aucun protocole ne dispose de mécanismes pour se protéger de l'arrivée simultanée de données, en quantité importante et provenant de plusieurs sources. Le récepteur ne sait pas se protéger contre ce type de saturation. Les protocoles disposent de moyens pour se remettre de cette situation mais pas de moyens préventifs.

XTP dispose du mécanisme de rate control (contrôle d'un débit moyen) pour limiter le trafic sur une route particulière, toutefois cela n'empêche pas un nombre arbitraire d'émetteurs de transmettre simultanément et intensivement vers le même récepteur. Ainsi, on ne sait pas se prémunir contre un afflux de paquets que le récepteur ne peut pas traiter.

4.4.2 Organisation des traitements

Principe FIFO

Pour TCP comme pour XTP les paquets sont traités dans l'ordre où ils arrivent. Dans les deux protocoles il y a des exceptions.

Pour XTP, un paquet avec l'indicateur SREQ positionné nécessite un traitement spécial : l'envoi d'un paquet de contrôle. L'envoi de ce paquet doit être effectué avant tout autre traitement de données sur ce circuit.

Dans TCP, un paquet marqué URGENT est traité différemment des autres. La donnée urgente est délivrée immédiatement à l'application réceptrice avant les données normales déjà en attente sur ce circuit. Cependant les paquets urgents ne constituent pas un flot de données distinct. Leur rôle essentiel est d'informer l'application réceptrice qu'un évènement exceptionnel est apparu; l'exécution de l'application émettrice a été suspendue, par exemple.

Priorité statique

Avec XTP, une application peut ouvrir plusieurs circuits, chacun avec une priorité différente. De cette façon, les paquets urgents peuvent dépasser les paquets non–urgents. Par exemple, le transfert de fichier peut avoir une priorité inférieure à celle d'un message d'une application temps réel, lui–même pouvant être à une priorité inférieure à celle d'un signal de synchronisation d'horloge (utilisation du champ "sort" du header).

TCP ne permet pas cette souplesse : il n'y a aucun moyen pour établir des priorités entre les circuits. De plus, il semble qu'il n'y ait aucune garantie qu'un paquet urgent sur un circuit donné soit traité avant les paquets normaux en attente sur d'autres circuits.

Priorité dynamique

XTP peut changer la priorité des paquets par le mécanisme "time–to–live". Toutefois, il y a un problème de coexistence avec les priorités statiques : est–il plus urgent de traiter un paquet avec une priorité statique de 3 par exemple ou un paquet avec une valeur time–to–live de 50 ms ?

4.4.3 Structure des paquets

En–tête

Alors que TCP a un en–tête de taille variable avec des champs variables en longueur, XTP se fait une règle de conserver l'en–tête constant aussi bien pour sa taille que pour les champs qui le composent (quatre octets). Pour cela, toutes les informations nécessitant un champ variable dans TCP, sont placées dans le segment central du protocole XTP (voir figure 2.12).

Plus particulièrement, XTP place les adresses dans le segment central et seulement dans le premier paquet ouvrant la connexion. Par la suite, le champ KEY du header permet d'identifier la connexion. Cette technique simplifie le traitement sur le plan de l'algorithme et de la quantité d'informations manipulées ainsi que pour le temps d'exécution.

Segment central

XTP dispose de deux types de segment intermédiaire : contrôle et données, alors que pour TCP, tous les octets suivant le header sont des informations pour l'application.

Le segment central de XTP contient quelques champs permettant l'identification d'éventuels trous dans la transmission et une retransmission sélective. Rien de comparable dans TCP qui doit par conséquent utiliser la méthode go–back–n pour ses retransmissions.

XTP utilise un champ de 32 bits pour communiquer la quantité de données acceptables (champ alloc) ce qui permet des transferts de paquet en continu (burst) de quatre giga–octets. Le champ équivalent dans TCP et TP4 est de 16 bits autorisant seulement 64 kilo–octets dans un burst. Après quoi l'émetteur doit attendre un paquet d'acquittement donnant une nouvelle valeur de fenêtre.

De plus, XTP dispose de deux champs (TIME et TECHO) permettant le calcul du RTT, ce que ne permet pas le format du paquet TCP. Pour connaitre le RTT, TCP arme un timer à chaque paquet envoyé et l'arrête quand il en reçoit l'acquittement. La possibilité de duplication de paquets d'acquittement rend cette méthode ambigüe, et oblige ce protocole à mettre en œuvre des algorithmes compliqués. TP4 ne définit pas de méthode pour obtenir le RTT, il le laisse libre au choix de l'implémentation.

Trailer

XTP est le seul à avoir un trailer, essentiellement constitué du checksum (contrôle de parité). La position du checksum est importante car il peut être calculé au fil de l'eau. C'est une caractéristique très intéressante pour les implémentations hardware.

4.4.4 Détection des erreurs

Checksum

Comme TCP, XTP utilise un checksum pour détecter les paquets en défaut. XTP utilise deux checksums : l'un placé à la fin du header et le deuxième dans le trailer.

Les checksums XTP sont prévus pour le traitement parallèle : l'émetteur peut commencer l'émission alors que le calcul du checksum du trailer n'est pas encore terminé. A l'opposé, le récepteur peut commencer à appliquer le protocole alors que les données ne sont pas encore toutes reçues.

Pour TCP, l'emplacement du checksum (dans le header) rend impossible tout traitement "au vol".

Acquittement

XTP et TCP utilisent des numéros de séquence pour acquitter les données reçues. Pour chacun des deux protocoles, la politique d'acquittement n'est pas définie en détail.

