Titre: Title:	Convertibilité des protocoles de communication
Auteur: Author:	Senan Bertrand Tohio
Date:	2003
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Tohio, S. B. (2003). Convertibilité des protocoles de communication [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/6998/

UNIVERSITÉ DE MONTRÉAL

CONVERTIBILITÉ DES PROTOCOLES DE COMMUNICATION

SENAN BERTRAND TOHIO
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAITRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)

AVRIL 2003

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTREAL

Ce mémoire intitulé:

CONVERTIBILITÉ DES PROTOCOLES DE COMMUNICATION

présenté par : TOHIO Senan Bertrand

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. QUINTERO Alejandro, Ph.D., Président

M. PIERRE Samuel, Ph.D., membre et directeur de recherche

M. SAVARIA Yvon, Ph.D., membre et codirecteur de recherche

Mme. BOUCHENEB Hanifa, Ph.D., membre

À mes parents ...

REMERCIEMENTS

Je tiens à remercier d'une part le professeur Samuel Pierre, mon directeur de recherche, pour ses conseils pertinents et sa bienveillance en ce qui concerne l'accomplissement de ce projet, et d'autre part le professeur Yvon Savaria, mon codirecteur, pour ses conseils et son soutien constant tout au long de ma recherche.

Je désire ensuite remercier Maria M. Mbaye avec qui j'ai travaillé sur ce projet pour son aide et ses suggestions. Je suis aussi reconnaissant à mes collègues du laboratoire Larim, pour leurs critiques pertinentes et constructives tout au long de ce travail, ainsi qu'à tous ceux qui m'ont communiqué des références et documents m'ayant permis de mener à terme mon projet.

Je suis enfin redevable envers mes parents et frères pour leur soutien moral constant et leurs prières quotidiennes.

RÉSUMÉ

Le développement de l'Internet, des applications réparties et systèmes distribués donne aujourd'hui à l'interconnexion des réseaux de données une importance bien particulière. Cependant, la résolution de cette interconnexion n'est souvent pas aisée, compte tenu des différences entre architectures et protocoles utilisés dans les réseaux à interconnecter. Si jusqu'à ces jours, des équipements à fonctionnalités fixes comme les ponts, les commutateurs, les routeurs, demeurent les types de relais les plus fréquents, la tendance des nouvelles générations semble tourner vers des équipements à fonctionnalités multiples, capables de supporter d'une part les hauts débits actuels, et, d'autre part, plusieurs types de protocoles (de toute couche). Ces équipements, programmables, qui ont commencé à émerger depuis quelques années seulement sont appelés *processeurs réseaux*. Ils sont conçus pour opérer à la frontière des réseaux. La conversion de protocoles demeure l'une de leurs principales tâches.

Le présent travail se place essentiellement dans ce contexte et a pour objectif principal d'étudier la convertibilité des protocoles de communication dans cet environnement multiprotocole. Pour y parvenir, nous avons fait un survol de quelques protocoles courants, ce qui nous a permis d'identifier les obstacles à la convertibilité, puis les critères de convertibilité des protocoles. Comme contribution de ce mémoire, nous avons proposé une méthode permettant de déterminer si la conversion est réalisable entre deux protocoles quelconques et l'avons testé sur quelques protocoles courants. La conversion est ici considérée au sens large et donc inclut aussi la complémentation de protocole.

Une autre contribution réside en la conception d'un schéma de convertisseur générique, permettant de supporter plusieurs types de protocoles rencontrés dans les réseaux de communication. Nous avons implémenté le modèle proposé et en avons fait une évaluation de performance. Les résultats obtenus nous ont permis de tirer comme conclusion que le modèle générique proposé induit un délai supplémentaire de conversion par rapport à un convertisseur simple de protocoles, d'où une perte de

performance. Cependant, une topologie à plusieurs processeurs (comme celle des processeurs réseaux) permet de pallier la perte de performance constatée en diminuant le délai de conversion des paquets. Les tests nous ont montré que les architectures incluant un nombre maximal de dix processeurs permettent d'obtenir des délais de conversion inférieurs ou comparables à ceux d'un convertisseur spécifique. Il faut préciser que le nombre de processeurs idéal peut dépendre du nombre de paquets à convertir.

Pour terminer, nous nous sommes intéressés au cas de la conversion des protocoles Firewire et Ethernet. L'application de l'approche proposée nous a permis de constater que les méthodes de complémentation et d'encapsulation peuvent être utilisées pour permettre la communication entre ces deux protocoles, dans un cas général.

ABSTRACT

Internet expansion, distributed systems and applications development, give nowadays a great importance to data networks interconnection. However, this interconnection is often tricky to resolve, due to differences between networks architectures and protocols. Until these days, equipments with fixed functions such as bridges, switches, are used as gateways to deal with network interconnection problem. However, for next generation networks, new types of network equipments are being designed to support multiple functions, high data rates and multiple protocols (several layers). They will also be programmable and placed at networks' edges. They are referred as *networks processors*. One of their main functions resides in packet modification that is protocol conversion.

The purpose of this thesis is essentially to study protocol convertibility in this context. To achieve our goal, we first make a survey of a few well-known protocols. This helps us to identify obstacles to conversion and convertibility criteria. As contributions in this thesis, we have proposed a method to determine whether conversion is achievable between two protocols, and we have tested it on a set of commonly used protocols. Let us note that conversion is not restricted to direct conversion, but it also includes the complementation method. As other contributions, we have designed a generic protocol conversion architecture that can support several protocol conversions. We then implemented the proposed model and evaluated its performance.

The results obtained enabled us to deduct some conclusions that can be summarized in two main points. First, the generic model adds significant conversion delays for packets compared to a simple converter for the protocols of interest. However, a parallel processing machine that includes several processors can give better results. Our tests show that best performances are obtained with less than ten processors. We must also note that the ideal numbers of processors depends on the number of packets that are waiting in the conversion queue.

Finally, we consider the conversion of Firewire and Ethernet protocols. Firewire is a standard that is emerging today in domestic networking applications. Our proposed approach shows that complementation and encapsulation methods are useful to interconnect networks using these two protocols.

TABLE DES MATIÈRES

REMERCIEMENTS V

RÉSUMÉ VI

ABSTRACT VIII

TABLE DES MATIÈRES X

LISTE DES FIGURES XIII

LISTE DES TABLEAUX XV

SIGLES ET ABRÉVIATIONS XVI

CHAPITRE I : INTRODUCTION 1

1.1 Définitions et concepts de base 1

1.2 Éléments de la problématique 2

1.3 Objectifs de recherche 5

1.4 Plan du mémoire 5

CHAPITRE II : INTERCONNEXION DE RÉSEAUX ET CONVERSION DE PROTOCOLE 7

2.1 Définitions et concepts de base 7

2.1.1 Concepts de protocoles, services, et interfaces de communication 7

2.1.2 Survol du modèle de référence OSI 9

2.2 Interopérabilité dans les réseaux de communications 12

2.2.1 Caractérisation de l'interopérabilité 13

2.2.2 Approches usuelles d'interconnexion entre protocoles 15

2.2.3 Quelques équipements d'interconnexion usuels 20

2.3 Conversion de protocole 22

2.3.1 Architectures d'interconnexion impliquant la conversion 23

2.3.2 Algorithmes classiques de construction de convertisseur 25

2.3.3 Vers une concrète implémentation de la conversion 26

2.3.4 La conversion de protocole et le « network processing » 29

2.4 Synthèses et observations 31

CHAPITRE III : MODÈLE D'UN CONVERTISSEUR GÉNÉRIQUE DE PROTOCOLES 33

3.1 Concepts de base 33

3.2 Survol et analyse de protocoles usuels 35
3.2.1 Choix des protocoles 36
3.2.2 Analyse du survol 37
3.2.3 Critères de classification 38
3.2.4 Étude comparative de quelques protocoles 38
3.2.5 Observations et synthèse 43
3.3 Convertibilité des protocoles 44
3.3.1 Limites de la convertibilité 44
3.3.2 Quelques obstacles à la convertibilité 45
3.3.3 Critères de convertibilité 51
3.3.4 Méthode de détermination de convertibilité proposée 54
3.3.5 Exemple d'application de la méthode proposée 60
3.4 Schéma de convertisseur générique de protocoles 62
3.4.1 Description et fonctionnement des éléments de notre modèle 63
3.4.2 Définition du modèle de protocole 65
3.4.3 Algorithme de conversion 67
CHAPITRE IV : IMPLÉMENTATION, MISE EN ŒUVRE ET ANALYSE DE PERFORMANCE .. 70
4.1 Implémentation du convertisseur générique de protocoles 70
4.1.1 Diagramme de classes 70
4.1.2 Implémentation des classes et structures de données 72
4.1.3 Algorithme du programme principal 83
4.1.4 Mise en œuvre du convertisseur 84
4.2 Application de notre algorithme de convertibilité 85
4.2.1 Détails d'application de l'algorithme 85
4.2.2 Analyse des résultats 88
4.3 Analyse de performance 90
4.3.1 Influence du modèle générique 91
4.3.2 Architecture à processeurs multiples 92
4.4 Modèle d'un processeur réseau : la plateforme Intel IXP1200 97

4.4.1 Mode d'opération 98
4.4.2 Modèle de Thread 100
4.4.3 Approche de simulation utilisée 100
4.5 Conversion des protocoles Firewire et Ethernet 101
4.5.1 Le protocole Firewire 101
4.5.2 Principales différences entre les protocoles Firewire et Ethernet 105
4.5.3 Étude de la convertibilité de Firewire et Ethernet 106
4.5.4 Résolution de l'interconnexion Firewire - Ethernet 108
4.5.5 Aperçu de notre implémentation 110
CHAPITRE V : CONCLUSION 113
5.1 Synthèse des travaux 113
5.2 Limitations des travaux 115
5.3 Indications de recherches futures 116

BIBLIOGRAPHIE 118
ANNEXES 124

LISTE DES FIGURES

Figure 2.1 Dialogue entre deux EP selon le protocole P 8
Figure 2.2 Le modèle de référence OSI 10
Figure 2.3 Autres modèles en couches 10
Figure 2.4 Illustration des concepts de PDU et SDU 12
Figure 2.5 Types de relais 13
Figure 2.6 Interconnexion à l'aide d'un relais 14
Figure 2.7 Schéma de conversion de protocole 16
Figure 2.8 Schéma de conversion de service 17
Figure 2.9 Schéma d'encapsulation de protocole 18
Figure 2.10 Complémentation de protocole 19
Figure 2.11 Conversion de protocole 22
Figure 2.12 Conversion à l'émission et à la réception 24
Figure 2.13 Schéma de la conversion dans les nœuds terminaux 25
Figure 2.14 Principaux modules d'un processeur réseau 30
Figure 3.1 Correspondance entre PCI de Ethernet et Token Ring 39
Figure 3.2 Informations de contrôle des protocoles HDLC et PPP 40
Figure 3.3 Protocoles intervenant dans le transfert de fichiers 45
Figure 3.4 Communication entre Pc (connexion) et Pd (datagramme) 47
Figure 3.5 Problème de conversion d'adresses au niveau du relais 48
Figure 3.6 Problème d'adressage rendant l'interconnexion difficile 49
Figure 3.7 Méthodologie de Green 55
Figure 3.8 Intersection entre les services de P1 et P2 57
Figure 3.9 Algorithme de convertibilité proposé 58
Figure 3.10 Schéma simplifié des modules de notre convertisseur de protocoles 63
Figure 3.11 Modèle de l'objet protocole 66
Figure 3.12 Algorithme de la fonction de conversion (P1→ P2) 69
Figure 4.1 Diagramme UML des différentes classes 71

Figure 4.2 Structure des classes *Protocol*, *Protocol_bit*, *Protocol_car* et *Message* 76
Figure 4.3 Structure des classes *Field, Header_Fields, PayloadField* et *Packet* 80
Figure 4.4 Structure interne des classes principales du convertisseur 82
Figure 4.5 Algorithme de la fonction principale 83
Figure 4.6 Délai de conversion des paquets (FDDI--Ethernet) 91
Figure 4.7 Délai de conversion des paquets (Token-Ring à Ethernet) 92
Figure 4.8 Impact du nombre de processeurs sur le délai de conversion 95
Token-Ring à Ethernet 95
Figure 4.9 Impact du nombre de processeurs sur le délai de conversion 96
FDDI à Ethernet 96
Figure 4.10 Architecture du processeur réseau Intel IXP1200 99
Figure 4.11 Connexion entre le processeur réseau et le PC 100
Figure 4.12 Un Bus Firewire 102
Figure 4.13 Le modèle en couches de Firewire 103
Figure 4.14 Format des paquets asynchrones et isochrones 104
Figure 4.15 Principales différences entre les protocoles Firewire et Ethernet 105
Figure 4.16 IP sur Firewire et IP sur Ethernet 108
Figure 4.17 Algorithme de conversion Firewire-Ethernet 112

LISTE DES TABLEAUX

Tableau 3.1 Correspondance entre éléments de service de TCP, NSP et ADSP 41
Tableau 3.2 Correspondance entre éléments de service de protocoles de messagerie 42
Tableau 4.1 Définition des services à la couche OSI 2 85
Tableau 4.2 Définition des services à la couche OSI 3 86
Tableau 4.3 Définition des services à la couche OSI 4 87
Tableau 4.4 Définition des services primaires 87
Tableau 4.5 Résultats de l'application de l'algorithme de convertibilité 89
Tableau 4.6 Différents types de paquets asynchrones de Firewire 104

SIGLES ET ABRÉVIATIONS

ACK	: Acknowledgement (ou accusé de réception)
ATM	: Asynchronous Transfer Mode
ARP	: Address Resolution Protocol
CLNP	: Connectionless Network Protocol
DECNET	: Digital Equipment Corporation Network
FDDI	: Fibber Distributed Data Interface
FTP	: File Transfer Protocol
IEEE	: Institute of Electrical and Electronics Engineers
IP	: Internet Protocol
ISO	: International Standard Organization
LAN	: Logical Network Area
LU	: Logical Unit
MAC	: Medium Access Control
NSP	: Network Service Protocol
OSI	: Open System Interconnection
PCI	: Protocol Control Information
QoS	: Quality of Service
RFC	: Request for Comment
SNA	: IBM Systems Network Architecture
TCP	: Transmission Control Protocol
VC	: Virtual Circuit

CHAPITRE 1
INTRODUCTION

Les réseaux de communication constituent aujourd'hui l'une des principales voies de transport de l'information. Ils sont composés d'équipements qui gèrent le transport de l'information depuis la source jusqu'à la destination. Les règles qui gouvernent ces échanges d'information sont appelées protocoles de communication. La prolifération des réseaux de communications au cours des dernières décennies a conduit à l'émergence d'architectures et de protocoles multiples qui rendent parfois difficile la communication entre les usagers de ces réseaux, puisque chaque réseau obéit à sa propre architecture et implante sa suite de protocoles. Or, pour que l'information puisse circuler sur ces réseaux, il faut trouver le moyen de les interconnecter. Diverses approches, comme par exemple la conversion de protocole, l'encapsulation, la complémentation, ont été suggérées et sont utilisées pour résoudre le problème d'interconnexion de réseaux. Ce mémoire traite un des aspects du problème d'interconnexion, à savoir la convertibilité des protocoles de communication dans les réseaux. Dans ce chapitre d'introduction, nous expliciterons d'abord quelques concepts de base et les éléments de la problématique. Par la suite, nous mentionnerons nos objectifs de recherche, la méthodologie à suivre et enfin le plan du mémoire.

1.1 Définitions et concepts de base

Un *réseau de communication* est un ensemble de machines communicantes reliées entre elles, permettant ainsi à des usagers (connectés à ces machines) de communiquer ou d'accéder à distance à des ressources situées sur les autres machines du réseau. Aujourd'hui, il en existe des multitudes et ils sont incompatibles, puisque chaque organisation (entreprises, universités, gouvernements etc.) crée son propre réseau pour faire partager ses ressources par ses utilisateurs. Les *protocoles de communication* définissent le format et la sémantique des messages échangés.

L'*interconnexion de réseaux* force à gérer le problème d'incompatibilité entre architectures et protocoles de réseaux différents. Elle est généralement réalisée grâce à des équipements désignés sous le nom générique de *relais ou passerelle (gateway)*. Le relais gère deux types de problème : l'un de type architectural, c'est à dire les couches où l'interconnexion aura lieu, et l'autre plus important qui est la communication entre protocoles, la gestion des différences de formats de messages et des règles pour l'acheminement de ces messages (Kristol et al., 1993). L'une des méthodes connues pour assurer ce dialogue entre protocoles est *la conversion de protocole*. Elle permet de faire une correspondance syntaxique et sémantique des messages émis par les protocoles communicants et d'assurer la synchronisation nécessaire entre eux.

Deux niveaux de conversion peuvent être considérés: la conversion au niveau protocole à l'aide d'un convertisseur de protocole et la conversion au niveau service à l'aide d'un adaptateur d'interface de service. La première réalise explicitement la correspondance entre les messages des protocoles P et Q pour chaque élément de service offert. Elle résout l'interconnexion aux couches où opèrent les protocoles à convertir et tend à fournir la conversion pour chaque fonctionnalité tandis que la seconde fait abstraction de l'implémentation des protocoles et considère seulement leurs interfaces ou primitives de service pour réaliser la conversion.

1.2 Éléments de la problématique

Divers travaux ont été réalisés pour synthétiser des convertisseurs de protocole (Okumura, 1986; Calvert et Lam, 1989; Tao et al., 1995; Liu et Yao, 1992). Ces travaux d'ordre général font abstraction des services offerts par les protocoles réels utilisés dans les réseaux. Ils utilisent pour la plupart les mêmes concepts et méthodes formelles employés dans la conception et la validation de protocole. Dès lors, les spécifications d'un protocole sont représentées par un ensemble d'entités de protocole communicant entre eux. Le comportement de chaque entité de protocole est modélisé par un automate sous la forme d'une machine à états finis communicants. On peut distinguer deux grandes catégories d'approches : celles qui utilisent les méthodes formelles pour

effectuer la synthèse du convertisseur à partir des requis de conversion de niveau protocole (ces requis indiquent les obligations en termes de traduction et de synchronisation des messages) du convertisseur, et celles qui font la synthèse plutôt à partir des spécifications de service que le système final doit offrir.

La majeure partie des recherches (Okumura, 1986; Calvert et Lam, 1989; Tao et al., 1992; Shu et Liu, 1990; Chang et Liu, 1990; Liu et Yao, 1992; Kristol et al., 1993; Tao et al, 1995) dans le domaine de la conversion de protocole ont utilisé ces méthodes formelles. Basées sur une modélisation purement théorique et abstraite du problème de conversion, celles-ci permettent une spécification formelle du convertisseur et utilisent des algorithmes qui demandent un travail minutieux et un temps de calcul assez élevé qui croit exponentiellement avec le nombre d'états possibles des protocoles. De plus, il n'est pas garanti qu'une spécification de convertisseur valide soit toujours réalisable.

Cependant, certains chercheurs (Green, 1986; Sunshine, 1990; Svobodova et al., 1990; Takei et al., 2001) se sont penchés sur un aspect un peu plus pratique du problème d'hétérogénéité; ils ont évalué les services offerts en général par les protocoles dans les réseaux et ont identifié quelques problèmes inhérents aux mécanismes et fonctionnalités de divers protocoles utilisés aujourd'hui, lorsqu'il s'agit de les faire interopérer. Quelques grandes lignes de ces problèmes ont été soulignées, telles que les différences entre architectures, les vitesses de communication, le service datagramme versus circuit virtuel, les différences entre mécanismes de routage, les variations de la taille des paquets d'un protocole à l'autre etc. L'hétérogénéité peut arriver à n'importe quelle couche du modèle de référence OSI. Si pendant longtemps les efforts se sont concentrés au niveau des protocoles des couches basses, de plus en plus les intérêts de recherche considèrent aussi les couches hautes. En effet, il a été constaté que le problème d'interopérabilité se pose encore bien plus souvent dans les nœuds terminaux des systèmes. Pour ces cas, l'implémentation d'une librairie au nœud final permet généralement de résoudre le problème (Auerbach, 1989). Des méthodes furent proposées, par exemple, pour réaliser la communication entre certains protocoles des

couches hautes du modèle de référence dont la couche transport (Auerbach, 1989) et la couche session (Takei, Okamura et Araki, 2001).

Traditionnellement, l'interconnexion a souvent considéré deux protocoles bien précis. Les équipements d'interconnexion usuels comme les routeurs, les commutateurs, les passerelles, opèrent souvent à une couche bien précise et réalisent des fonctions de classification de paquets, de gestion de trafic, et parfois de modification de paquets. Pour offrir les services de nouvelle génération qui impliquent une convergence de la voix, des données et de la vidéo, tels la vidéoconférence, la qualité de service (QoS) différenciée, les concepteurs de circuits intégrés se concentrent sur la construction de processeurs réseaux (circuits hautement intégrés) qui vont pouvoir supporter plusieurs fonctionnalités que l'on retrouve dans les équipements réseaux. Ces circuits intégrés vont pouvoir assumer des fonctions de base comme la classification, la modification, le cryptage de paquets et la gestion du trafic. La modification de paquets implique la conversion de protocole. Ces circuits intégrés de haute performance sont programmables et équipés d'outils qui vont permettre de décrire les protocoles en utilisant un langage de haut niveau. Ils seront placés à la frontière, entre le réseau et l'usager (endroit où l'usager se connecte au fournisseur de service), et devront donc être capables de faire la reconnaissance et la conversion de protocoles. Le défi principal réside dans le fait que ces processeurs doivent être conçus pour opérer à n'importe quelle couche. Considérant la variété des protocoles et leur nombre, il est normal de se poser des questions sur la convertibilité éventuelle de deux protocoles quelconques.

Le problème de la convertibilité des protocoles peut se définir donc comme suit : étant donné deux protocoles quelconques P et Q offrant chacun un ensemble bien précis (fini) de services et dont les mécanismes de fonctionnement sont bien connus, est-il possible d'envisager une conversion entre P et Q, c'est à dire un passage des messages d'un équipement comprenant le protocole P à un autre comprenant Q et vice-versa, d'une manière transparente à l'usager de ces équipements. Dans cet environnement multiprotocole où doivent opérer les processeurs réseau, la première question à laquelle il faut trouver une réponse est de savoir s'il est possible de construire une architecture de

convertisseur qui soit capable de supporter tous les protocoles de communication existants et futurs.

1.3 Objectifs de recherche

L'objectif de ce mémoire est d'identifier les conditions nécessaires à la convertibilité des protocoles tout en faisant ressortir les éléments éventuels qui constituent des obstacles lors de la conversion, puis de proposer des approches de solution. Nous considérons la conversion au sens large, c'est-à-dire incluant la complémentation. Plus spécifiquement, ce mémoire vise à:

- étudier la faisabilité d'un convertisseur générique de protocole en identifiant les facteurs qui constituent des obstacles à la convertibilité ;
- esquisser une approche qui permette de prédire si, pour deux protocoles donnés, une conversion est possible de sorte à respecter les services à offrir aux couches supérieures sur la base des spécifications de ces protocoles (format des messages, services offerts), sans passer par des méthodes de spécification formelles ;
- proposer et évaluer un schéma de convertisseur générique de protocoles en ayant soin d'en prouver la convertibilité.

1.4 Plan du mémoire

La suite du mémoire est organisée comme suit. Dans le chapitre 2, nous présenterons les concepts de base inhérents au domaine des réseaux et protocoles, discuterons des architectures d'interconnexion de réseaux et ferons une synthèse de la littérature relative au domaine de la conversion de protocole.

Dans le chapitre 3, nous étudierons un ensemble de protocoles et déduirons les critères de base qui permettront d'effectuer une classification des protocoles. Nous pourrons alors analyser plus amplement certaines situations de conversion et montrer comment certaines propriétés des protocoles peuvent constituer des obstacles à la conversion. Nous proposerons ensuite une méthode permettant de prédire la

convertibilité de deux protocoles quelconques et un schéma de convertisseur générique de protocole.

Au chapitre 4, nous présenterons quelques résultats de la méthode proposée, appliquée à quelques protocoles courants, et l'implémentation de notre convertisseur générique. Des mesures de performance nous permettront de tirer des conclusions quant à l'efficacité du schéma de convertisseur proposé. Nous considérerons, par la suite, le cas de la conversion des protocoles Firewire (IEEE1394) et Ethernet. Dans un premier temps, nous étudierons la faisabilité de cette conversion grâce à l'approche proposée, identifierons les obstacles, puis présenterons les méthodes de résolution proposées.

Pour terminer, le chapitre 5 fait une synthèse du travail réalisé, des résultats obtenus, des limitations de notre approche et donne des directions en vue de recherches futures.

CHAPITRE II
INTERCONNEXION DE RÉSEAUX ET CONVERSION DE PROTOCOLE

Dans ce chapitre, nous passons en revue les méthodes usuelles d'interconnexion de réseau et les situations impliquant la conversion de protocole. Nous définirons d'abord quelques concepts de base, caractériserons l'interopérabilité et présenterons quelques approches fréquemment utilisées pour réaliser l'interconnexion. Par la suite, nous discuterons des architectures impliquant la conversion et ferons un survol de quelques travaux en rapport avec la conversion de protocole. Nous terminerons en présentant quelques observations par rapport aux travaux actuels et les motivations de notre recherche dans le cadre de la nouvelle génération d'équipements réseaux nommés «*processeurs réseau*».

2.1 Définitions et concepts de base

Dans cette section, nous introduisons quelques concepts dont nous nous servirons par la suite. Nous rappelons brièvement les notions de protocole, service, interface de communication et faisons un bref survol du modèle de référence par rapport auquel nous allons nous situer tout au long de notre travail.

2.1.1 Concepts de protocoles, services, et interfaces de communication

D'une manière générale, un protocole est un ensemble de règles qui dictent le déroulement d'une activité ou les échanges entre les partenaires durant une activité selon une certaine convention. En téléinformatique, ce mot désigne un ensemble de règles syntaxiques et sémantiques que doivent suivre les entités communicantes pour la transmission des données. Ainsi, dans le domaine des réseaux de communications, les *protocoles* sont essentiellement les règles préétablies sur lesquels se fondent les

programmes utilisés pour organiser les échanges d'informations avec d'autres systèmes, à travers les supports de télécommunications utilisés.

Nous nommerons *entité de protocole* (EP) un système qui implante un protocole donné. Ainsi, une entité de protocole *A* pourra communiquer facilement avec une autre entité *B* implantant le même protocole *P* comme indiqué à la Figure 2.1.

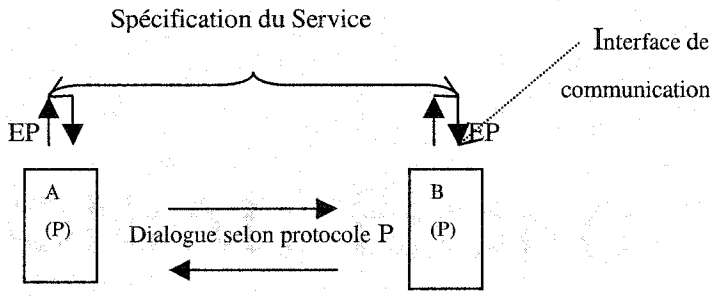


Figure 2.1 Dialogue entre deux EP selon le protocole P

Un *service* est une fonctionnalité offerte par une entité. Dans un réseau de communications, les entités communicantes vont s'échanger des messages pour offrir un service donné. Le transport des données, le routage, la liaison de données, le contrôle de flux, sont des exemples de service offerts dans les réseaux de communications. Ce concept est souvent utilisé étroitement avec la notion de protocole, surtout lorsqu'il s'agit de faire une classification. C'est d'ailleurs le fondement du modèle de référence en couches défini par l'ISO, dans laquelle chaque couche possède des fonctions spécifiques et offre des services bien déterminés à la couche supérieure, en s'appuyant sur ceux qui sont offerts au niveau inférieur. Les entités de même niveau communiquent par l'intermédiaire des services fournis par la couche inférieure. Ainsi, une entité de la couche N+1 utilisera les services de la couche N pour fournir les services N+1.

Nous pourrons alors redéfinir la notion de protocole de communication d'une couche N comme un ensemble de règles syntaxiques et sémantiques gouvernant les échanges entre deux entités communicantes de niveau N, afin d'utiliser le service de niveau N-1 pour fournir à leur tour un service de niveau N (Kristol et al., 1993).

Une entité de protocole va offrir un service à partir d'une *interface de communication*. C'est le point où il reçoit les requêtes de l'usager et fournit à ce dernier les services demandés.

2.1.2 Survol du modèle de référence OSI

Compte tenu de la diversité des protocoles et pour assurer l'interopérabilité entre les différents réseaux, l'ISO a élaboré en 1980 une architecture générale pour le développement des normes que l'on nomme communément « *modèle de référence OSI*» pour l'interconnexion des systèmes ouverts, suffisamment riche pour supporter les principaux aspects de la communication entre systèmes informatiques. Il est fondé essentiellement sur la notion de service. Chaque couche de ce modèle de référence définit un type de service, ce qui permet de classer les protocoles selon les services qu'ils offrent. Il comporte sept couches hiérarchisées, chacune remplissant un ensemble de fonctions homogènes comme l'illustre la Figure 2.2.

La première couche, la *couche physique*, assure la transmission sur lien physique des unités de données entre deux nœuds quelconques. La seconde, la *couche liaison de données,* fournit un lien virtuel de communication pour l'envoi des paquets et assume des mécanismes de contrôle d'erreurs, de flot et de séquence. La troisième couche, la *couche réseau* assume les fonctions de routage, contrôle de séquence, contrôle de flot et de congestion. La *couche transport* est la quatrième couche; elle gère le transport des données et assume aussi des mécanismes d'adressage, de synchronisation de contrôle d'erreur et de flux de bout en bout. La *couche session* organise et structure les sessions de dialogue entre entités coopérantes. Quant à la *couche présentation*, elle transforme les données dans une forme standardisée, acceptable des entités qui utilisent le service. Elle réalise le cryptage et la compression de données. Enfin, la *couche application* fournit à l'usager une interface avec un ensemble de services d'information distribués (ex: transfert, accès et gestion de fichiers, etc.). En résumé, les couches basses (1 à 3) réalisent les fonctions liées à la transmission de données, tandis que les couches hautes (4 à 7) s'occupent de la communication entre applications.

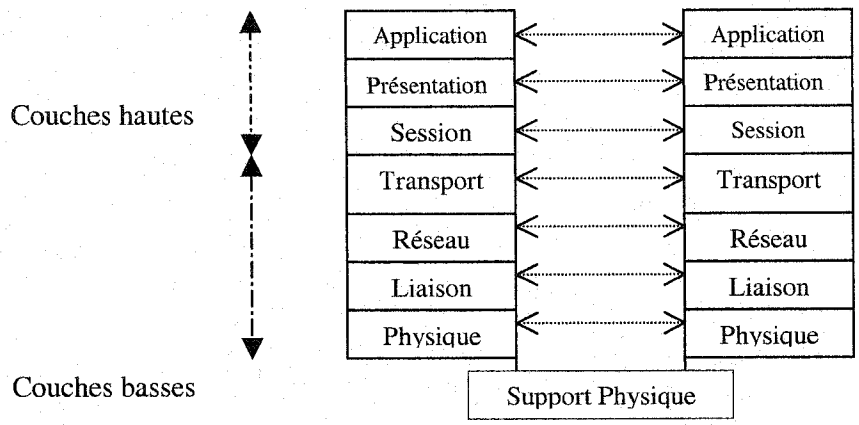
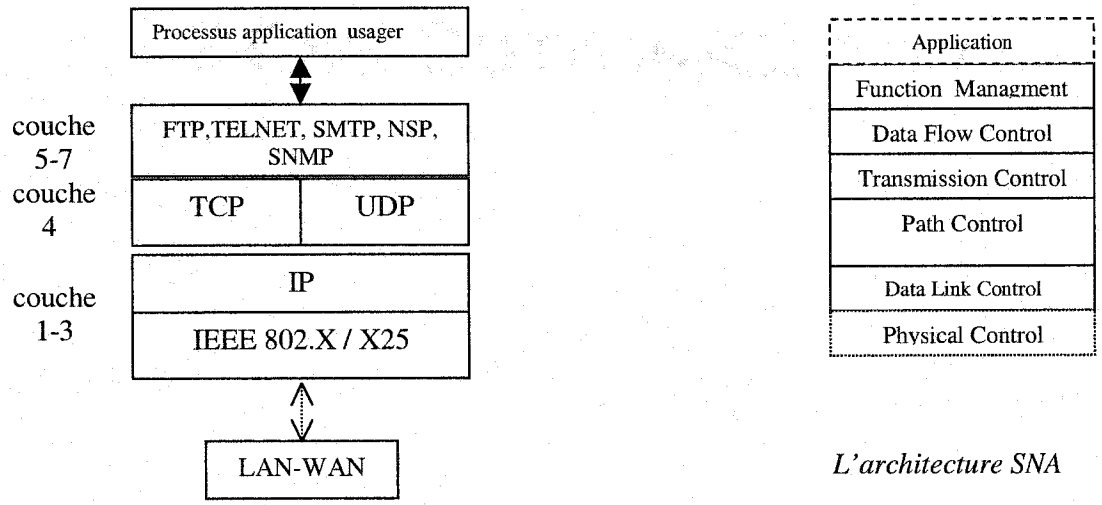


Figure 2.2 Le modèle de référence OSI



Le modèle TCP-IP

L'architecture SNA

Figure 2.3 Autres modèles en couches

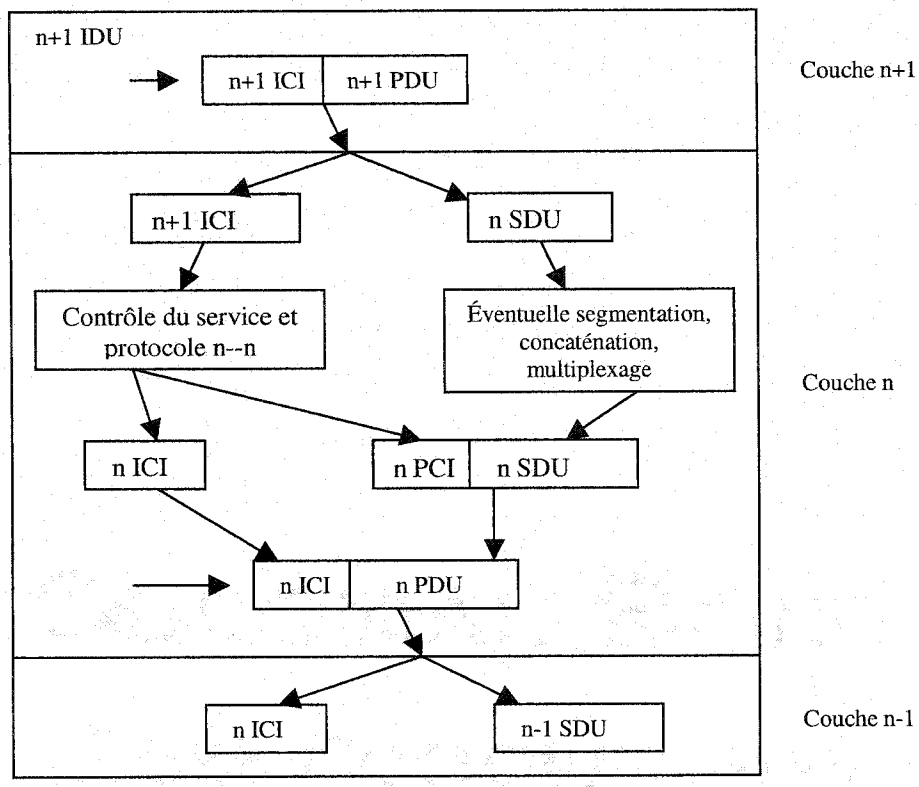
Plusieurs autres architectures (propriétaires) en couches coexistent cependant, malgré l'effort de standardisation de l'ISO, comme le modèle SNA de IBM, celui de TCP/IP largement utilisé dans Internet et qui est même plus apprécié (Figure 2.3).