TCP spécifie que le receveur doit décider de l'opportunité d'envoyer un acquittement. Il peut être relatif à plusieurs paquets et ne doit pas être retardé. Dans ce cadre, la politique d'acquittement est laissée au choix de l'implémentation, qui doit faire un compromis entre un débit élevé et un faible RTT. Si chaque paquet est acquitté, le RTT diminue mais le débit est affecté par le trafic des acquittements.

XTP spécifie que la décision de l'envoi de l'acquittement revient à l'émetteur : le récepteur n'envoit aucun acquittement de sa propre initiative mais seulement quand il reçoit un SREQ. Le choix du moment opportun pour envoyer un SREQ peut être laissé à l'application émettrice. Quand elle a besoin d'un débit élevé, elle peut demander un ACK seulement à la fin d'un burst. De cette façon XTP peut s'adapter aux besoins des applications utilisatrices. De plus, un récepteur peut envoyer un acquittement négatif dès qu'il détecte un trou dans le flux des données reçues.

Timers

TCP et XTP utilisent un timer pour évaluer le temps au–delà duquel un paquet est considéré comme perdu. Les deux protocoles attendent deux fois le RTT environ avant de ré–émettre.

Dans TCP, un timer est armé à chaque paquet. Une estimation précise du RTT est essentielle pour les performances : un RTT trop élevé provoque une perte de temps avant de re–transmettre, alors qu'une estimation trop courte génère une re–transmission prématurée, ce qui provoque des duplications d'acquittement et des distorsions du RTT.

Dans XTP, le timer n'est utilisé que pour un paquet qui sollicite une réponse. La performance en est nettement moins affectée par un RTT inadapté, d'autant plus qu'il est calculé d'une manière plus précise que dans TCP.

4.4.5 Correction des erreurs

Mécanisme Go–Back–N

XTP et TCP utilisent cette méthode. XTP n'a recours à ce mécanisme que pour une raison de compatibilité avec les premières versions du protocole. Pour TCP, c'est la seule façon de corriger les erreurs.

La quantité de données retransmises dépend de la valeur du RTT, plus précisément de la quantité de données pouvant être émises par une station pendant la durée d'un RTT. Fonction de l'implémentation choisie, ce paramètre peut entrainer la re–transmission d'un volume important de données.

L'apparition d'une erreur dans un burst oblige l'émetteur à ré–émettre l'espace d'un RTT. Pendant la re–transmission, le taux d'occupation de l'émetteur comme du récepteur est équivalente à celle du burst en cours. Cependant, si l'émetteur fonctionne avec une application source temps–réel, il peut avoir besoin d'augmenter son débit. Côté réception, la congestion s'accentue et oblige le protocole à rejeter encore plus de paquets, provoquant l'augmentation des re–transmissions. Si le RTT est important, cela peut conduire à une réaction en chaine de retransmissions pouvant aller jusqu'à un arrêt complet du trafic sur le réseau.

Il y a une grande variété dans les implémentations de TCP notamment sur des points qui influencent le RTT, donc la possibilité d'une réaction en chaîne de retransmissions.

Retransmission sélective

La retransmission sélective n'est possible que sur XTP. Un récepteur TCP n'a aucun moyen pour dire à l'émetteur qu'il a reçu un burst complet à l'exception de quelques paquets.

Par la retransmission sélective, XTP peut envoyer un burst complet de paquets, sans se préoccuper de sa longueur, puis demander l'état des récepteurs et ne retransmettre que les paquets manquants. La réaction en chaine des retransmissions ne peut donc pas apparaitre.

Possibilité d'ignorer les erreurs

XTP peut invalider le calcul du checksum des données (NOCHECK) et même interdire toute retransmission (NOERR). TCP ne permet pas ces options; pour TCP/IP la détection des erreurs peut être évitée par l'utilisation directe de IP, bien que cela détruise le concept de flux de données.

La possibilité d'invalider la correction des erreurs peut être très utile pour des applications telles que la vidéo. Il peut être acceptable d'avoir une partie des données corrompue pourvu que l'ensemble du flot des données soit délivré.

4.4.6 Prévention des erreurs

Contrôle de flux

TCP et XTP utilisent une fenêtre de taille variable pour contrôler le flux des paquets et éviter ainsi au receveur d'être submergé par manque d'espace mémoire.

Dans TCP la taille de la fenêtre est mise à jour sur l'initiative du récepteur. La décision est prise suivant un algorithme conçu de telle façon que TCP puisse assurer un flux continu des données et éviter les situations "stop–start". Ainsi, un acquittement avec une nouvelle fenêtre est envoyé quand le récepteur estime que l'émetteur arrive à la fin de sa fenêtre précédente. Suivant l'espace mémoire disponible, cela conduit à générer un RTT important ou de faible valeur.

Dans XTP, la taille de la fenêtre est mise à jour sur l'initiative de l'émetteur, à moins que le mode RESERVATION soit choisi. La décision de demander une mise à jour de la fenêtre dépend de l'implémentation. Elle peut être basée sur le RTT : l'émetteur demande une nouvelle valeur de façon à l'avoir juste avant d'avoir épuisé la fenêtre en cours. Dans le mode réservation, l'espace mémoire n'est pas dans l'espace du protocole mais dans l'espace de travail de l'application. La décision des mises à jour de fenêtre revient donc à l'application.

Contrôle de débit

Le contrôle de flot d'XTP est insuffisant pour éviter toute perte ou rejet des paquets reçus : l'espace mémoire peut être suffisant pour accueillir le paquet et le processeur d'une puissance de traitement insuffisante pour s'accomoder du débit auquel les paquets arrivent.

Dans XTP, les deux paramètres "rate" et "burst" permettent de contrôler le débit des données en entrée sur un circuit donné. Le récepteur peut augmenter ou diminuer le débit en fonction des conditions du trafic et des ressources disponibles.