Une entité à une couche *N* fournit un ensemble de services (appelés *N*-services) à l'entité de niveau *N*+1. Les entités communicantes de niveau *N* sont des *entités paires*. L'ISO a aussi défini quelques concepts importants : *SAP* (Service Access Point), *PDU*

(Protocol Data Unit), *SDU* (Service Data Unit). Deux couches adjacentes dialoguent au moyen de primitives de service à travers une interface normalisée appelée SAP. L'endroit où la couche supérieure *N*+1 accède aux services de niveau *N* est appelé *point d'accès de service de niveau N* noté N-SAP. Les entités paires de niveau *N* communiquent via des *unités de données de protocole* (PDU) ou *N*-PDU (pour spécifier qu'on se situe à la couche *N*). Elles deviendront des unités de données de service (SDU) à la couche suivante. Ainsi, les *N*-SDU sont en réalité les (*N*+1) PDU en supposant que l'information passe de la couche *N*+1 à la couche *N*. La Figure 2.4 donne un aperçu des relations entre les différents concepts précités.

Ainsi, pour une couche *N*, il existe deux niveaux d'abstraction : *le niveau service N* et *le niveau protocole N*. Le service *N* est la fonctionnalité à offrir aux entités de la couche *N*+1, et le protocole *N* définit le comportement des entités dans la couche *N*. De même, les *primitives de service* sont introduites pour décrire l'interface entre des couches adjacentes *N* et *N*+1. Elles peuvent être résumées comme suit :

- Requête : primitive émise par un usager du service *N* pour invoquer une fonction particulière (ex : demande de connexion, transfert de données etc);
- Indication : émise par le fournisseur de service indiquant que l'utilisateur du service *N* correspondant a demandé un service donné;
- Réponse : primitive émise pour signaler qu'un service demandé peut être complété;
- Confirmation : émise par le fournisseur de service indiquant que les services demandés par l'usager de service *N* sont complétés.



Légende
SAP (Service Access Point) : point d'accès au service
IDU (Interface Data Unit): Unité de données d'interface
ICI (Interface Control Information) : informations de contrôle d'interface
PDU (Protocol Data Unit) : Unité de données du protocole
SDU (Service Data Unit) : données du service

Figure 2.4 Illustration des concepts de PDU et SDU

2.2 Interopérabilité dans les réseaux de communications

Dans cette section, nous rappelons le problème d'interopérabilité, résumons brièvement quelques approches utilisées pour traiter ce problème et mentionnons quelques équipements réseaux qui, en pratique, sont utilisés pour le résoudre.

2.2.1 Caractérisation de l'interopérabilité

Le développement de la téléinformatique et des applications connexes, en particulier les systèmes répartis, et surtout l'émergence des applications qui se fondent sur l'Internet, a rendu indispensable le besoin de mettre en relation des réseaux physiquement distincts, indépendamment des types de protocoles qu'ils utilisent. Dans certains cas, les réseaux sont du même type et utilisent les mêmes protocoles. L'interconnexion est alors relativement facile à réaliser avec des équipements d'interconnexion tels les répéteurs, les routeurs. Cependant, il arrive fréquemment que les machines à relier soient situées dans des réseaux qui obéissent à des architectures complètement différentes, donc utilisant des protocoles différents. Dans ces cas, il faut trouver un moyen propice pour relier les deux réseaux de manière à ce que l'interconnexion soit transparente.

Les dispositifs électroniques qui assument cette fonctionnalité sont des *relais*. Mais au lieu d'être un équipement (matériel), le relais peut être aussi un réseau de transport. C'est le cas par exemple lorsqu'on interconnecte deux réseaux TCP-IP en utilisant un réseau de transport X.25. La Figure 2.5 illustre les types de relais.

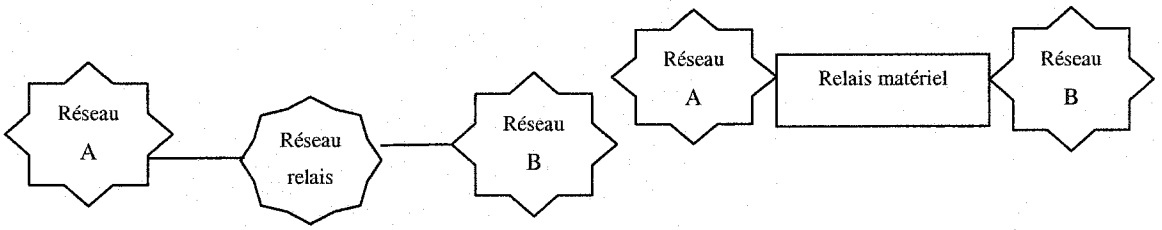


Figure 2.5 Types de relais

L'interconnexion peut être définie comme « toute technique permettant à un réseau de communiquer avec ou d'utiliser les services sur un autre réseau » (Sunshine, 1990). La suite de protocoles utilisée dans un réseau *A* n'étant pas forcément la même dans le réseau *B*, il devient alors important de trouver un moyen pour faire passer les messages d'un réseau à l'autre, de sorte que les services fournis restent les mêmes. C'est le rôle du

relais. La Figure 2.6 illustre le dialogue entre deux machines situées dans des réseaux différents *A* et *B* respectivement, qui à une couche *N* quelconque, utilisent des protocoles différents. Le relais va gérer l'hétérogénéité au niveau de cette couche, de sorte que les messages envoyés par la machine 1 selon le protocole *B* soient compris par la machine 2 selon le protocole *C* et vice-versa.

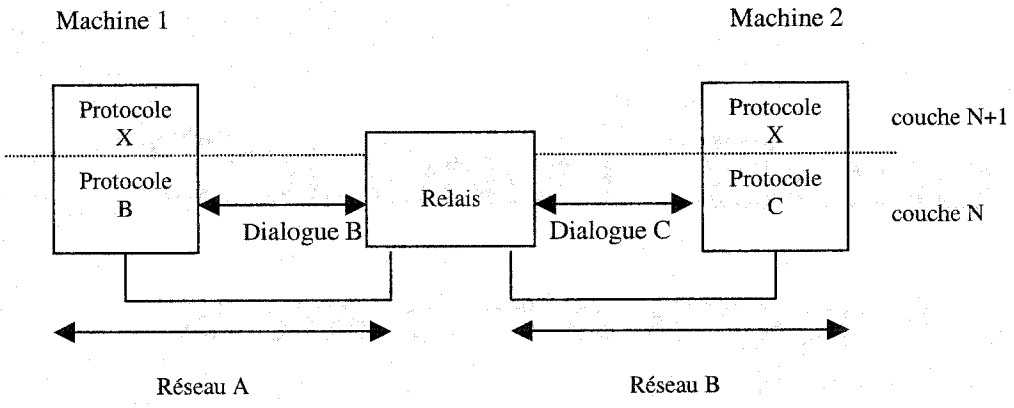


Figure 2.6 Interconnexion à l'aide d'un relais

Une solution au problème d'interopérabilité entre protocoles est de modifier l'une des entités communicantes ou les deux, de sorte qu'elles utilisent le même protocole. Cela peut coûter cher, surtout lorsque plusieurs couches de l'architecture rentrent en jeu. Parmi les approches existantes utilisées pour résoudre le problème d'interopérabilité, les plus connues sont : la conversion de protocole, la conversion de service, l'encapsulation et la complémentation de protocole qui s'apparente à une forme de conversion de service (Chang et Liu, 1990). Ces techniques peuvent aussi bien être implantées au niveau d'un relais ou d'un nœud final (destination).

Il est important de faire la distinction entre la fonction d'un relais et celle d'un convertisseur qui réalise l'une des formes de conversion énumérées précédemment. Un relais est un système qui représente un point particulier de transit de données entre plusieurs réseaux physiques ou logiques, peu importe que les protocoles utilisés soient

les mêmes ou non. Le convertisseur fait la correspondance à une couche précise entre différents protocoles de communication. Il peut être implémenté au niveau d'un relais ou dans un nœud terminal (Fluckiger, 1988). Nous allons, dans la prochaine section, expliciter chacune des techniques d'interconnexion précitée.

2.2.2 Approches usuelles d'interconnexion entre protocoles

Considérons deux systèmes informatiques S1 et S2 devant communiquer entre eux et obéissant respectivement à des architectures *A* et *B* bien définies. On suppose que dans l'architecture *A*, la suite de protocoles opérant au-dessus de la couche *M* est la même ou équivalente à la suite de protocoles opérant au-dessus de la couche *N* de l'architecture *B*. Cependant, les protocoles opérant dans les couches *M* et *N* sont différents ou incompatibles. Il faut alors trouver le moyen de réaliser la communication entre les deux systèmes de manière à offrir, de bout en bout, les services de la couche *M* et ceux de la couche *N* respectivement dans les architectures *A* et *B*, aux couches supérieures adjacentes. Nous allons dans la suite présenter sommairement quelques techniques fréquemment utilisées pour réaliser la communication entre des protocoles différents : la conversion de protocole, la conversion de service, la complémentation, l'encapsulation.

2.2.2.1 Conversion de protocole

Cette approche suggère l'implémentation d'un convertisseur qui va convertir les paquets de données émis selon le protocole de la couche *N* (*N*-PDU) de sorte qu'ils soient compréhensibles par le protocole de la couche *M*. La conversion a lieu au niveau de chaque unité de données de protocole (PDU) de la couche *M* de l'architecture *A* et de la couche *N* de l'architecture *B*. Si nous nommons *P* et *P'* respectivement les protocoles des couches *M* et *N*, la conversion assure le respect de la sémantique de bout en bout au niveau des formats et fonctionnalités internes des protocoles *P* et *P'* et la synchronisation existant entre eux (Figure 2.7).

Cette approche a suscité beaucoup d'intérêt dans et nombre de travaux ont été réalisés pour proposer des algorithmes ou méthodes efficaces pour générer la spécification d'un convertisseur correct entre deux protocoles de manière générale (Okumura, 1986; Lam, 1988; Chang et Liu, 1990). La conversion entre PDU est faisable lorsque les protocoles sont similaires ou peuvent être considérés similaires en faisant certaines restrictions sur leur implémentation (Svobodova et al., 1990). L'application de la conversion de protocole est, par exemple, appropriée dans le cas de l'interconnexion entre un réseau Ethernet et un réseau Token Ring.

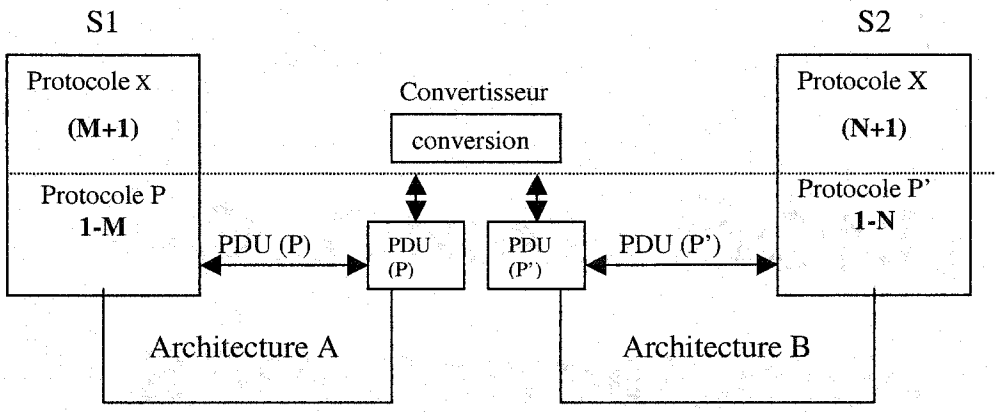


Figure 2.7 Schéma de conversion de protocole

2.2.2.2 Conversion de service

La conversion a lieu entre les primitives compatibles (ou séquences de primitives) offerts par les services de la couches *M* de l'architecture *A* et ceux de la couche *N* de l'architecture *B*. De ce fait, le nœud d'interconnexion (relais) agit comme un nœud terminal pour les protocoles en jeu dans chaque architecture. La conversion de bout en bout a lieu seulement au niveau des primitives de services (Svobodova et al., 1990). La conversion de service est généralement plus simple à réaliser et peut même constituer dans certaines situations la seule solution à envisager. Elle est appropriée lorsque les protocoles originaux sont radicalement différents mais offrent des services similaires (ex : mécanismes de contrôle de flux différents). L'un des principaux atouts de cette

méthode est que l'on peut faire abstraction totalement de l'implémentation des protocoles, étant donné que la conversion se réalise au-dessus des couches *M* et *N* considérées. La conversion est opérée au niveau des unités de données de service (SDU) plutôt que de celles du protocole (PDU). La Figure 2.8 en donne une illustration.

Selon le degré de ressemblance entre les services offerts par les protocoles, on peut distinguer différentes sortes d'interconnexion au niveau service: la concaténation de service et l'adaptation d'interface de service. La première correspond au cas où les services fournis par chacun des deux protocoles s'équivalent et il existe une correspondance entre les interactions initiées par chacun des protocoles agissant dans les couches *M* et *N*. L'adaptation d'interface de service concerne les situations où les services ne sont pas similaires mais il existe un ensemble de services communs aux deux protocoles qui satisfont aux requis des couches supérieures (Sunshine, 1990).

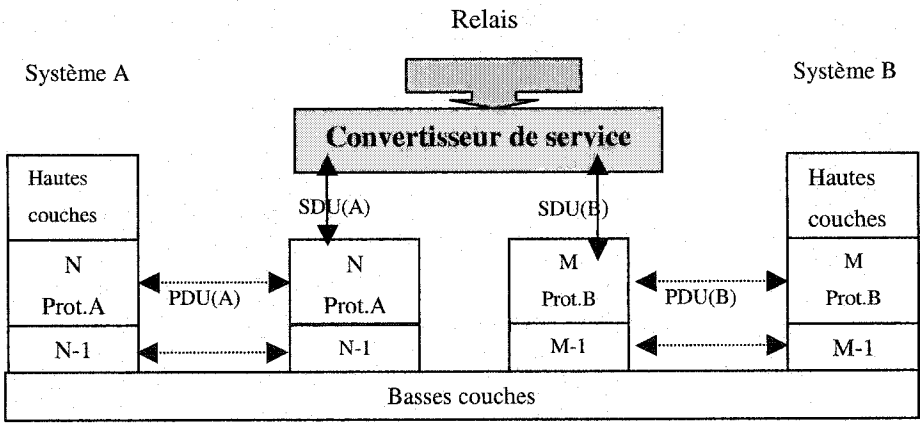


Figure 2.8 Schéma de conversion de service

2.2.2.3 Encapsulation

L'encapsulation consiste à utiliser ou modifier le protocole de l'un des réseaux pour engloutir le protocole de l'autre réseau. Il arrive des cas où des paquets de données issus d'un réseau local donné, à destination d'un autre réseau similaire, doivent transiter par un réseau intermédiaire qui utilise un moyen de transport différent de celui du

premier. L'interconnexion de réseau TCP-IP via un réseau X25 en est un exemple bien illustratif. La solution utilisée dans ce genre de situation consiste à transporter les PDU du protocole *A* dans ceux du protocole *B* (au niveau du premier relais) et à faire l'opération inverse au niveau du relais final (Chan et Liu, 1990). Le PDU du protocole *A* est alors restitué. La Figure 2.9 illustre ce phénomène.