Les deux paramètres peuvent être négociés pendant la phase d'ouverture de connexion, auquel cas ils ne peuvent plus être modifiés pendant toute la durée de cette connexion. Les stations peuvent conserver une trace de cette largeur de bande et se protéger d'un afflux soudain du trafic.

Dans TCP/IP, il n'y a pas de mécanisme pour éviter d'être submergé par le trafic entrant. Cependant le récepteur peut utiliser la requête QUENCH de ICMP quand il a épuisé son espace mémoire. Les autres stations peuvent alors utiliser l'algorithme du "slow startup". Cela évite les retransmissions effrénées qui augmentent encore la congestion.

Contrôle de transmission

Le contrôle de flux ainsi que le contrôle de débit peuvent indépendamment l'un de l'autre interdire les émissions. Pour XTP, "allocation" et "credit" doivent être positifs pour qu'une transmission puisse avoir lieu. La valeur allocation peut devenir nulle avant celle du credit, dans ce cas le débit de la transmission devient inférieur à celui qui a été négocié. Ainsi il peut arriver que le récepteur XTP ne soit pas capable de supporter le débit négocié.

4.4.7 Performance

Capacité de traitement

La transmission sur FDDI utilisée par les stations actuelles est étroitement liée au processeur, autrement dit une station de travail met plus de temps pour préparer un paquet que l'adaptateur n'en met pour l'envoyer. Le débit est limité par la vitesse à laquelle une station peut traiter un paquet. TCP/IP utilise deux protocoles ce qui implique davantage de manipulations que XTP.

Le débit maximum est aussi influencé par les algorithmes d'allocation mémoire et de contrôle de flux. Le manque d'espace mémoire en réception peut bloquer l'émission. D'autre part l'émetteur peut avoir épuisé son espace mémoire et attendre un acquittement pour libérer l'espace occupé par des paquets acquittés.

De plus le traitement des acquittements peut réduire le débit. Dans TCP, le traitement d'un acquittement peut être assez long car il nécessite l'arrêt de tous les timers pour tous les paquets concernés par l'acquittement. Dans XTP, le traitement d'un acquittement ne concerne qu'un timer. Si l'émetteur demande des acquittements espacés, le temps de traitement de l'acquittement est négligeable en comparaison de TCP.

Rafales de paquets

Une source temps réel, telle que la caméra vidéo ou un microphone, peut générer après numérisation des données en rafales. L'application traitant ces données doit pouvoir les passer au protocole, rafale par rafale et laisser la mise en mémoire à la charge du protocole. Si les tampons du protocole pour la connexion ne sont pas assez importants pour absorber les rafales, alors l'application devra les prendre en charge. Mais une couche supplémentaire de tampons monopolise davantage de ressources processeur, multiplie les manipulations de données et réduit la vitesse d'émission.

Dans XTP, l'émetteur peut négocier avec le récepteur la taille des bursts ainsi que la vitesse de transmission nécessaire. Le récepteur peut adapter l'espace mémoire en conséquence. Par une utilisation judicieuse du burst, une bonne implémentation d'XTP peut très bien fonctionner avec de multiples sources temps–réel.

Dans TCP, il n'y a aucun moyen de connaitre le type de données échangées sur un circuit. Une source temps réel peut mettre la transmission en défaut par manque de buffer à l'une des extrémités pendant un burst. Si l'émetteur peut rattraper l'application pendant un trou entre deux bursts, la seule conséquence est que la transmission de quelques bursts aura été prolongée. Cependant cela implique que les bursts soient suffisamment espacés pour qu'aucune donnée ne soit perdue.

4.4.8 Améliorations

M. Germuska et G. Morgan [Transfer juillet/août 1992] proposent des améliorations à la définition du protocole pour trois grands points.

Contrôle de débit

Les paramètres rate et burst sont utilisés pour calculer le temps pris par un burst. Cependant l'émetteur peut utiliser ce temps à sa guise : envoyer rapidement tous les paquets d'un burst et attendre ou alors, espacer l'envoi de chaque paquet et attendre un bref instant entre chacun des envois. De cette façon, la protection du récepteur n'est pas optimum.

De plus, l'émetteur peut être beaucoup plus puissant que le récepteur. Dans le pire des cas, un émetteur très puissant pourrait émettre des rafales très rapides vers un récepteur plus faible, ne pouvant pas faire face à une telle charge.

Même dans un réseau où toutes les stations sont de puissance égale, un récepteur peut être mis en difficulté de la même manière par plusieurs émetteurs émettant simultanément vers le même récepteur. Le récepteur pourrait ne pas être en mesure de faire face à cette situation même si les différents émetteurs respectent les valeurs négociées pour rate et burst.

Il serait intéressant d'éviter la saturation de la réception en donnant la possibilité au récepteur XTP d'imposer une plage minimum de silence entre les paquets d'un burst. Sous ces conditions, la situation serait améliorée aussi dans le cas d'émissions multiples vers un même récepteur.

L'amélioration suggérée est de faire porter le contrôle de la transmission sur des silences entre les paquets et des trous entre les bursts, à la place du contrôle par la valeur rate. De plus la longueur du paquet ainsi que le nombre de paquets par burst devraient être utilisés à la place du paramètre burst.

Allocation des tampons

A moins d'avoir opté pour une allocation de tampon très subtile, XTP ne peut pas garantir totalement la réception des données à la vitesse moyenne qu'il a négociée.

On pourrait virtuellement garantir un espace mémoire adapté si le récepteur était contraint à réserver pour chaque circuit, un buffer circulaire suffisamment grand pour recevoir au moins deux bursts. Le récepteur rejetterait alors toute requête nécessitant un espace supérieur à la taille autorisée.

Evidemment, il existe certainement des solutions plus économiques, mais ce système simple pourrait être proposé comme une option dans XTP. Dans tous les cas, une allocation mémoire adaptée est essentielle pour de hautes performances, et ce point nécessitera des études plus approfondies.

Interface de programmation avec les applications (API)

L'avantage majeur d'XTP réside peut–être dans le fait qu'il permet de configurer chaque circuit pour l'adapter au trafic, allouer l'espace mémoire approprié et estimer le temps processeur nécessaire. Ainsi les meilleures performances peuvent être atteintes avec un espace mémoire et une puissance de processeur bien adaptés.

Cependant, les stations ne peuvent pas tirer avantage de cela, à moins que les applications puissent indiquer au protocole quel type de trafic attendre sur un circuit donné. Or les applications seules ont le moyen de connaitre cette information. Il serait donc intéressant que le protocole prévoit le traitement d'un appel API permettant à une application demandant une connexion, d'en spécifier dans le détail le trafic (longueur des paquets, nombre de paquets par burst, silence entre les paquets et espacement des bursts).

4.5 Conclusion

Les performances de XTP sont–elles suffisamment convaincantes pour justifier sa substitution à TCP/IP ?

Si on se contente de la comparaison des résultats obtenus avec ces deux protocoles, la tentation est forte de dire que XTP n'est pas aussi performant que le prétendent les membres du consortium chargé de sa promotion. Cependant les mesures effectuées ne portent que sur un seul critère d'évaluation du protocole, le débit. Tous les autres points forts de XTP, comme la diffusion fiable, les différents niveaux de priorité, n'ont pas été évalués dans notre implémentation.

Par sa définition, XTP dispose d'atouts pour sur–classer TCP/IP et TP4. La raison essentielle en est qu'il fournit des facilités pour s'adapter au trafic sur le réseau. Or la connaissance du trafic est fondamentale pour de bonnes performances. Ceci a été démontré par des protocoles spécifiques tels que NETBLT, destiné au transfert de masse, et VMTP, conçu pour du traitement par messages.

Cependant, au niveau de sa définition, le protocole XTP laisse quelques points fondamentaux optionnels, au choix de l'implémentation. Il peut arriver qu'une implémentation donnée ne prenne pas en compte ces caractéristiques, et ne s'adapte pas au trafic auquel cas il n'y aura aucune amélioration par rapport à TCP/IP. Les suggestions faites par M. Germuska et G. Morgan [Transfer juillet/août 1992] vont dans le sens de cette amélioration.

Chapitre 5

CONCLUSIONS

Après l'étude du protocole XTP et de son implémentation sur DPX2, il est difficile d'apporter une conclusion immédiate et définitive sur les mérites d'un tel protocole vis à vis des autres protocoles existants. La question qui se pose est plutôt de savoir si ce protocole remplit les objectifs pour lesquels il a été conçu et quelles sont les perspectives ouvertes par la plateforme de communication que nous avons constituée.

5.1 Motivations

La nature fondamentale de l'informatique change : les applications ne sont plus confinées sur un processeur individuel. Elles sont maintenant réparties sur plusieurs calculateurs et le réseau devient un facteur critique dans l'architecture de tels ensembles.

Les protocoles existants ont été développés à une époque où les réseaux étaient relativement lents, la largeur de bande pour les communications était une ressource rare et coûteuse. L'évolution des technologies et l'apparition de nouveaux besoins ont fait apparaitre la nécessité de modifications importantes dans les protocoles de transport, ce qui est à l'origine d'un débat encore ouvert à ce jour : faut-il modifier et étendre les protocoles existants ou bien est-il préférable de développer de nouveaux protocoles avec des mécanismes et fonctionnalités spécifiques pour les nouveaux environnements et les applications ?

Ainsi XTP a été conçu pour une combinaison de services plus riches : ouverture de connexion rapide, retransmission et acquittements sélectifs, priorité des messages, acheminement de données spéciales (out–of–band data), formats d'adressage multiples, datagrammes fiables, mode multicast,...

5.2 Services et performances

Services

XTP peut fournir une gamme complète de services nécessaires aux systèmes distribués, ainsi que d'autres mécanismes très intéressants lui permettant d'offrir à ces utilisateurs une flexibilité importante. En résumé, ce protocole privilégie l'implémentation de mécanismes par rapport à une politique particulière pré–établie, réalisant le fait que l'utilisateur seul dispose d'informations suffisantes pour optimiser réellement les paramètres d'un échange de données. Cette particularité, originale par rapport aux autres protocoles permet à XTP de répondre dynamiquement à l'utilisateur pour une large gamme d'applications.

Performances

Bien sûr, l'aspect performances est essentiel dans les objectifs poursuivis par tout développement. Comme nous l'avons constaté au cours de l'étude comparative avec TCP/IP, la version logicielle de XTP que nous avons réalisée a atteint le même niveau de performances que TCP/IP. Un résultat analogue a été obtenu avec des travaux semblables menés sur le réseau BISDN du projet Berkom ainsi que sur Ethernet, dans un environnement SunOS dérivé de Unix BSD4.3 [TRANSFER septembre/octobre 1992]. Si cela parait insuffisant, il est à noter que par son ancienneté et l'intérêt qu'il suscite, TCP/IP a bénéficié de multiples optimisations, alors que XTP, protocole récent, n'est pas encore stabilisé.