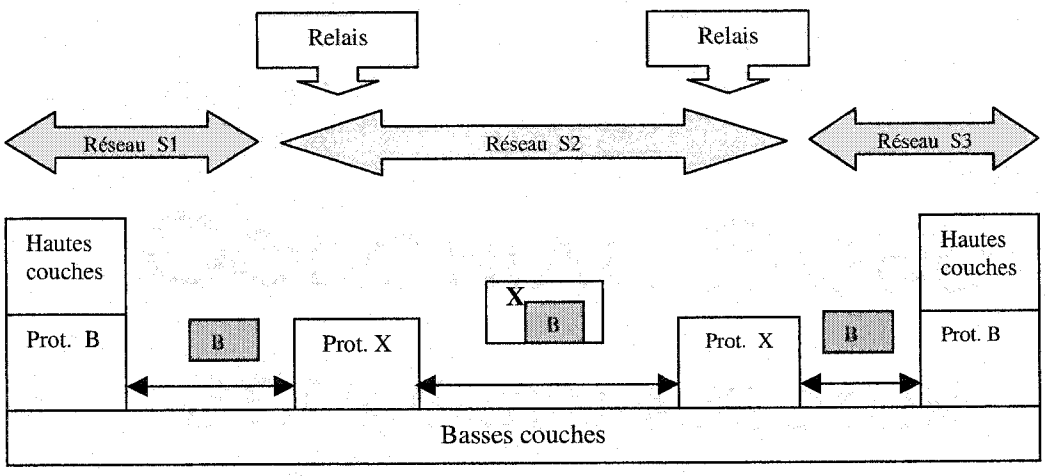


Figure 2.9 Schéma d'encapsulation de protocole

2.2.2.4 Complémentation de protocole

Les approches énumérées précédemment deviennent inefficaces dans le cas d'une grande incompatibilité entre les protocoles, c'est à dire que l'un des protocoles n'offre pas les fonctionnalités (services) de l'autre. Pour pallier ce problème, une méthode consiste à construire une couche virtuelle au-dessus des protocoles originaux de manière à faire disparaître la disparité entre eux. François et Potocki (1983) indiquent que cette solution devient incontournable lorsque les protocoles ne sont pas convertibles et diffèrent seulement sur certains aspects. La complémentation permet d'augmenter les services de l'un ou l'autre des protocoles ou les deux, de manière à étendre les fonctionnalités communes des deux protocoles. Elle est proche du problème de synthèse de protocole qui consiste à générer les spécifications de protocole à partir de celles du

service offert. La couche virtuelle est construite à partir des protocoles de sorte qu'elle contienne un sous-ensemble des états de chaque protocole. Si on considère de nouveau nos systèmes *S1* et *S2* ainsi que les couches *M* et *N* des architectures *A* et *B* respectives, la complémentation permettrait d'augmenter soit les services de *M*, soit ceux de *N* ou des deux (Figure 2.10). La complémentation peut être considérée, sous un certain angle comme une sorte de conversion de service.

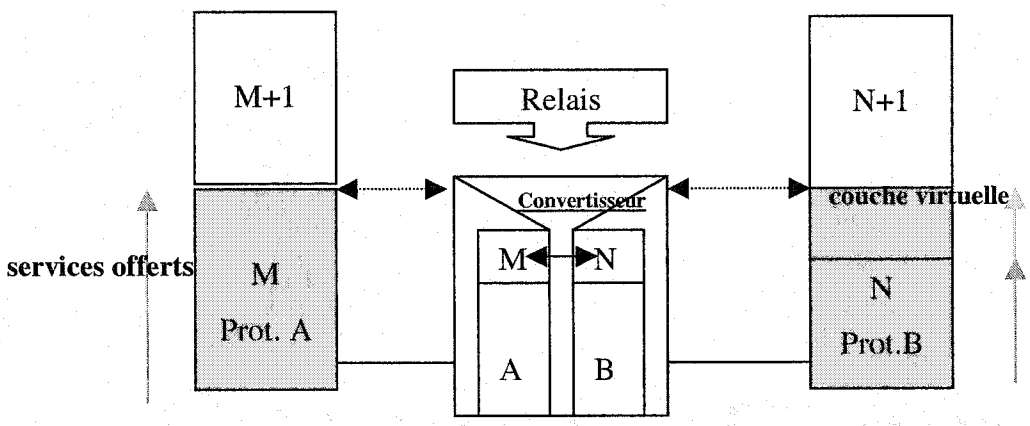


Figure 2.10 Complémentation de protocole

Au nombre des avantages de cette méthode, on peut citer une meilleure conversion de service par une meilleure connaissance de l'entité locale (Calvert et Lam, 1989). Cependant, cela peut être coûteux lorsqu'on considère le changement qui doit être apporté aux nœuds internes qui supportent les protocoles à complémenter. Il faut préciser que la complémentation peut être réalisée aussi bien dans un nœud d'interconnexion que dans un nœud terminal.

2.2.3 Quelques équipements d'interconnexion usuels

Plusieurs types d'équipements sont utilisés aujourd'hui pour interconnecter des réseaux et sont, en général, conçus pour résoudre le problème d'hétérogénéité à une couche précise du modèle de référence. On peut les classer en quatre grandes catégories, selon les couches concernées. Nous avons :

- les répéteurs, qui agissent au niveau 1 du modèle OSI. Ce sont des unités d'interconnexions locales;
- les ponts (bridges) sont des éléments d'interconnexion de niveau 2;
- les routeurs qui interviennent au niveau 3;
- Les passerelles, terme utilisé pour tous les systèmes agissant dans les couches supérieures à la couche 3;
- les processeurs réseau qui émergent depuis quelques années.

- *Les répéteurs*

Les répéteurs agissent au niveau physique et réalisent plutôt la prolongation ou l'adaptation d'un support (passage du câble coaxial à la fibre optique par exemple). Ils sont utilisés pour accroître la portée géographique du signal ou pour réaliser l'isolation galvanique de segments de réseau alimentés par des réseaux électriques différents. Nous ne nous intéresserons pas tellement à ces équipements puisqu'ils ne réalisent pas en tant que tel des fonctionnalités de conversion de protocole.

- *Les ponts*

Ils permettent d'interconnecter deux ou plusieurs réseaux ayant des couches physiques différentes. La frontière d'hétérogénéité se situe au niveau de la couche 1. Ils ont accès à l'adresse MAC de la trame, ce qui leur permet de réaliser la fonction d'acheminement. La table d'acheminement permet de savoir sur quel port une trame reçue doit être routée. On distingue les *ponts locaux* et les *ponts distants* (ou ponts à encapsulation) qui interconnectent des réseaux locaux via une liaison spécialisée ou un

réseau de transport. Ils réalisent la fonction d'adaptation de protocole entre le protocole local et celui du lien d'interconnexion (X25, Frame Relay, ATM, etc.).

- *Les routeurs*

Ce sont des éléments d'interconnexion de niveau 3 qui acheminent les données vers un destinataire défini par une adresse de niveau 3. Ils permettent le relayage des paquets entre deux réseaux d'espace d'adressage homogène (IP-IP, ISO-ISO etc.). Dans le cas où on se retrouve avec deux réseaux d'adressage hétérogène (IP - ISO par exemple), il faut un mécanisme de conversion particulier pour résoudre la conversion de l'espace d'adressage. C'est le rôle des passerelles inter-réseau. Ils offrent des fonctionnalités supplémentaires par rapport aux ponts entre autres : segmentation et assemblage, contrôle de congestion.

- *Les passerelles*

C'est un terme générique pour désigner toute unité d'interconnexion opérant dans les couches hautes. Les passerelles peuvent mettre en relation des systèmes totalement hétérogènes et peuvent opérer jusqu'à la couche application (passerelles applicatives). Une application d'un environnement *A* voit l'application distante de l'environnement *B* comme si cette dernière appartenait au monde *A*. La passerelle entre messagerie ISO X400 et la messagerie SMTP (Simple Mail Transfer Protocol) de l'Internet en est un exemple.

- *Les processeurs réseau*

Ce sont les équipements de la nouvelle génération, conçus pour être programmables, de manière à effectuer toutes les tâches réalisées par les autres équipements communément utilisés jusqu'à présent, offrir des services différenciés et des niveaux de QoS variés. Outre l'interconnexion physique, ils font du routage, de la conversion de protocole, de l'ingénierie de trafic, etc.

De tout ce qui précède, on peut déduire que l'interconnexion peut arriver à n'importe quelle couche et les moyens utilisés pour la réaliser peuvent être matériels (ponts, routeurs) ou logiciels (passerelles d'application). Il faut cependant constater que

parmi ces éléments d'interconnexion, certains réalisent la fonction de conversion de protocole, ce qui n'est pas le cas pour d'autres.

2.3 Conversion de protocole

Nous avions mentionné dans ce qui précède que la conversion de protocole est un moyen pour assurer la communication entre des systèmes utilisant des protocoles différents. Un convertisseur de protocole est utilisé comme médiateur entre les deux protocoles incompatibles.

Pour illustrer, considérons deux protocoles donnés *P* et *Q* initialement incompatibles. Les entités *P1* (resp. *Q1*) et *P2* (resp. *Q2*) implantent le protocole *P* (resp. *Q*) et peuvent donc communiquer directement entre elles (Figure 2.11 a et b). Notre but est d'assurer la communication entre eux via une entité que nous appelons convertisseur de protocole *C*. Nous voulons faire communiquer *P1* (resp.*Q1*) et *Q2* (resp.*P2*) via *C* (Figure 2.11 c). Le but de ce dernier est de recevoir les messages provenant de l'un des protocoles (ex: *P1*), de les transformer de sorte qu'ils puissent être compris par l'autre protocole (*Q2*) et que les informations véhiculées restent intactes (données et informations de contrôle).

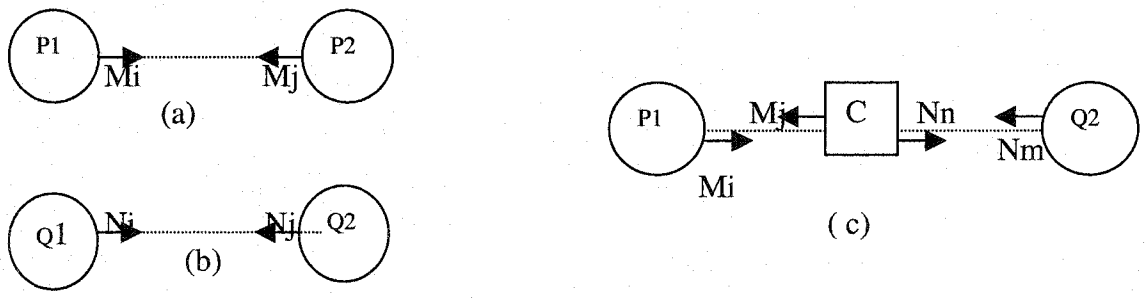


Figure 2.11 Conversion de protocole

La formulation du problème de conversion de protocole entre deux réseaux *A* et *B* peut se résumer comme suit :

Étant donné deux systèmes ou réseaux *A* et *B* utilisant des architectures différentes,

- les couches *M* de *A* et *N* de *B* utilisent différents protocoles, resp. *P* et *Q*;
- un même protocole *U* opère aux couches *M*+1 de *A* et *N*+1 de *B*;
- une spécification de service *S* a été définie pour la couche *U*.

Il faut pouvoir faire correspondre les évènements un à un d'un protocole à l'autre. La spécification de service définit les fonctionnalités minimales attendues par les utilisateurs (couches de plus haut niveau). Nous allons maintenant donner un aperçu général des différentes situations où s'applique souvent la conversion de protocole.

2.3.1 Architectures d'interconnexion impliquant la conversion

Une situation classique où la conversion de protocole intervient est schématisée à la Figure 2.12. Les protocoles au-dessus de la couche *N* utilisent les services de la couche *N*, qui appartient à l'architecture de communication *A* aussi bien au niveau de l'entité émettrice que de l'entité réceptrice. Nos deux entités communicantes obéissent donc à l'architecture *A*. Cependant, le réseau est hétérogène car il y a un segment (V) entre les deux entités qui obéit à une autre architecture *B*. Pour résoudre le problème, on peut utiliser deux convertisseurs dont le rôle serait de faire correspondre la couche *N* de l'architecture *A* à la couche *M* de l'architecture *B*. Seules les couches 1 à *M* de l'architecture *B* sont utilisées. Il y a plusieurs exemples de cette sorte de conversion en pratique. Auerbach (1989) illustre ce cas de figure avec le relais TCP-IP de IBM permettant de transporter les paquets IP en utilisant un réseau SNA.

Une autre variation du problème est celle dans laquelle la conversion a lieu à l'un des nœuds finaux, au niveau de l'une des entités communicantes. La conversion a lieu au point où la couche *N*+1 invoque la couche *N*. Dans ce cas, le convertisseur convertit les requêtes de primitive de service *N* en requêtes de primitive de service *M*. Auerbach (1989) donne un exemple de ce cas de figure où SNA serait l'architecture *A*, LU6.2 serait le service *N* et DECNET l'architecture *B* (Fig. 2.13-b).

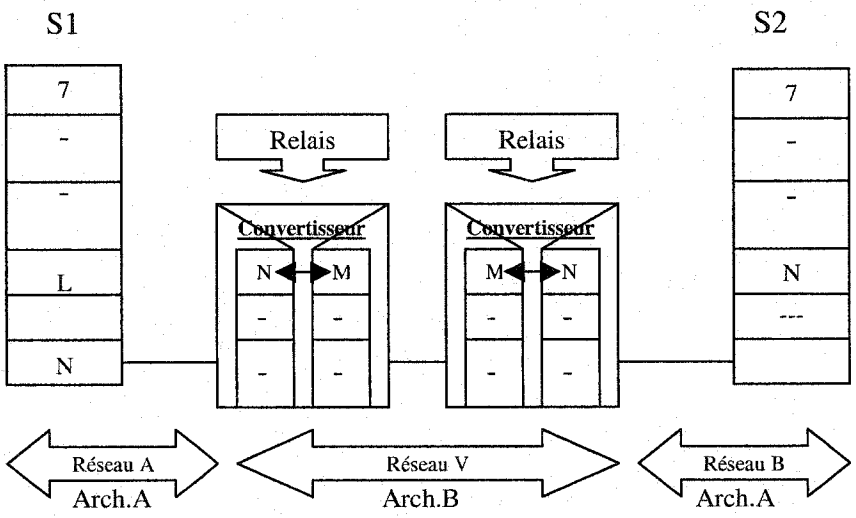


Figure 2.12 Conversion à l'émission et à la réception

La 3^{e} forme de conversion appelée substitution de protocole (*protocole substitution*) est celui de la Figure 2.13-a où la conversion a lieu dans les nœuds finaux (Green, 1986). Dans ce contexte, le réseau peut être homogène en ce qui concerne l'architecture *B*. Cependant, les protocoles de haut niveau tels que vus par l'application (*N*+1 à 7 de l'architecture *A*) ne sont pas ceux qui sont normalement utilisés avec les basses couches du réseau (couches 1-*M* de l'architecture *B*). Le convertisseur devra donc permettre ici l'interaction entre les couches *M* et *N*+1. Il s'agit plutôt d'une sorte de migration d'applications ou de services, pour réutiliser les termes de Auerbach (1989). Ce dernier illustre ce type de conversion en donnant l'exemple de l'application X-Windows sur VM/CMS. X-Windows utilise normalement des « stream sockets » avec TCP/IP, alors que VM/CMS n'utilise pas une telle interface.

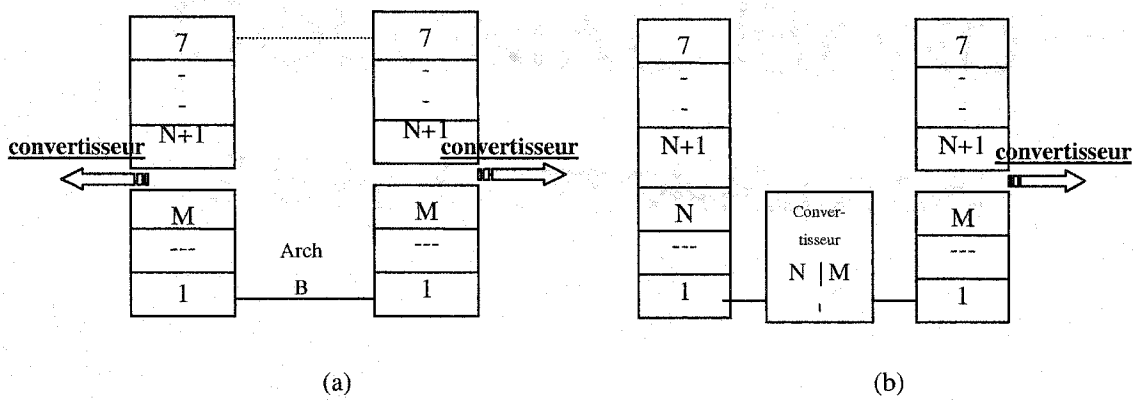


Figure 2.13 Schéma de la conversion dans les nœuds terminaux

2.3.2 Algorithmes classiques de construction de convertisseur

Dans cette section, nous faisons un survol des algorithmes proposés dans la littérature pour construire des convertisseurs de protocole. En effet, diverses méthodes ont été proposées pour construire un convertisseur approprié pour deux protocoles quelconques. La conversion peut se réaliser à différents niveaux. Elle peut être faite au niveau des services offerts, ou au niveau protocole par modification des unités de données de protocole. Les approches qui ont été proposées peuvent ainsi être classées en deux groupes : celles qui se basent sur les spécifications des entités de protocole à mettre en relation et des requis au niveau protocole, et celles qui utilisent plutôt les spécifications des services requis au niveau des protocoles. Les requis au niveau protocole concernent essentiellement les conversions au niveau des formats de protocoles de même que la synchronisation requise entre les entités de protocole en relation, tandis que les requis au niveau service traduisent les services offerts par les protocoles qui doivent être conservés une fois que le convertisseur est construit. Les approches basées sur les spécifications de service sont qualifiées de Top-Down et les autres qui se fondent sur les requis au niveau protocoles sont nommes Bottom-Up.

La plupart des modèles proposés dans la littérature pour la construction de convertisseur se basent sur les spécifications formelles (Yao et Liu, 92; Tao et Goosens, 1992 ; Calvert et Lam, 1989 ; Lam 1988 ; Okumura, 1986 ; Chang et Liu , 1990). Nous

ne traiterons pas de ces modèles puisque notre travail ne s'inscrit pas réellement dans cette optique.

2.3.3 Vers une implémentation concrète de la conversion

Certains chercheurs ont abordé le problème de l'interopérabilité sous un angle moins abstrait en étudiant des cas concrets de conversion de protocoles (Green, 1986; François et Potocki, 1983; Sy et al., 1987; Takei et al., 2001) ou des concepts importants touchant à l'interconnexion des systèmes comme l'adressage, le routage, etc. (Sunshine, 1990; Svobodova, 1990).

Green (1986), après avoir discuté brièvement de certaines situations où intervient la conversion de protocole, a proposé une méthodologie de résolution de l'interopérabilité sous la forme d'algorithme permettant de choisir entre la méthode de complémentation et celle de la conversion proprement dite, et qui utilise les requis de l'application. De plus, il introduit certaines notions comme "fonctions atomiques de protocole" qui désignent l'ensemble des services fournis par les deux architectures (interfaces, contrôles et protocoles paires) au point de conversion, "hard mismatch" et "soft mismatch" pour indiquer le degré de disparité entre les protocoles.

Bien avant, Gien et Zimmermann (1979) avaient noté un ensemble de techniques qui constituent la base des architectures de réseaux comme le multiplexage, la commutation, la cascade, l'encapsulation, la hiérarchisation en couches. Ils mentionnent quelques principes que les réseaux à interconnecter doivent respecter à savoir:

- les niveaux de service équivalents;
- un ensemble de propriétés rendant l'interconnexion viable, comme la l'enchaînement des services, le multiplexage et l'espace d'adressage global.

Par le passé, Cerf et Kirstein (1978) s'étaient intéressés à l'interconnexion des PSDN (Public Switch Data Network); ils avaient énoncé un ensemble de concepts techniques associés et comparé les services datagramme et circuit virtuel en se basant principalement sur ARPANET et X25-X75. Au-delà des travaux ayant discuté de

certains aspects particuliers de l'interconnexion, on peut citer aussi des recherches qui se sont focalisées sur des protocoles d'une couche précise du modèle OSI. Nous donnons, dans la suite, un bref aperçu de travaux en rapport avec la conversion de protocole à chaque couche OSI.

Conversion aux couches 3 et 4

Deux grandes catégories de service sont présentes à ces couches: le service connecté et le service non-connecté ou datagramme. Certains chercheurs (Boggs et al., 1980) ont concentré leurs études sur le mode datagramme en présentant IP (avec TCP) comme un protocole simple mais assez puissant et flexible qui permet de supporter plusieurs réseaux à commutation de paquets en tolérant des variations dans la taille, les défaillances de transmission (détection et correction), la fragmentation, le contrôle de flot et d'erreur de bout en bout, etc. Piscitello et al. (1986) discutent de la question de la conversion entre service datagramme et service connecté et indique les raisons pour lesquelles le mode non connecté est plus facile à supporter en pratique: les datagrammes offrent moins de services que les circuits virtuels. Plusieurs réseaux offrent éventuellement un service VC, mais d'autres le diffèrent au protocole de la couche 4. Kawa et Bochman, (1987) discutent eux aussi de l'interconnexion à la couche 3 des deux types de services (datagramme et circuit virtuel) en considérant le contexte de l'OSI et indiquent qu'une sorte de complémentation est nécessaire pour assurer une conformité au-dessus de la couche 3 de chaque réseau individuel.

Auerbach (1989) a concentré ses travaux sur la couche transport et a remarqué que les protocoles de transport actuels utilisés dans les réseaux et employés par les programmes peuvent être classés en un nombre restreint de catégories, nommés types d'abstraction de transport. Il a proposé une librairie extensible de sous-routines qui réalise une forme de conversion des protocoles de couche transport nommé "Transport Abstraction Converter Toolkit". Il a aussi mentionné quelques observations pertinentes:

- La conversion de protocole aux nœuds finaux est un cas important de la conversion de protocole. De plus, les outils utilisés pour réaliser la conversion dans un nœud final peuvent aussi être utilisés pour construire des relais.
- La conversion de protocole au niveau transport (et au-delà) peut être réalisée avec succès sans une modification critique du code du système. Une librairie est suffisante.

Conversion à la couche 5

Quelques recherches ont porté sur des protocoles de la couche session. Sy et al. (1987) présente une méthode de conversion entre la couche session OSI et l'application LU 6.2 de SNA qui offre une communication entre deux entités (SNA LU-LU session). L'approche utilisée se résume en quatre étapes: identification des fonctions communes à OSI et SNA, définition de la procédure de conversion de protocole entre OSI et SNA, établissement de la méthode de conversion des adresses, la gestion des exceptions. (Tajiwara et al., 1982) présentent une méthode de conversion de protocole à la couche 5 entre l'architecture DCNA (Data Communication Network Architecture) et SNA (Systems Network Architecture), basée sur la correspondance entre les fonctions d'établissement et terminaison de connexion, de transfert de données de l'un ou l'autre des protocoles. Tout récemment, Takei et al. (2001) ont proposé une approche pour construire un relais réalisant une conversion entre deux protocoles couramment utilisés dans les sessions multimédia: H.323 et SIP. Ce sont deux protocoles utilisés dans la téléphonie IP.

Conversion de protocole aux hautes couches

Aux couches Présentation et Application, on parle de *passerelles d'application* ou *conversion de protocole complète*. Les entités communicantes sont les usagers au niveau application. De tels convertisseurs de protocoles n'agissent pas entre de larges réseaux, mais plutôt au nœud terminal. Il faut noter que la littérature sur la manière dont les conversions de protocoles sont réalisées dans les nœuds terminaux n'est pas développée ou presque inexistante.

2.3.4 La conversion de protocole et le « network processing »

L'interopérabilité entre réseaux a suscité ces dernières années un grand intérêt de la part des concepteurs de circuits intégrés. Les objectifs actuels tendent vers la construction de processeurs réseaux dont les fonctionnalités dépassent le simple acheminement des paquets basés sur une consultation d'un champ de l'entête. Il s'agit maintenant offrir avec les processeurs réseau des services différenciés, avec différents niveaux de QoS (qualité de service), et faire du routage de niveau 4 ou même de niveau application. Ces circuits hautement intégrés sont conçus de manière à pouvoir réaliser une série de tâches assez complexes qui ne sont pas possibles avec les équipements réseau traditionnels. Williams (2001) résume les principales fonctionnalités de ces équipements comme suit :

- Reconnaissance de protocole et classification : les paquets sont identifiés sur la base de certaines informations contenues dans leurs entêtes;
- Fragmentation et assemblage : les paquets sont désassemblés, étudiés et assemblés pour leur acheminement;
- Mise en file d'attente et contrôle d'accès : une fois les paquets classifiés, ils sont placés dans une file appropriée, il peut y avoir des opérations de filtrage.
- Ingénierie de trafic : les données sont acheminées vers le port de sortie approprié (câble ou fibre optique pour assurer que les contraintes de délai de l'application sont respectées);
- QoS à offrir à certains types de paquets;
- Modification de paquets : les champs d'un paquet peuvent être modifiés pour mettre à jour ou insérer de nouvelles informations.
- Détection et correction d'erreur.

Welfeld (2001) schématise l'architecture générale d'un processeur réseau en quatre modules principaux (Figure 2.14), s'occupant chacun de la classification, de la modification, du cryptage et de la gestion du trafic respectivement. La majeure partie des recherches actuelles touche au module de classification de paquets. La classification

nécessite une inspection des paquets et peut se réaliser sur un seul ou plusieurs champs de l'entête. Elle peut même nécessiter d'examiner en profondeur le paquet (dans la charge utile du PDU) et de prendre des décisions sur la base de valeurs de centaines de caractères. C'est le cas lorsqu'il faut identifier une URL ou un "cookie" dans une trame.

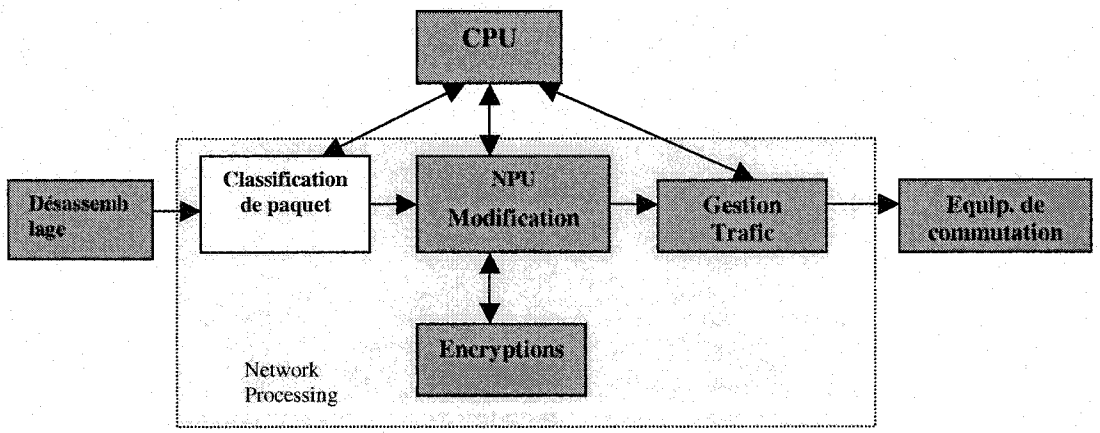


Figure 2.14 Principaux modules d'un processeur réseau

Welfeld (2001) a souligné le fait que la classification avancée de paquets est une tâche compliquée qui nécessite d'inspecter au delà des couches réseau et transport, i.e. jusqu'aux couches session et application. Plusieurs algorithmes ont été proposés pour améliorer la classification et diminuer le temps de recherche. Gupta et McKeown (1999, 2001) ont proposé des algorithmes basés sur des méthodes géométriques et heuristiques. Bailey et al. (1994), Feldmann et Muthukrishnan (2000), Baboescu et Varghese (2001); Srinivasan et al. (1999) ont aussi proposé divers algorithmes de classification améliorant à chaque fois des aspects non encore explorés.

Après la classification intervient la modification des PDU. En modifiant les PDU, ils réalisent une conversion de protocole. Il faut cependant noter que, dans le cas présent, le protocole à convertir n'est parfois pas connu d'avance du processeur, car ce dernier est conçu pour en supporter plusieurs, contrairement aux anciens équipements qui, généralement, étaient conçus pour un nombre restreint de protocoles spécifiques. De plus, les processeurs sont placés à la frontière des réseaux, aux endroits où des trafics

provenant de plusieurs usagers, de plusieurs sous-réseaux viennent s'agréger. La littérature ne nous a pas renseignés pour l'instant sur des travaux relatifs à la conversion des protocoles dans un environnement où les protocoles à convertir ne sont pas connus d'avance. Lorsqu'on réalise que ces dispositifs devront opérer dans des environnements multiprotocoles, la question qui surgit automatiquement est de savoir si tous les protocoles seront supportés, c'est à dire pourront être convertibles.

2.4 Synthèses et observations

Cette revue de littérature nous a renseignés sur les architectures d'interconnexion, les approches utilisées pour réaliser l'interopérabilité entre protocoles et les travaux théoriques relatifs à la conversion de protocole. Certaines recherches ont proposé des méthodes formelles de spécification de convertisseur. Un point commun à toutes les approches fondées sur des méthodes de description formelles: elles utilisent un modèle mathématique et des modélisations sous la forme de machines à états finis communicants pour déterminer s'il est possible de construire un convertisseur à partir des protocoles initiaux ou des spécifications de service requis. D'une part, elles demandent un temps de calcul non négligeable, surtout lorsque le nombre d'états devient important (problème d'explosion combinatoire) et ont surtout pour objectif de valider la spécification d'un convertisseur qui sera possiblement implanté dans un relais.

En pratique, le problème de la conversion peut toucher plusieurs couches dans une architecture. Pour cela, la plupart des approches considérées dans le domaine de la recherche utilisent une abstraction générale du problème, en ce sens que plusieurs aspects sont ignorés, incluant l'adressage, le routage et la gestion de réseau. Nous avons remarqué que certaines études ont touché le problème d'hétérogénéité d'une façon plus pratique en proposant des méthodes et outils pour faire communiquer des protocoles (Auerbach, 1989; Takei et al., 2001). D'autres ont porté sur la conversion d'une paire de protocoles précise. Certains problèmes rencontrés lorsqu'on interconnecte des réseaux ou des systèmes utilisant différents protocoles comme les problèmes d'adressage entre

différents domaines (OSI et non-OSI par exemple) ont préoccupé certains chercheurs (Svobodova et al., 1990).

Pour finir, nous avons constaté que la conversion de protocole est une fonctionnalité importante dans le domaine de la nouvelle génération d'équipements réseau nommés "*processeurs réseaux*", qui est en plein essor. Ces processeurs ont pour objectif d'assumer un ensemble de fonctionnalités incluant celles des équipements réseaux actuels. Ils devront faire la classification, la modification des paquets, la gestion de trafic, le cryptage des données, etc. Les récentes études se sont concentrées sur la classification de paquets qui est la première étape. Après avoir reconnu le protocole d'un paquet, la seconde étape va consister à convertir ce dernier de sorte que l'information ne soit pas déformée et soit acheminée jusqu'à l'application ou l'usager destinataire. Ces relais multiprotocoles n'auront pas forcément une connaissance de l'application pour laquelle est destiné le paquet. En d'autres termes, si un paquet formaté avec un protocole P arrive et doit être acheminé vers un réseau utilisant un protocole Q, la conversion de protocole entre P et Q doit se faire, bien que l'on ait aucune connaissance particulière sur les requis de l'application utilisant Q à l'autre bout. On se rappelle que Green (1986) avait proposé une méthodologie générale de prédiction de la convertibilité des protocoles, connaissant les requis de l'application qui les utilise. Il devient maintenant important d'identifier une technique qui permettra de décider de la convertibilité en faisant abstraction de l'application qui utilise les protocoles et de la caractériser en précisant ses limites.

CHAPITRE III
MODÈLE D'UN CONVERTISSEUR GÉNÉRIQUE DE PROTOCOLES

Dans le chapitre 1, nous avions précisé les objectifs de notre recherche et présenté au chapitre 2, de façon générale, les travaux reliés, trouvés dans la littérature. Dans ce chapitre, nous allons dans un premier temps identifier des propriétés communes aux protocoles en général, puis les critères qui nous aideront à en faire une classification. Par la suite, nous identifierons les obstacles à la convertibilité et déduirons quelques critères nécessaires pour que deux protocoles quelconques soient convertibles. Nous proposerons par la suite une méthode pour identifier la convertibilité et terminerons avec un schéma de convertisseur générique, flexible, pour supporter la conversion de protocoles quelconques satisfaisant à des critères précis que nous préciserons.

3.1 Concepts de base

Il convient de rappeler certaines notions ou expressions que nous allons utiliser fréquemment dans ce chapitre. Il s'agit principalement des notions de *protocole orienté bit* et *protocole orienté caractère*, *service datagramme* et *circuit virtuel, l'adressage et ses différents types, l'accusé de réception* et enfin *la fragmentation.*

Protocole orienté bit et protocole orienté caractère

Les protocoles peuvent être classés en deux grandes catégories selon le format de leurs informations de contrôle (entêtes) : ceux dont les champs de l'entête ont un format et une longueur fixes (ou variables avec une longueur maximale connue), et qui peuvent s'écrire en binaire sur un certain nombre de bits, appelés *protocole orienté bit,* et ceux dont les entêtes s'écrivent plutôt en ASCII (ou un autre format de représentation) et dont la longueur n'est en général pas fixe. Ces derniers, que l'on nomme *protocole orienté caractère,* opèrent pour la plupart aux hautes couches et, dans leurs spécifications, on

donne plutôt le contenu des messages envoyés ou reçus (au lieu du format de paquet), les commandes et réponses ainsi que leurs paramètres. IP est un exemple de protocole orienté bit, tandis que HTTP est orienté caractère.