5.3 Faut–il changer de protocole de transport ?

Deux alternatives

Nous avons vu précédemment que les applications nouvelles ont des exigences accrues en matière de fonctionnalités et de performances pour les sous–systèmes de communication. Si l'on considère l'utilisation largement répandue de TCP/IP et des suites de protocoles OSI, il est difficile d'arrêter une décision entre deux alternatives : est–il préférable d'étendre les fonctionnalités de ces protocoles déjà bien installés pour s'adapter aux nouvelles applications ou bien doit–on introduire un ensemble nouveau de protocoles, tels que XTP, explicitement conçus pour faire face aux nouvelles exigences des applications et aux modifications de l'environnement ?

Justifications pour introduire XTP

A la suite de cette étude, il apparait souhaitable d'abandonner les protocoles actuels au bénéfice des nouveaux tels que XTP lorsque les conditions suivantes peuvent être satisfaites :

– Dans une configuration avec un réseau à haut débit et une interface capable de fournir un accès rapide au réseau, il serait difficile d'atteindre une vitesse supérieure avec les mécanismes des protocoles actuels. Alors qu'un protocole allégé comme XTP, peut être implémenté dans une version matérielle capable d'exploiter la vitesse des nouveaux supports de transmission, pour les besoins d'une application temps réel par exemple.

– Quand le besoin de nouveaux dispositifs se fait sentir, notamment la transmission multimedia ou le mode multicast, il serait absolument indispensable d'ajouter ces caractéristiques aux protocoles actuels. En raison de leur définition même de telles extensions dégraderaient considérablement leurs performances. L'utilisation de XTP serait alors plus aisée, efficace et cohérente.

Enfin, les extensions à apporter aux protocoles existants seraient tellement importantes que l'interopérabilité avec les versions précédentes de ces protocoles serait très difficile voire impossible. Le passage à cette nouvelle version serait alors aussi laborieuse et délicate que le passage à un nouveau protocole.

Les considérations précédentes ne nous permettent donc pas de conclure que XTP peut d'ores et déjà être utilisé en lieu et place des protocoles actuels. Les performances de ce protocole ne sont pas suffisamment convaincantes pour en justifier une utilisation universelle. Cependant les résultats obtenus sont encourageants et font espérer des améliorations avec les prochaines versions.

5.4 Perspectives

Enfin, il est important de souligner ici que l'évolution des technologies, des infrastructures et des standards dans le monde des communications conduit à définir de nouvelles caractéristiques pour les futurs sous–systèmes de communication.

D'une part, ils devront supporter de multiples protocoles et d'autre part, ils devront isoler au maximum la partie transmission (couches 1à 4) de la partie traitement de l'information (couches 5 à 7) de façon à maintenir l'interopérabilité entre les applications, quelque soit les protocoles de transport sélectés. XTI et TPI constituent des exemples de ces interfaces destinées à standardiser l'accès aux différents protocoles de transport, alors que XTP vient enrichir de ses nouveaux services les différents protocoles de transport déjà disponibles.

Il semble bien adapté pour les types d'applications émergeant dans les années 1990 : communication entre calculateurs et dispositifs de visualisation à haute résolution, systèmes de contrôle temps réel en milieu industriel ou sur du matériel embarqué (bateaux, avions, véhicules spaciaux), traitements pour image médicale à distance, systèmes transactionnels de type client/serveur, et téléconférences multimedia.

Ainsi la plateforme que nous avons constituée s'inscrit dans ce mouvement, né de l'évolution des technologies et des besoins et visant à apporter sur le marché un environnement cohérent et compétitif pour une informatique répartie et intensive à l'échelle mondiale.

BIBLIOGRAPHIE

BULL SA
Streams Primer – Streams Programmer's Guide – B.O.S.

M. Miller
Internet Working – A guide to network communication

Douglas E. COMER, David STEVENS
Internetworking with TCP/IP

Robert M.Sanders, Alfred C.Weaver
The Xpress Transfer Protocol (XTP) – A Tutorial
Computer Networks Laboratory – University of Virginia.

Protocol Engines Incorporated
XTP Protocol Definition – Revision 3.5
PEI – California

Protocol Engines Incorporated
KRM Distribution Guide for XTP 3.5 – KRM Release 1.6
PEI – California.

Vicki H. Rosenthal
The Transport Provider Interface (TPI) Specification for System V
AT&T Bell Laboratories.

R. J. LEWIS, J.K. FELLIN, D.J. OLANDER.
A STREAMS-based Data Link Provider Interface
AT&T Bell Laboratories.

A. DANTHINE, Y. LABAGUETTE
Comparison of Transport Protocol Mechanisms
OSI 95 High Performance OSI Protocols with Multimedia Support on HSLAN's and BISDN.
Université de Liège, Faculté de Sciences Appliquées. Avril 91.

V. ROCA
XTP sur TRANSPUTERS : Implémentation et évaluation.
Rapport de projet de fin d'études.
Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble –1991

XTP FORUM
TRANSFER – XTP Information
Volume 5 – number 4 – juillet/août 1992
Volume 5 – number 5 – septembre/octobre 1992

M. HABERT
Emerging Communications Technologies
Technical Update – Octobre 1992

A. NOVEL, O. ROMAGNY
Adaptation d'Interfaces de transport pour le protocole de réseaux à haut débit XTP
Rapport de projet de fin d'études.
Institut National de Sciences Appliquées de Lyon – (INSA juin 1992)

K. BELGAIED, M. HENDAZ
Portage de XTP sous STREAMS
Rapport de projet de fin d'études.
Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble – (ENSIMAG 1992)

K. BELGAIED

Portage de XTP sous STREAMS et comparaison avec TCP/IP

DEA D'informatique.

Université Joseph Fourier – Informatique et Mathématiques Appliquées de Grenoble – (IMAG 1992)

M. HENDAZ

Portage de XTP sous STREAMS et mise au point du mode Transactionnel

DEA D'informatique.