Service orienté connexion et service datagramme

Dans un *service non-connecté* (ou *datagramme*), chaque paquet est en général transmis avec l'adresse de destination et est routé individuellement, de sorte qu'il n'y a pas de garantie que les paquets arrivent à destination dans le même ordre où ils ont été envoyés. La machine réceptrice reçoit les données sans envoyer d'accusé de réception à la machine émettrice. UDP est un protocole offrant un service datagramme. Ce service est propice aux applications qui peuvent tolérer des pertes.

Dans un *service orienté connexion* (ou *circuit virtuel*), une connexion est d'abord établie entre les nœuds communicants. La source A informe le réseau de son intention de communiquer avec B, le réseau envoie la requête à B qui peut l'accepter ou la refuser. S'il l'accepte, la connexion est établie et les données sont ensuite envoyées. Les caractéristiques d'un service orienté connexion sont :

- les paquets arrivent dans l'ordre où ils ont été envoyés;
- ils suivent tous le même chemin préalablement établi lors de l'établissement de la connexion;
- s'il y a congestion entre temps, les connexions futures sont refusées;
- en général, la machine réceptrice envoie des accusés de réception lors de la communication.

Le protocole TCP est une bonne illustration du service orienté connexion. Les applications qui ne tolèrent pas des pertes de données, comme les transferts de fichiers, vont utiliser un service orienté connexion.

Accusé de réception

En général, dans un service orienté connexion, le destinataire envoie un message (soit un paquet de contrôle ou valeur d'un champ de l'entête) à l'émetteur pour indiquer si

le paquet a été bien reçu.

Adressage

L'adressage permet d'indiquer la localisation d'une ressource. D'une manière générale, on peut dire que les mécanismes d'adressage sont présents à toutes les couches du modèle de référence. Comme on peut le remarquer, aux plus basses couches, on a des adresses physiques (adresse MAC dans les LANs). À la couche réseau, on parle d'adresse logique qui existe sous diverses formes selon l'architecture dans laquelle on se retrouve. Dans le monde TCP/IP par exemple, on a les adresses IP; dans X.25, elles s'écrivent au format X.121. Decnet et Appletalk utilisent un adressage qui réfère à un ou plusieurs réseaux, mais n'est pas unique. Aux plus hautes couches, on a aussi des mécanismes d'adressage basés en général sur les numéros de port des applications à la couche Transport, ou même des noms de domaine spécifiés par des adresses URL (ex. HTTP).

Fragmentation

Dans nombre de cas, une longueur maximale est prévue pour la charge utile des paquets. Dès lors, quand la longueur d'une trame dépasse la longueur maximale autorisée pour le paquet (nommé *MTU : Maximum Transfer Unit*), il faut le fragmenter. La fragmentation implique d'un autre coté l'assemblage et, pour permettre la récupération des données fragmentées, des champs sont prévues dans l'entête du paquet pour indiquer le numéro de séquence du paquet, la longueur totale du datagramme initial et parfois l'identificateur de flot.

3.2 Survol et analyse de protocoles usuels

Notre objectif dans cette section est de considérer un ensemble de protocoles courants et de les analyser en vue d'identifier des critères de classification de protocoles. Par la suite, nous ferons une étude comparative de quelques paires pour identifier les

ressemblances, les points communs qui peuvent faciliter la conversion et éventuellement ceux qui peuvent la rendre difficile.

3.2.1 Choix des protocoles

Compte tenu du nombre de protocoles existants (plusieurs centaines), il convient de choisir une bonne méthodologie de sélection des protocoles à considérer dans notre survol. Nous voudrions que notre ensemble soit assez représentatif des différentes catégories de protocoles, qu'il nous permette d'avoir une vue générale (pas forcément exhaustive) des propriétés qui les caractérisent aussi bien aux basses qu'aux hautes couches du modèle de référence, et ceci indépendamment de l'architecture considérée. Pour cela, notre principe de sélection des protocoles sera le suivant:

- considérer différentes architectures de protocoles ;
- dégager dans chaque architecture un ensemble de protocoles offrant des services différents et opérant à des couches différentes (respectivement au modèle OSI).

Le survol consistera à identifier, pour chaque protocole considéré, les fonctions principales (services de base offerts), les mécanismes de fonctionnement, la couche concernée puis le format des paquets ou des messages. Au terme de cette analyse, nous pourrons établir un ensemble de facteurs qui aideront à la classification des protocoles.

Les architectures considérées sont: la suite la plus populaire TCP/IP, la suite ISO (normalisation), les architectures propriétaires Decnet, Appletalk. Nous allons aussi considérer d'autres suites de protocoles comme X25, H.323, et des protocoles LAN. En bref, les protocoles étudiés sont : IPv4, IPv6, TCP, UDP, SMTP, HTTP dans TCP/IP; ISO-IP, ISO-TP (0-4), ISO-SP, ISO-X400 dans ISO; DRP, NSP, SCP de Decnet, DDP, ASP, ADSP de l'architecture Appletalk et enfin X25, H323 (RTP, RTCP, H.225, H.261), MPLS, ATM et quelques protocoles LAN comme Ethernet, FDDI, Token Ring. Les résultats de ce survol figurent en annexe pour certains protocoles.

3.2.2 Analyse du survol

Il ressort de cette analyse que les propriétés et les types de services rencontrés au niveau des protocoles sont nombreux et variés, de sorte qu'il est difficile d'en faire une liste exhaustive. Cependant, nous pouvons mentionner quelques points clés. Tout d'abord, considérant les protocoles orientés bit, des fonctionnalités communes peuvent être notées au niveau des protocoles LAN par exemple (niveau liaison): ils disposent souvent de champs d'adressage physique, de champs pour le contrôle d'erreur des données de la trame (*checksum*), un champ pour les données utiles transportées, et souvent d'autres champs de contrôle dépendamment des services supplémentaires offerts. On retrouve souvent les deux modes de fonctionnement à savoir : le mode sans connexion ou le mode avec connexion. Certains offrent des mécanismes d'accusé de réception, inclus soit dans le paquet de données, soit dans un paquet de contrôle. On notera souvent des informations de fragmentation, de contrôle de flot et parfois de contrôle de congestion. Certains champs sont quasi-obligatoires, tandis que d'autres sont optionnels. Il convient aussi de remarquer que tout ce que nous venons de mentionner concerne essentiellement les protocoles opérant dans les couches basses (jusqu'à la couche transport).

Concernant les protocoles orientés caractère, un format de message clair et précis n'est souvent pas défini. Ils opèrent souvent par envoi de commandes et réponses, ayant des paramètres ou des codes. Les fonctionnalités et options des commandes, les formats de messages diffèrent considérablement d'un protocole à l'autre, de sorte qu'il nous est difficile de trouver des points communs pour des protocoles utilisés à divers fins dans les hautes couches. Il faut ajouter qu'au-delà des protocoles orientés bit et orientés caractère, nous avons aussi des protocoles qui sont en réalité des algorithmes comme les protocoles de routage (RIP, OSPF). Ces derniers ne feront pas partie de notre étude par la suite.

3.2.3 Critères de classification

Nous pouvons résumer quelques critères de classification déduits de notre survol comme suit : services principaux offerts, nature du protocole (orienté bit ou caractère), adressage, service orienté connexion ou datagramme, fragmentation, mécanisme d'accusé de réception (ACK), contrôle de flux, contrôle d'erreur des données transportées (checksum), format unique de paquets ou plusieurs types de paquets, limitation de taille des paquets (MTU), vitesse de transmission (physique), mode full duplex ou semi-duplex, etc. Cependant, nous avons remarqué que les ressemblances se font plus grandes chez les protocoles opérant à une même couche du modèle de référence. Ceci voudrait signifier que la classification des protocoles devrait se faire d'abord selon les services qu'ils offrent.

3.2.4 Étude comparative de quelques protocoles

Nous considérons quelques paires de protocoles à différentes couches et montrons que des informations analogues se retrouvent bien souvent chez des protocoles opérant à la même couche et offrant des services semblables.

Protocoles de la couche réseau

On peut remarquer que IPv4, IPv6 (TCP/IP) et ISO-IP (ISO) offrent quasiment les mêmes services: adressage réseau, fragmentation et assemblage, contrôle de l'entête, options etc. Ils opèrent selon le même mode datagramme. L'étude des formats des paquets de ces protocoles (cf. annexe A) montre qu'il est aisé de faire une correspondance directe entre la plupart des champs de l'entête de l'un ou l'autre de ces protocoles. On peut croire qu'une convertibilité est possible entre ces trois protocoles.

Protocoles de la couche liaison de données

Les protocoles Ethernet, Token Ring et FDDI sont spécifiés pour les réseaux LAN. Ethernet (IEEE802.3) est basé sur une topologie de bus et une méthode d'accès CSMA/CD. Dans Token Ring, les stations sont connectées en anneau et chacun peut

entendre son voisin émettre. La permission d'émettre est gérée par un jeton. FDDI, elle, est une technologie à100Mbps qui utilise un jeton chronométré et un double anneau. Bien que chacune de ces technologies possède des méthodes d'accès différents à la liaison physique, force est de constater qu'ils offrent tous des services similaires: ils permettent en effet l'acheminement des trames sur la liaison physique, fournissent l'adressage MAC pour les équipements communicants, offrent souvent un contrôle d'erreur et un contrôle d'accès à la liaison. Nous avons tenté de faire une correspondance entre les informations d'entête de trame *Ethernet(802.3)* et *Token Ring*. La Figure 3.1 illustre ces correspondances.

Token Ring	**Ethernet**
Start DEL	Start of Frame del
Access control byte	---------
Frame control byte	---------
Destination address 6 bytes	Destination address 6 bytes
Source address 6 bytes	Source address 6 bytes
Information (LLC or MAC)	Data Information - Pad
FCS	FCS
Route information 0-30 bytes	---------
---------	Length / type 4 bytes
Frame status 1 byte	------

Figure 3.1 Correspondance entre PCI de Ethernet et Token Ring

En étendant notre analyse à d'autres protocoles de la couche liaison qui ne sont pas spécifiques aux LAN comme HDLC, PPP, on se rend compte que les mêmes éléments de service reviennent. Les champs *délimiteur, Address, Checksum, Control et Payload* en sont des exemples. Cela confirme la remarque faite précédemment: les mêmes informations se retrouvent sous différents formats dans des protocoles offrant des services similaires. La Figure 3.2 montre les informations contenues dans les entêtes de PPP et HDLC.

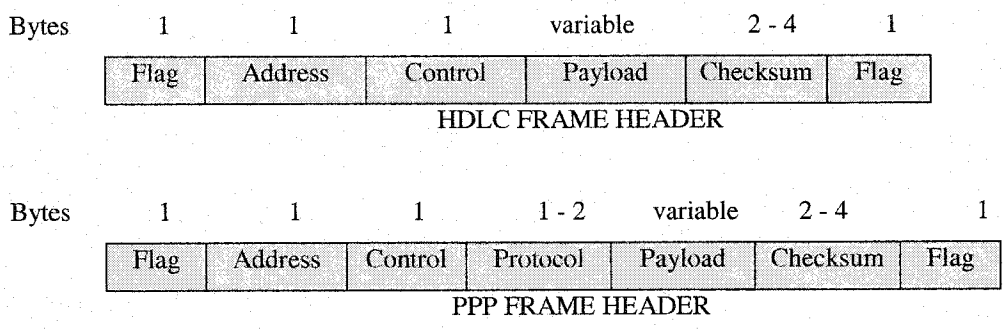


Figure 3.2 Informations de contrôle des protocoles HDLC et PPP

Protocoles de la couche transport

Considérons à présent quelques protocoles opérant à la couche transport du modèle de référence. Leur fonction principale est d'assurer le transport de bout en bout entre les nœuds communicants. Nous choisissons TCP du monde Internet, NSP de Decnet et ADSP de Appletalk. On constatera que TCP et NSP offrent quasiment les mêmes services au protocole de la couche réseau: connexion virtuelle, transport de bout en bout, fragmentation et assemblage de messages, contrôle d'erreur, contrôle de flux. Une comparaison entre les informations contenues dans les entêtes montre qu'ils possèdent plusieurs champs analogues, pour lesquels on peut faire une correspondance directe. Il s'agit en particulier des champs contenant les informations d'adressage, le numéro d'accusé de réception, le numéro de séquence, de même que les informations de contrôle de flot. ADSP est souvent considéré comme à mi-chemin entre les couches transport et session. Il fournit un canal d'acheminement des données dans l'ordre avec contrôle de flux, un peu comme TCP. Le Tableau 3.1 indique la correspondance entre les champs de l'entête de chacun de ces protocoles.

Comme on peut le constater, outre les champs de service communs, il existe aussi des champs spécifiques à chaque protocole qui traduisent les services supplémentaires qu'offre ce dernier. Il est évident que, lors d'une éventuelle conversion, ces éléments de service particuliers à chaque protocole ne pourront être supportés.

Tableau 3.1 Correspondance entre éléments de service de TCP, NSP et ADSP

TCP	NSP	ADSP
destination port	*Destination link address*	Destination connection ID
Source port	*Source link address*	Source connection ID
Acknowledge number	*Acknowledge number*	Receive sequence number
Window	*Acknowledge other*	Receive window size
Sequence number	*Segment number*	Send sequence number
------	*Flow control { Scp_req,}*	
Urgent pointer	------	------
Option	------	------
Flags	------	Control flag, Ack request flag,
Offset	------	------
Data	*Data*	*Data*
------	*BOM/EOM* début/fin de message)	End of message flag
------		Attention flag

Protocoles utilisés dans la messagerie électronique (couche application)

Le courrier électronique est largement utilisé dans les communications. Nous allons comparer trois protocoles de messagerie: SMTP (MIME), X.400 et Mail-11.

Le protocole SMTP est l'un des protocoles couramment utilisés dans le cadre du transfert de courrier entre serveurs. Il est simplement basé sur l'envoi de chaînes ASCII et le retour de codes d'erreur. SMTP fonctionne en mode synchrone, i.e. que le client envoie une commande et attend la réponse du serveur (qui ressemble à un accusé de réception) avant de continuer. Il nécessite un protocole de transport fiable garantissant l'acheminement des données dans l'ordre, comme TCP pour la connexion entre les machines. SMTP est l'un des protocoles compatibles avec les spécifications de la MIME (Messagerie Internet, RFC822).

Quant à la norme X.400, elle présente des services et des protocoles pour le transfert du courrier électronique entre des machines basées sur la structure de la poste. Le MHS (Message Handling System) désigne le système global de messagerie et deux protocoles P1 et P3 sont utilisés pour le transfert du courrier entre les entités de transport

MTA (Message Transfert Agent). Pour les courriers, un protocole P2 est utilisé en plus entre les agents usagers.

Une tentative de correspondance entre les messages du système de messagerie interpersonnel de X.400 et ceux du monde SMTP montre une certaine similitude au niveau de certains éléments de service. Dans le Tableau 3.2, figurent une liste de correspondances basiques qui peuvent être faites entre les informations contenues dans l'entête du message SMTP et celle de X.400 (extrait des RFC 1506 et 2162). Des correspondances similaires existent aussi entre le système de messagerie temps-réel utilisés dans l'architecture Decnet Phase IV (et Decnet/OSI), nommé Mail-11 ou VMSmail, et X.400. Des protocoles P1 et P2 semblables à ceux définis dans X.400 existent aussi dans Mail-11. La RFC2162 indique les correspondances entre les champs de l'enveloppe (P1) et de l'entête (P2) Mail-11 et ceux de l'entête X.400. Cependant, certains champs de X.400 n'existant pas dans Mail-11 ne peuvent donc être supportés dans la conversion.

Tableau 3.2 Correspondance entre éléments de service de protocoles de messagerie

X.400 service elements	Mail-11 service elements	RFC822/MIME
P1.Originator	P1-11.'From:'	822-MTS.Originator
P1.Primary Recipients	P1-11.'To:'	822-MTS.Primary Recipient
P2.Originator	P2-11.'From:'	822.'From:'
P2.Primary Recipients	P2-11.'To:'	822.'To:'
P2.Copy Recipients	P2-11.'CC:'	822.'Cc:'
P2.Submission Time St.	P2-11.Date	822.'Date:'
P2.Subject	P2-11.'Subj:'	822.'Subject:'

Les principaux éléments de service contenus dans l'entête Mail-11 sont supportés dans la conversion vers X.400. Cependant, il n'en est pas de même lorsqu'on passe de X.400 à Mail-11. Les RFC 1327 et 2162 donnent de plus amples détails sur l'interconnexion entre les deux systèmes de messagerie considérés. De même, on voit

qu'on peut aussi établir une correspondance entre la messagerie d'Internet spécifié dans la RFC822 appelée communément MIME et Mail-11 (Tableau 3.2).