Université Joseph Fourier – Informatique et Mathématiques Appliquées de Grenoble – (IMAG 1992)

C. Diot

Architectures Hautes Performances pour l'Implantation des Protocoles de Communication de Niveau Transport.

PhD thèse de l'Institut National Polytechnique de Grenoble – janvier 1991

M. Gerla – L. Kleinrock

Flow Control : a comparative survey

IEEE Trans. Comm., Vol. 28, No 4, Avril 1980

BULL SA

Guide & Reference Manual – Bull DPX/2 – XTI

Bull Open Software – février 1992 –

GLOSSAIRE

acquittement	Négatif : paquet envoyé par un récepteur indiquant à l'émetteur une rupture dans la séquence des messages reçus (demande de retransmission). Positif : paquet envoyé par un récepteur indiquant à l'émetteur la bonne réception en séquence des messages jusqu'au rang spécifié (accusé de réception).
ARP	Address Resolution Protocol
ARPANET	Advanced Researched Projects Agency Network
ATM	Asynchronous Transfer Mode
BISDN	Broadband Integrated Service Digital Network
bulk	terme imprécis qui recouvre aussi bien un volume important de données (images, bases de données,...) que la nécessité d'une grande largeur de bande
contrôle d'erreur	mécanisme permettant de s'assurer que les messages envoyés sont correctement arrivés à destination, malgré d'éventuelles erreurs de transmission (perte de paquets, saturation de tampons,...)
contrôle de flux	mécanisme mis en œuvre pour contraindre un émetteur à ne pas envoyer plus de paquets que le ou les récepteurs peuvent en accepter.

datagramme	unité de données transférées entre 2 utilisateurs du mode sans connexion.
DCM	Distributed Computing Model
DLPI	Data Link Provider Interface
FDDI	Fiber Distributed Data Interface
fenêtre	– d'émission : permet à l'émetteur de connaitre les bornes de la séquence des trames qu'il peut envoyer. – de réception : définit la liste des numéros de séquence des trames acceptables par le récepteur.
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ISDN	Integrated Service Digital Network
Isochrone	trafic de données continu telque celui généré par une source audio ou vidéo.
KRM	Kernel Reference Model
LAN	Local Area Network (réseaux locaux)
LLC – MAC	Logical Link Control et Media Access Control. Dans les réseaux locaux, la couche liaison de données (couche 2) est divisée en deux sous–couches : LLC pour la partie supérieure et MAC pour la partie inférieure.
MAN	Metropolitan Area Network
multicast	mode diffusion où un message peut être envoyé à plusieurs postes distants.
medium	lien matériel utilisé pour la propagation ou la transmission de signaux
PEI	Protocol Engine Incorporated
retransmission Go–Back–N	Technique de retransmission qui consiste à renvoyer tous les paquets depuis celui sur lequel porte l'erreur
RPC	Remote Procedure Call
SDH	Synchronous Digital Hierarchy
sockets	Spécification d'interface Berkeley BSD 4.3 pour la couche transport

STREAMS	Mécanisme noyau permettant le développement de services répartis et de drivers pour la communication de données.
TCP	Transmission Control Protocol
TPI	Transport Provider Interface
TPDU	Transport Protocol Data Unit : élément d'information circulant sur le réseau.
TSAP	Transport Service Access Point
TSDU	Transport Service Data Unit : élément d'information utilisé entre l'application et le protocole de transport.
UDP	User Datagram Protocol
VLSI	Very Large Scale Integrated
VMTP	Versatile Message Transaction Protocol
WAN	Wide Area Network (réseaux étendus)
XTI	X/Open Transport Interface
XTP	Xpress Transfer Protocol

PRESENTATION DE KRM

Architecture générale

De même que pour tout protocole de communication, deux flux de données fondamentaux existent :

– le flux entrant des données reçues depuis le réseau et destinées à l'application

– le flux sortant des données soumises par l'utilisateur et destinées à être transmises sur le réseau.

XTP incluant des fonctionnalités de routage (c'est un protocole de transfert), il peut donc être amené à retransmettre un paquet reçu, s'il ne lui est pas destiné. Cela constitue donc un troisième flux de données.

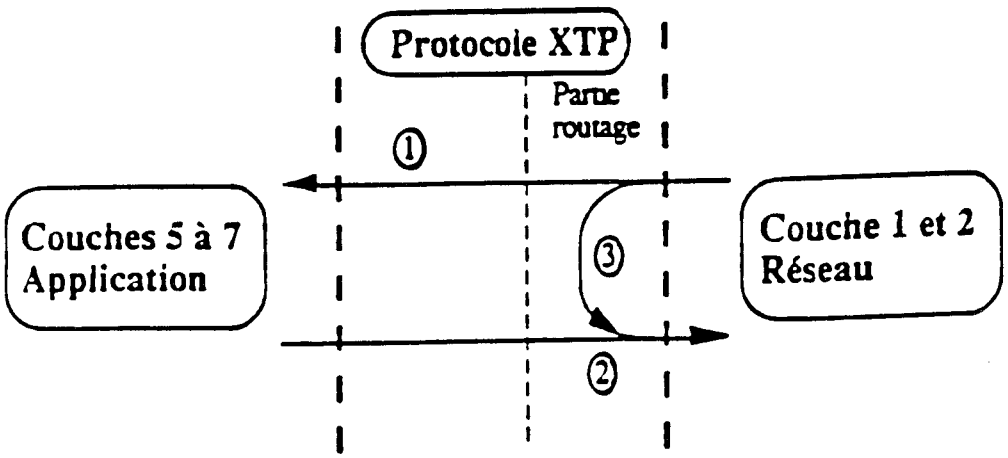


Fig. .1 représentation des 3 flux de données

Ces considérations permettent de présenter une première modélisation de KRM qui fait apparaitre d'une part le fonctionnement du protocole partagé entre le traitement des flux entrant et sortant, et d'autre part une première idée de la structure d'implémentation :

- les tâches relatives au flux des données entrantes
- celles qui sont relatives aux flux des données sortantes
- les informations nécessaires au fonctionnement du protocole et qui sont partagées par les deux types de tâche.

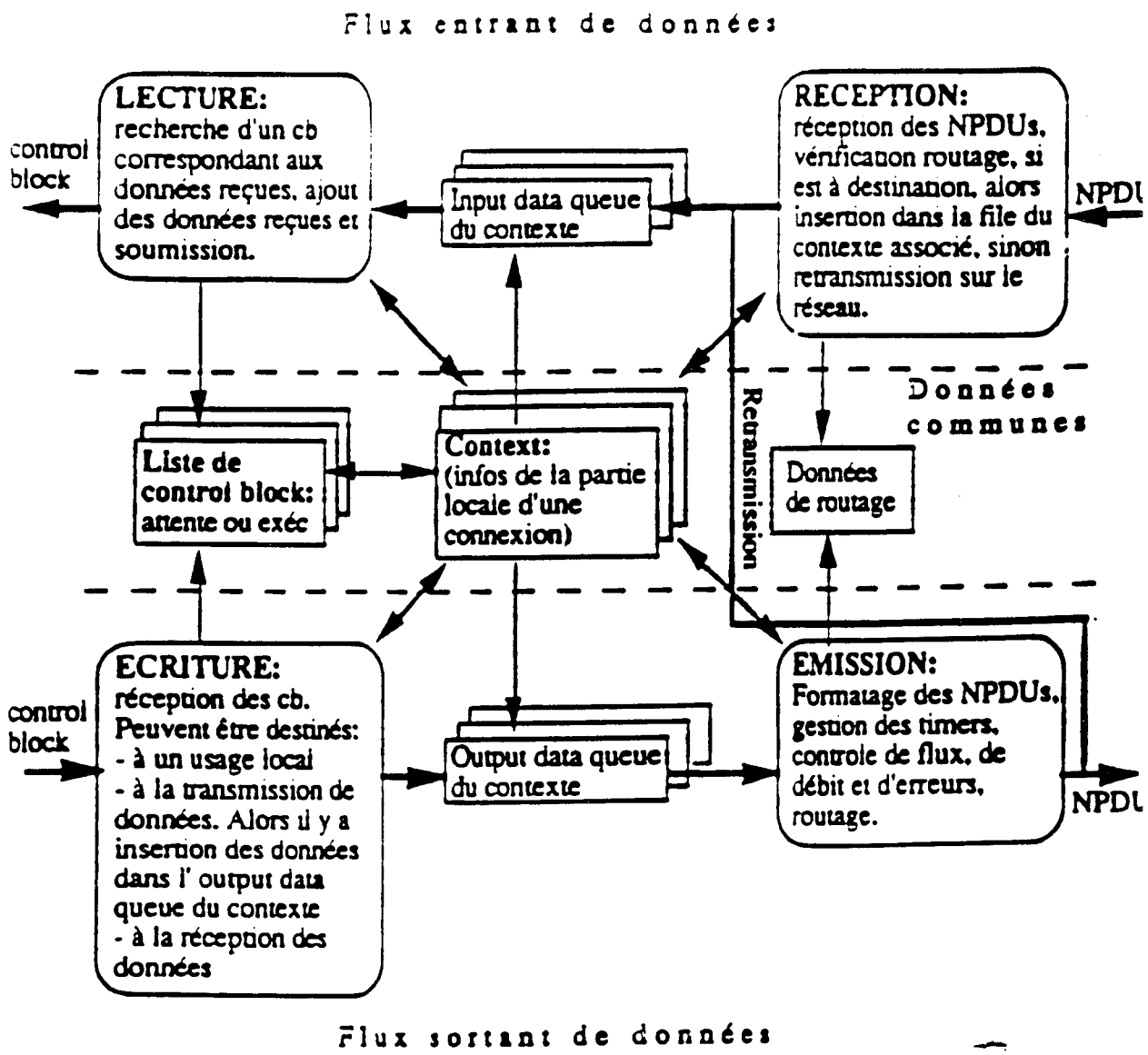


Fig. .2 Architecture générale et flux de données

Remarques

Les quatre éléments réception, émission, lecture et écriture constituent une décomposition des tâches devant être effectuées par le protocole. La décomposition en processus, que ce soit celle de KRM ou bien celle de notre adaptation diffère sous bien des aspects.

Structures de données :

– control block : dans la version initiale la structure cb assure l'interfaçage entre le protocole et l'utilisateur. Le portage sous STREAMS a permis de la supprimer.

– context : c'est la structure centrale de KRM. On y maintient à jour toutes les informations propres au contexte (partie locale de la connexion). Tout le fonctionnement de KRM s'articule autour de cette structure de donnée. Elle contient des données de gestion du protocole, les informations véhiculées par les paquets de contrôle, les données nécessaires à la gestion des flux entrant et sortant, enfin les informations nécessaires à la gestion du mode multicast. Il existe autant de structures context que de connexions où la station est impliquée.