3.2.5 Observations et synthèse

D'un point de vue global, on peut remarquer que les protocoles offrant des services similaires ont des mécanismes et formats (de paquets) similaires. De même, des protocoles qui offrent des services radicalement différents vont souvent avoir des formats de paquets totalement différents, et les informations transportées par l'un ou l'autre n'auront souvent aucun rapport.

Nous pouvons ainsi conclure que la notion de classification des protocoles est étroitement liée à la notion de service et par conséquent à l'architecture OSI. C'est d'ailleurs le premier critère qui va permettre d'identifier la convertibilité. Dans la section précédente, nous avons pu identifier des points communs et établir la correspondance entre les informations de contrôle de protocoles offrant des services similaires. L'ensemble des services communs devient alors un indicateur assez important lors de l'identification de la convertibilité des protocoles. Certains chercheurs (Lam, 1988; Yao et Liu, 1992; Green, 1986) l'ont déjà mentionné plus ou moins indirectement. Dans la liste des méthodes formelles qui ont été proposées pour générer une spécification de convertisseur de protocole, Calvert et Lam (1989) suggéraient de construire le convertisseur à partir des spécifications des protocoles P et Q et du service S qu'ils doivent offrir (chap.2). Bochmann (1990) a aussi déjà noté que le problème d'interconnexion de protocoles se résout plus efficacement en considérant les services de communication des systèmes à interconnecter et qu'on peut avoir plusieurs convertisseurs de protocole des mêmes protocoles initiaux si on considère différents services à offrir (chap.2).

3.3 Convertibilité des protocoles

Nous allons maintenant préciser les limites de la convertibilité et mentionner quelques obstacles qui entravent la conversion des protocoles. Ceci nous amènera à identifier les critères que doivent remplir deux protocoles quelconques pour qu'ils soient convertibles. Nous proposons par la suite notre algorithme de convertibilité suivi d'un exemple d'application.

3.3.1 Limites de la convertibilité

Une question à laquelle nous allons tenter de trouver une réponse est de savoir s'il est possible de construire un convertisseur universel de protocoles, i.e. étant donné deux protocoles quelconques P1 et P2, est-il possible d'envisager une convertibilité quelconque entre eux? La réponse peut paraître assez évidente si nous considérons les observations faites dans la section précédente (3.2.5). On pourrait aussi considérer des exemples concrets pour répondre à cette question.

Considérons un sous-ensemble des protocoles intervenant dans un système offrant des services de transfert de fichiers entre deux machines par exemple, illustré à la Figure 3.3. Limitons-nous à l'ensemble des quatre protocoles que sont FTP, TCP, IP, CLNP. La question précédente reviendrait alors à se demander s'il est possible d'envisager la convertibilité entre deux protocoles quelconques de l'ensemble {FTP, TCP, IP, CLNP}. Si nous prenons tout simplement la paire (FTP, IP), la réponse est immédiate et apparaît intuitive. Nous savons que FTP est un protocole orienté caractère qui offre un service de transfert de fichiers, et opère au-dessus d'un protocole de transport qui garantit un transport fiable des données comme TCP. Celui-ci utilise les services d'un autre protocole de la couche réseau (comme IP) chargé de l'adressage et du routage des données de la source jusqu'à la destination. IP dispose des mécanismes pour adresser les nœuds communicants et des informations permettant la fragmentation et l'assemblage des paquets, tandis que FTP fournit plutôt des informations concernant le fichier à transférer, la longueur des données, le numéro de page, etc. (RFC959). Il est

évident qu'il n'y a aucun rapport entre les informations de contrôle des deux protocoles, ils n'ont aucun service commun et il n'y a pas de raison d'envisager une convertibilité entre eux. De la même manière, il ne serait pas raisonnable de vouloir convertir un protocole qui fournit des services de messagerie comme SMTP en un protocole LAN de la couche liaison de données comme Ethernet. On se rend compte avec ces exemples triviaux qu'il n'est pas raisonnable d'envisager la conversion d'un protocole quelconque X en un autre protocole quelconque Y, d'où l'importance d'identifier les critères de convertibilité des protocoles.

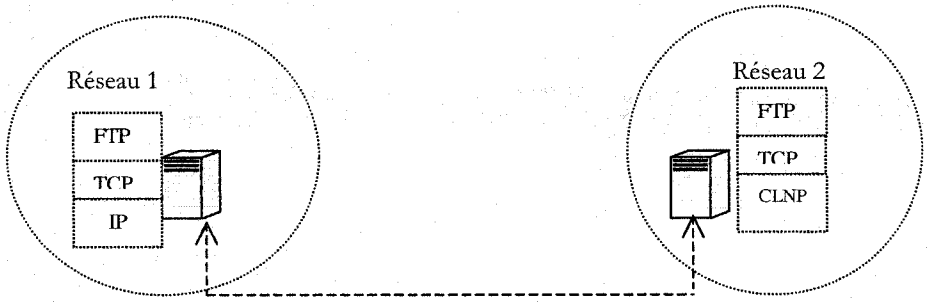


Figure 3.3 Protocoles intervenant dans le transfert de fichiers

3.3.2 Quelques obstacles à la convertibilité

Après avoir remarqué que la conversion de protocole n'est réalisable que sous certaines conditions, il convient de les identifier. Mais avant, nous indiquons comment certaines propriétés des protocoles peuvent constituer des obstacles à leur convertibilité.

3.3.2.1 Service datagramme versus service orienté connexion

Nous montrons ici que la différence entre modes de fonctionnement, datagramme et circuit virtuel, est un obstacle à la convertibilité. Soient *Pd* et *Pc* deux protocoles, l'un opérant en mode datagramme dans un réseau A (*Pd*) et l'autre en mode circuit virtuel dans un réseau B (*Pc*). En général, dans le service datagramme, les paquets sont délivrés de manière indépendante et disposent dans leur entête de toute l'information d'adressage nécessaire pour se rendre à la destination. En mode connecté, il y a plutôt un numéro de canal pour la connexion à utiliser, et tous les paquets du même

flot suivent la même route. On suppose donc que les paquets *Pc* contiennent un numéro de voie logique indiquant le circuit virtuel qu'ils utilisent, alors que ceux de *Pd* ont les adresses des nœuds source et destination.

- Premier problème: difficulté de correspondance entre les adresses de nœuds de *Pd* (adresse IP par exemple) et la connexion (comme dans X.25) à utiliser pour les paquets *Pc,* connexion dont le numéro n'a aucun rapport avec les adresses de nœud destination?
- Deuxième problème: le mode connecté (ou circuit virtuel) rajoute des services supplémentaires par rapport au mode datagramme, comme la fiabilité et la délivrance des paquets dans l'ordre. Des paquets provenant de *Pd* arrivent dans un ordre aléatoire et, s'ils doivent être acheminés directement, *Pc* les recevra aussi dans un ordre aléatoire. De plus, la fiabilité n'étant pas forcément garantie au niveau de *Pd*, il en serait de même quand on passe de *Pd* vers *Pc*.
- Autres problèmes critiques dans la conversion de *Pd* vers *Pc*: comment établir la connexion avec les équipements de destination puisque les adresses destination peuvent varier d'un paquet à l'autre dans le mode datagramme ? Si *n* paquets arrivent l'un à la suite de l'autre au niveau du relais avec des adresses de destination différentes, va t-on établir *n* circuits virtuels et envoyer un seul paquet par circuit?

La Figure 3.4 illustre quelques-uns des problèmes soulignés. La machine B3 envoie des paquets 1 et 2 respectivement à la machine A1 qui les reçoit dans l'ordre inverse parce que le paquet 2 est arrivé avant le paquet 1 au niveau du relais. On peut aussi voir que plusieurs circuits virtuels, correspondants à chaque destination, sont créés dans le réseau A à chaque fois qu'un paquet est reçu par le relais. Cette technique peut congestionner le relais et causer la perte régulière des paquets car, pendant l'établissement des multiples connexions, le tampon de réception du relais peut se remplir rapidement.

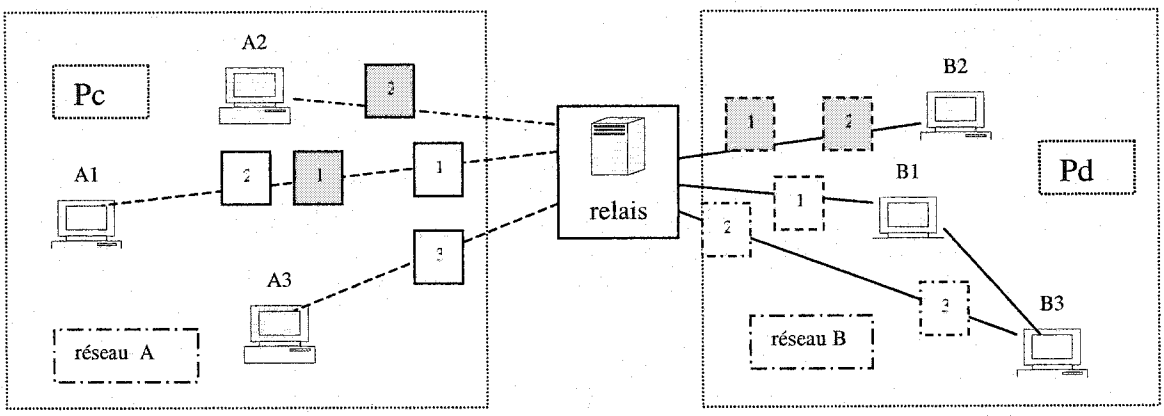


Figure 3.4 Communication entre Pc (connexion) et Pd (datagramme)

Si des paquets d'un même flot toujours destiné à un même équipement arrive au relais entre *Pd* et *Pc*, il est possible de concevoir un tampon de grande capacité qui enregistre d'abord tous les paquets provenant du réseau utilisant le protocole *Pd*, ensuite créer une connexion fiable avec la machine destinataire (utilisant *Pc*) avant d'envoyer les données. En pratique, ce n'est pas le cas et cette approche peut se révéler inefficace, surtout pour des données devant être délivrés en temps réel.

On voit donc comment la différence de mode de fonctionnement peut entraver la convertibilité des protocoles. Si les protocoles initiaux possèdent un mode commun de fonctionnement, il sera plus simple d'utiliser ce mode commun.

3.3.2.2 Différences entre mécanismes d'adressage

Nous avons constaté dans notre survol de protocoles que l'adressage intervenait à presque toutes les couches. Nous montrons dans la suite que les différences entre logique d'adressage peuvent constituer un frein à la conversion, surtout en l'absence d'un mécanisme de correspondance des adresses.

Considérons deux protocoles P_1 et P_2 ayant chacun un mécanisme d'adressage, disons Adr_1 et Adr_2 respectivement, et deux réseaux N1 et N2 implémentant

respectivement P_1 et P_2. On suppose que les deux systèmes d'adressage n'ont pas les mêmes sémantiques, i.e. les logiques sont différentes. Le relais placé à la frontière des deux réseaux doit être capable de réaliser une correspondance entre les adresses Adr1 et Adr2 pour que la conversion soit possible, et c'est là que réside tout le problème.

Certaines adresses n'ont de signification que dans le réseau local (comme les numéros VPI-VCI contenus dans un paquet ATM), tandis que d'autres caractérisent un nœud précis dans le monde sans ambiguïté possible (comme une adresse IPv4). La conversion entre ces deux types d'adresses est souvent difficile voire impossible à réaliser de façon dynamique, et il faut parfois recourir à des mécanismes complexes (comme ARP, table statique de correspondance d'adresses) dans les relais. La conversion de P_2 en P_1 peut donc être problématique si les deux protocoles n'ont aucun système d'adressage commun.

La Figure 3.5 illustre un cas de figure dans lequel le protocole P_1 est le protocole IP et P_2 le protocole ATM. La machine M_1 du réseau N1 doit communiquer avec la machine M_2 du réseau N2 à travers le réseau ATM. Les paquets de M_1 contiennent l'adresse IP de la machine M_2. À la frontière entre le réseau IP et le réseau ATM, le relais doit inclure le numéro de circuit virtuel permettant d'aller vers la machine M_2.

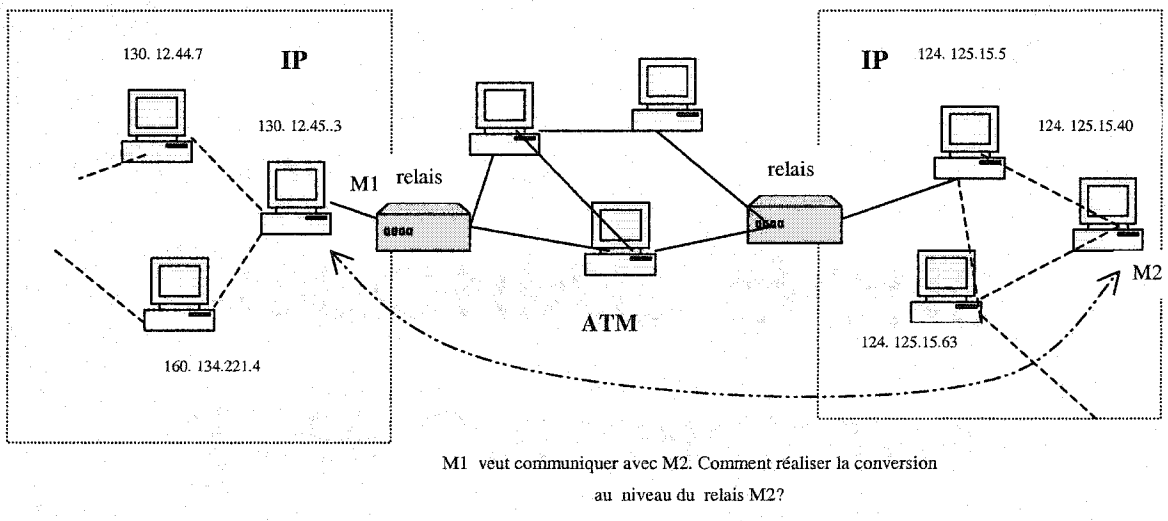


Figure 3.5 Problème de conversion d'adresses au niveau du relais

Étant donné qu'il n'y a aucune similitude entre ces deux mécanismes d'adressages, il se pose un problème d'adressage. Une solution peut consister à garder une table de correspondance entre adresses IP et les numéros de circuits qui permettent de les atteindre. Mais le nombre de circuit étant limité, on peut se retrouver face à une situation où plusieurs machines IP se voient attribuer un même circuit virtuel.

Le dernier scénario est celui dans lequel les deux systèmes ont des adressages locaux n'ayant aucun rapport. C'est le cas par exemple d'une machine M_1 se trouvant dans un réseau Decnet devant communiquer avec une autre M_2 située dans un environnement Appletalk (Figure 3.6). Rappelons que, dans ces réseaux, l'adressage est composé d'un numéro de zone suivi d'un numéro de nœud. On se retrouve encore face au même problème de conversion. Quelle valeur inclure dans l'entête du message pour indiquer qu'on communique avec M2 qui est dans le réseau Appletalk? Dans ce genre de situation, une solution peut consister à maintenir une table de correspondance entre les adresses Adr1 et Adr2. En général, cela est possible lorsqu'il existe un adressage commun aux deux systèmes à interconnecter à une couche donnée. On va alors transiter par ce mécanisme d'adressage commun pour établir la correspondance des adresses.

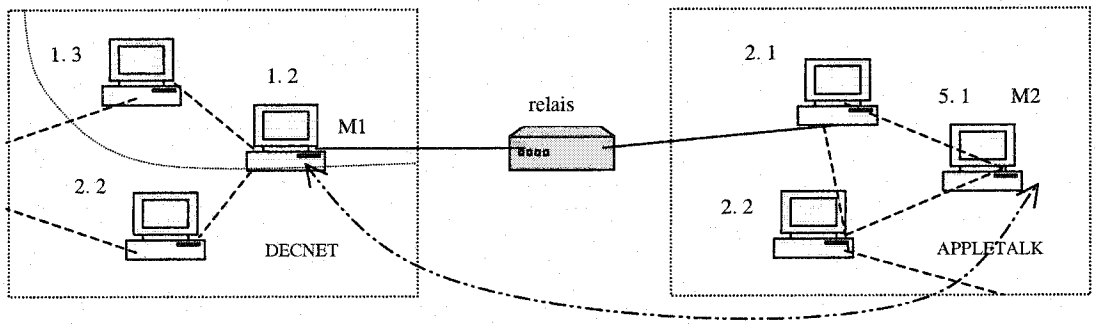


Figure 3.6 Problème d'adressage rendant l'interconnexion difficile

3.3.2.3 Différences dues aux mécanismes d'accusé de réception

Nous indiquons ici l'une des raisons pour lesquelles le mécanisme d'accusé de réception (ACK) est une limite à la convertibilité si un seul des protocoles en dispose.