AUTOMATE D'ETATS FINIS DE TCP

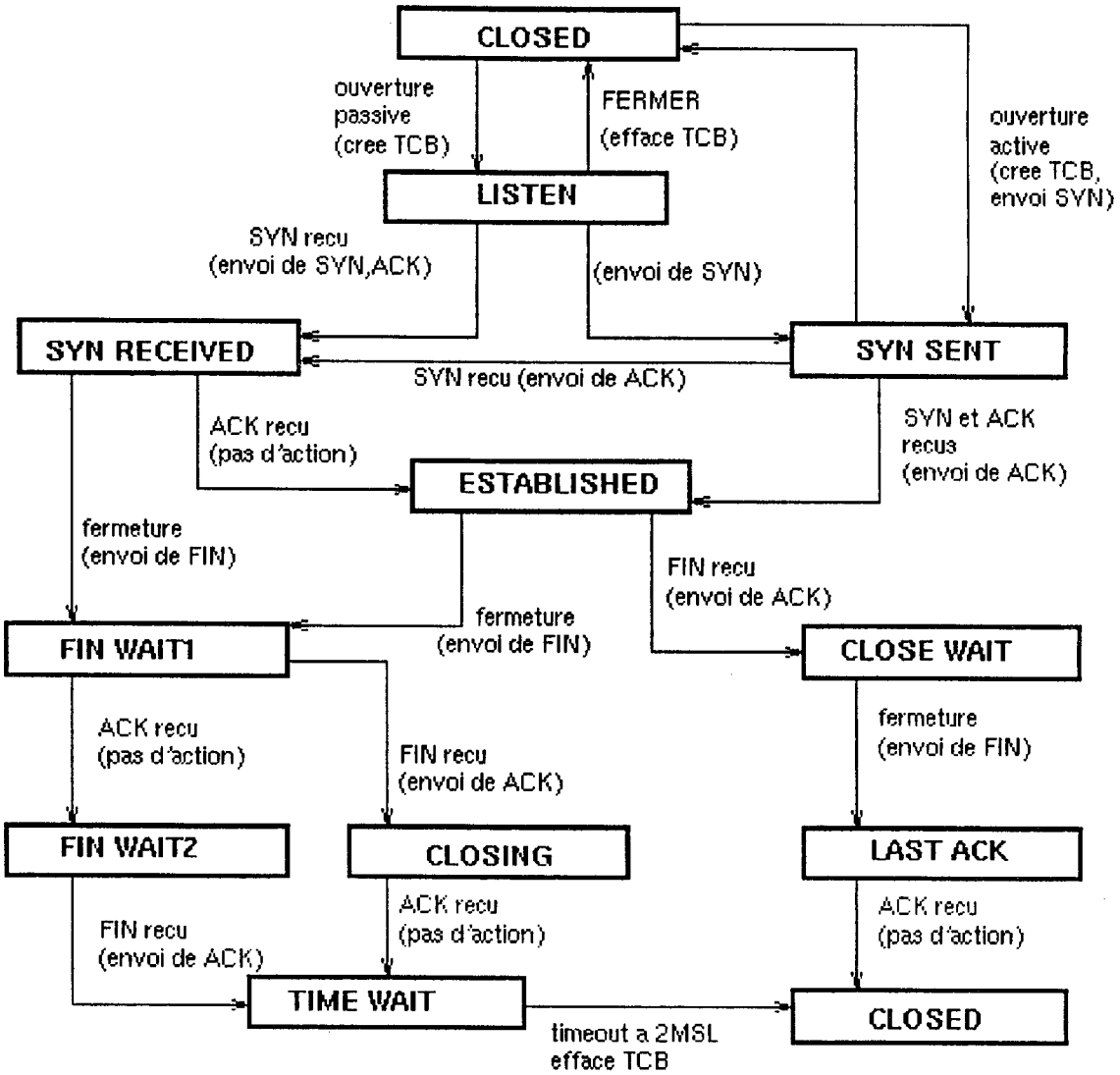


Fig. 1 diagramme d'états TCP

AUTOMATE D'ETATS FINIS DE TCP

Etats pendant l'ouverture d'une connexion:

CLOSED : connexion fermée

LISTEN : état des serveurs en attente de demandes de connexions de clients.

SYN SENT : connexion active, a envoyé SYN

SYN RECVD : a envoyé et reçu SYN

Etat d'une connexion établie : **ESTABLISHED**

Etats pendant la fermeture à la demande du client:

CLOSE WAIT : a reçu FIN, attend close

LAST ACK : a reçu FIN et close, attend FIN ACK

CLOSED : connexion fermée

Etats pendant la fermeture à la demande du serveur :

FIN WAIT-1 : envoit FIN

CLOSING : attend FIN ACK

FIN WAIT-2 : a reçu FIN ACK, attend FIN

TIME WAIT : 2MSL d'attente avant la fermeture

CLOSED : connexion fermée

Etude et réalisation d'une plateforme de communications intégrant le protocole pour réseau à haut débit XTP

Michèle Boutet

Mémoire d'ingénieur C.N.A.M. Grenoble 1993

Ce projet s'inscrit dans le cadre des réseaux hauts débits. Le but poursuivi est de réaliser une plateforme intégrant à la fois un protocole d'ancienne génération, TCP, et un nouveau protocole XTP ou eXpress Transport Protocol, qui a été conçu dès le départ pour répondre aux besoins actuels et futurs des applications, en termes de débit et de fonctionnalités.

Posséder une plateforme intégrant de façon similaire ces deux protocoles permettra ensuite de réaliser des expérimentations sur XTP et d'apprécier l'opportunité d'une éventuelle migration de TCP vers XTP.

Dans ce mémoire, le système de communications de la plateforme est d'abord introduit, en détaillant chacun de ses éléments. Puis, l'intégration du protocole XTP dans ce système de communications est présentée. Enfin, les protocoles sont comparés à la fois au niveau des performances et des fonctionnalités.

MOTS-CLES

réseaux, protocoles de communication, XTP, TCP/IP, étude de performances, intégration de protocoles, streams.

KEYWORDS

network, communication protocol, XTP, TCP/IP, performance studies, protocol integration, streams.