Soient P_1 et P_2 deux protocoles à convertir. P_1 utilise un mécanisme d'accusé de réception tandis que P_2 n'en dispose pas. On remarquera que le bon fonctionnement de ce mécanisme d'ACK est souvent crucial pour les protocoles qui l'utilisent comme TCP, X.25, etc., car ils attendent à chaque envoi la confirmation de la réception par le destinataire. Dans un scénario typique de communication, P_1 envoie un message, disons m, à P_2 et attend un ACK. Étant donné que P_2 n'implémente pas ce mécanisme d'accusé de réception, il ne peut émettre de réponse pour dire à P_1 qu'il a reçu le message envoyé par ce dernier. Il incombe alors au relais chargé de réaliser la conversion entre les deux protocoles de retourner l'ACK à P_1. Le problème : Quand envoyer l'accusé de réception à P_1? On pourrait choisir d'implémenter au niveau du relais une solution qui consisterait à envoyer l'ACK tout de suite après avoir envoyé le message converti vers P_2. Cette solution n'est cependant pas correcte puisque P_1 peut recevoir l'ACK et générer un autre message sans que P_2 n'ait reçu le premier. On voit ainsi qu'il serait difficile d'assurer le bon fonctionnement du protocole P_1 dans ce processus de conversion.

3.3.2.4 Un seul type versus plusieurs types de paquets

Certains protocoles ne disposent que d'un seul type (et donc un seul format) de paquet tandis que d'autres en ont plusieurs (ex: X.25, IEEE1394). C'est là un autre type de problème qui gêne la conversion. En effet, soit à convertir un protocole P_1 ayant plusieurs types de paquets (commandes, contrôles, données), en un protocole P_2 disposant d'un format unique de paquets (i.e., commandes, contrôles et/ou données sont tous dans le même paquet). En général, si les deux protocoles supportent les mêmes paramètres de commande et de contrôle et offrent des services similaires, la correspondance peut être aisée. Mais, dans les cas où P_2 ne disposerait pas des options de contrôle ou de commande de P_1, que faut-il faire avec les paquets correspondants générés par P_1? Le principal problème qui se pose alors est la correspondance entre

certains types de paquets du protocole P_1 et celui de P_2 si les services offerts par les premiers ne sont pas prévus dans P_2. Un exemple concret est le cas X25-IP. Les paquets de commande de X25 qui gèrent l'établissement ou la terminaison des connexions n'ont pas de correspondance dans IP. Nous avons dans la discussion précédente utilisée le terme "paquets", ce qui réfère surtout aux protocoles des couches basses, mais toutes les observations faites peuvent s'appliquer aussi aux protocoles des hautes couches manipulant des messages.

3.3.2.5 Limite de taille de paquets

Les différences entre la taille limite des paquets peuvent constituer aussi un obstacle dans le processus de conversion de deux protocoles. Ceci arrive surtout au niveau des protocoles orientés bit aux basses couches. Si le MTU (Maximum Transfert Unit) du protocole P_1 est inférieur à celui de P_2, cela n'influence pas la conversion dans le sens $P_1 \rightarrow P_2$. Mais dans le sens opposé, il faudrait implémenter un mécanisme de fragmentation et assemblage pour pallier ce problème. Autrement, les paquets de taille supérieure à la taille maximale tolérable seront jetés.

On peut encore continuer avec les difficultés dues à la nature (bit ou caractère) des protocoles. C'est le point de remarquer que les obstacles à la convertibilité sont multiples et en faire une liste exhaustive est difficile. Nous allons pour cela, dans la suite, établir les critères généraux permettant de décider de la convertibilité des protocoles.

3.3.3 Critères de convertibilité

Sur la base de l'étude réalisée et des observations faites précédemment, nous énonçons ci-dessous un ensemble de critères nécessaires à la convertibilité de deux protocoles quelconques P et Q. Nous commençons d'abord par introduire les notions de «*services primaires*» et «*services secondaires*», que nous utiliserons par la suite.

Définitions

Étant donné un protocole P offrant un ensemble S de services :

- Un élément S_i de S est dit *primaire* s'il est obligatoire au bon fonctionnement de P. Autrement dit, P ne peut plus offrir aux couches supérieures ses services si un élément empêche le fonctionnement de S_i. Pour un protocole d'acheminement de données opérant à la couche réseau par exemple, l'adressage est un service critique qui, s'il ne fonctionne pas correctement, influera sur le fonctionnement de tout le protocole, puisque les données ne seront pas délivrées au bon destinataire. Ainsi, l'adressage est un service primaire. Un champ offrant un service primaire est un *champ primaire*.
- Un élément S_j de S est *secondaire* si le protocole P peut fonctionner sans forcément offrir le service S_j aux couches supérieures, i.e. S_j peut être désactivé sans que cela n'ait une influence particulière sur P. Un champ contribuant à offrir ce service est appelé *champ secondaire*.
- Un service qui n'est pas primaire est secondaire.

Critères nécessaires à la convertibilité

1. Il existe un ensemble non vide de services communs aux deux protocoles P et Q. (Green, 1986).
2. Il existe un ensemble non vide de services primaires communs à P et Q. Autrement, on s'assurera que, pour chaque protocole, l'ensemble des services primaires a son équivalent dans l'ensemble des services de l'autre protocole. Si un service primaire n'a pas son équivalent, alors il ne peut être offert. On pourra alors utiliser une approche de complémentation pour réaliser la conversion de ce service.
3. Les services primaires identiques doivent être convertibles d'un protocole à l'autre. L'adressage en est un exemple. S'il est présent dans P et Q, il est nécessaire d'avoir un mécanisme de conversion des adresses; à défaut d'un tel mécanisme, le service, bien qu'il soit commun aux protocoles, ne serait pas convertible.

4. Les protocoles doivent offrir, idéalement, le même type de service: datagramme ou orienté connexion. Dans les cas où ils opèrent selon différents modes, la conversion sera souvent difficile à réaliser. En général, une approche de complémentation pourra être utilisée pour pallier le problème le cas échéant (surtout le mode datagramme et le mode circuit virtuel avec accusé de réception).
5. Les protocoles à convertir disposent d'un format bien défini (syntaxe précis) pour leurs messages. Ce point signifie que les protocoles de type algorithmique comme RIP et OSPF ne sont pas concernés.
6. Les protocoles considérés doivent être tous deux soit orientés bit (format de paquet connu) ou orientés caractère (commandes et réponses). Dans le cas contraire, il faut un mécanisme de correspondance bien précis entre les bits et les caractères.
7. Si *P* et *Q* opèrent aux hautes couches et offrent des mécanismes d'établissement de connexion entre processus, il faut de plus une correspondance directe entre les commandes utilisées par l'un et l'autre pour l'établissement ou la terminaison de connexion, le transfert des données et peut être le contrôle de ce transfert.

Lorsque tous les critères énumérés sont respectés, on peut assumer que *P* et *Q* sont convertibles, pour offrir l'ensemble des services communs (aussi bien primaires que secondaires) convertibles entre eux.

Critères suffisants pour décider de la convertibilité

Deux protocoles *P* et *Q* sont convertibles s'ils offrent les mêmes services (aussi bien primaires que secondaires) et si il existe, pour chaque service commun, un mécanisme de correspondance entre l'information contenue dans chacun des protocoles et correspondant à ce service. De plus, P et Q opèrent selon le même mode (datagramme ou connecté) et disposent de formats similaires de paquets ou de messages.

3.3.4 Méthode de détermination de convertibilité proposée

Nous proposons, dans la suite, une approche pour identifier la convertibilité des protocoles. Nous présentons d'abord une méthode proposée par Green (1986). Par la suite, la nouvelle technique est explicitée et étayée par un exemple.

3.3.4.1 La méthode de Green

Au chapitre 2, nous avions noté que Green (1986) avait déjà proposé une méthode permettant de prédire si la conversion directe est possible entre deux protocoles donnés. Si elle ne l'est pas, l'approche suggérée est soit la conversion indirecte, soit la complémentation. La Figure 3.7 illustre cette technique.

Cette méthode peut se résumer en quatre étapes: identification des nœuds et couches, identification des fonctionnalités "atomiques" des protocoles, détermination de l'ensemble de fonctions ou services communs, comparaison avec les requis de l'application. Si les requis de l'application sont respectés, et s'il n'y a pas une grande incompatibilité entre les protocoles, alors la conversion (directe ou indirecte) est possible. Sinon, on choisit la voie de la complémentation. L'approche de Green utilise, outre les spécifications et informations de contrôle des protocoles, les requis de l'application pour laquelle ils seront utilisés. En d'autres termes, connaissant P et Q ainsi que leurs syntaxes, formats de messages, informations de contrôle, elle ne permet pas de décider de leur convertibilité tant que les requis de l'application ne sont pas précisés.

Nous voulons une technique moins contraignante, qui tient compte uniquement des propriétés des protocoles et nous indique éventuellement les services supportés.

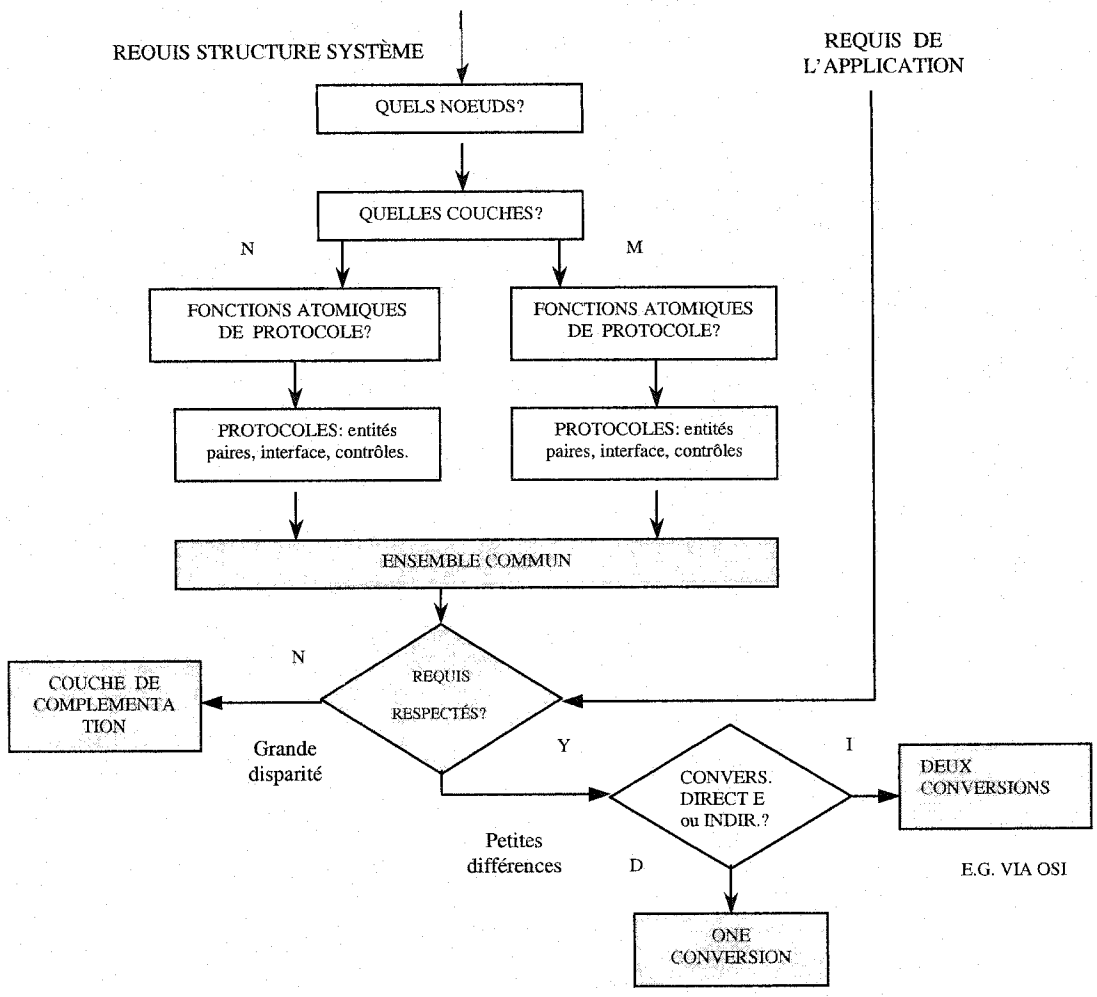


Figure 3.7 Méthodologie de Green

3.3.4.2 L'algorithme proposé

La technique que nous proposons permet d'identifier si la conversion est réalisable et retourne les services qui sont supportés. Elle ne nécessite donc pas de connaître les requis de l'application.

Dans la suite, il est bien clair que le mot "conversion" désigne la conversion au sens large, i.e. la correspondance entre services de chacun des protocoles en présence, englobant aussi les mécanismes de complémentation. L'encapsulation peut résoudre l'interopérabilité lorsque la conversion n'est pas possible. Voici les principales étapes de la méthode proposée dont le schéma est donné à la Figure 3.9.

Étape 1 : Identification des services de chaque protocole.

Cette étape cruciale se divise en deux sous étapes: l'identification des services primaires puis celle des services secondaires. Au terme de cette phase, tous les services offerts par le protocole et, en particulier, ceux dont les éléments figurent dans les informations de contrôle de protocole (entête, commandes etc.) sont identifiés. Soit *S* l'ensemble des services communs aux deux protocoles, *S1* l'ensemble des services du protocole P1 et *S2* celui de P2. *Spri1* est l'ensemble des services primaires de P1 et *Spri2* celui de P2. *Ssec1* désigne l'ensemble de toutes les fonctions secondaires offertes dont les éléments se retrouvent dans le PCI du protocole P1; il en est de même pour *Ssec2*.

Étape 2 : Identification des services primaires communs aux deux protocoles.

Il suffit de faire l'intersection *Spri* de *Spri1* et *Spri2*. Si *Spri* est vide, alors vérifier si *Spri1 (resp. Spri2)* est un sous-ensemble de *S2 (resp. S1)*. Si *Spri* est vide et que *Spri1 (resp. Spri2)* n'est pas un sous ensemble de *S2 (resp. Spri1)*, l'algorithme termine par un échec, i.e. P1 et P2 ne sont pas convertibles, car n'offrent pas des services similaires. Dans le cas contraire, on peut envisager une possibilité de convertibilité. Si *Spri* = *Spri1* = *Spri2*, on peut alors assurer la convertibilité dans les deux sens.

Étape 3 : Détermination des services convertibles.

Aussi bien services primaires que secondaires sont examinés individuellement pour décider de ceux dont les éléments sont convertibles d'un protocole à l'autre. Les services communs qui ne sont convertibles ni directement, ni par addition même d'une fonction de complémentation, ne peuvent être offerts à l'usager après la conversion. On les soustrait donc de l'ensemble des services communs et on procède ainsi jusqu'à obtenir l'ensemble *S'* des services communs convertibles entre les deux protocoles.

Étape 4 : À la fin de l'algorithme, si *S* est non vide et contient *Spri1* ou *Spri2*, alors les protocoles peuvent être convertis et offrir les services contenus dans *S*.

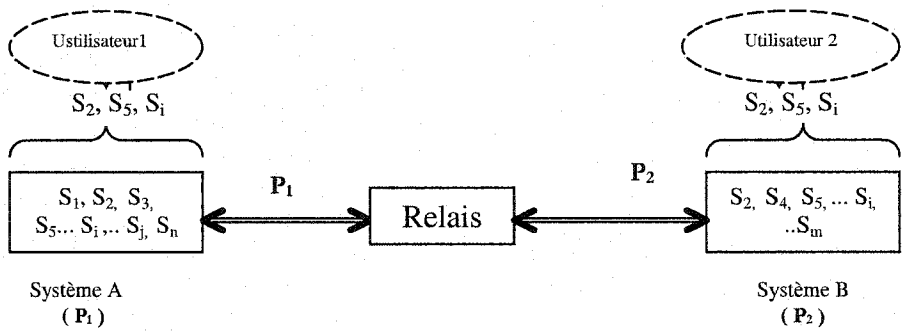


Figure 3.8 Intersection entre les services de P1 et P2

Si *Spri1 (resp. Spri2)* $\subset$ *S*, alors la conversion est possible dans le sens P2 vers P1 (resp. P1 vers P2).

3.3.4.3 Identification des services primaires et secondaires

L'identification des services primaires des protocoles est une étape nécessaire à la bonne utilisation de la méthode proposée. De toute évidence, si elle est mal faite, elle aura un impact irréversible sur le déroulement et le résultat final de l'algorithme. Nous préconisons pour cette raison quelques indices pour la détermination de l'ensemble des services primaires et secondaires d'un protocole P.

- Considérer la (ou les) fonction(s) principale(s) de P. Dans plusieurs cas, elle correspond à celle de la couche OSI à laquelle opère P. Pour illustrer, considérons les protocoles IP, Ethernet, Firewire, MPLS. IP opère à la couche réseau et ses fonctions principales sont l'acheminement des datagrammes se basant sur l'adresse du nœud destinataire, la fragmentation et l'assemblage des paquets (services de base de la couche réseau de OSI). Ethernet opère à la couche liaison et sa fonction principale est l'acheminement des trames entre deux nœuds d'un LAN avec service datagramme. Il en est de même pour Firewire. Quant à MPLS, sa fonction principale est de permettre un routage basé sur la source; elle est différente de la

fonctionnalité première de la couche liaison. En général, ces fonctions principales vont correspondre aux services primaires.

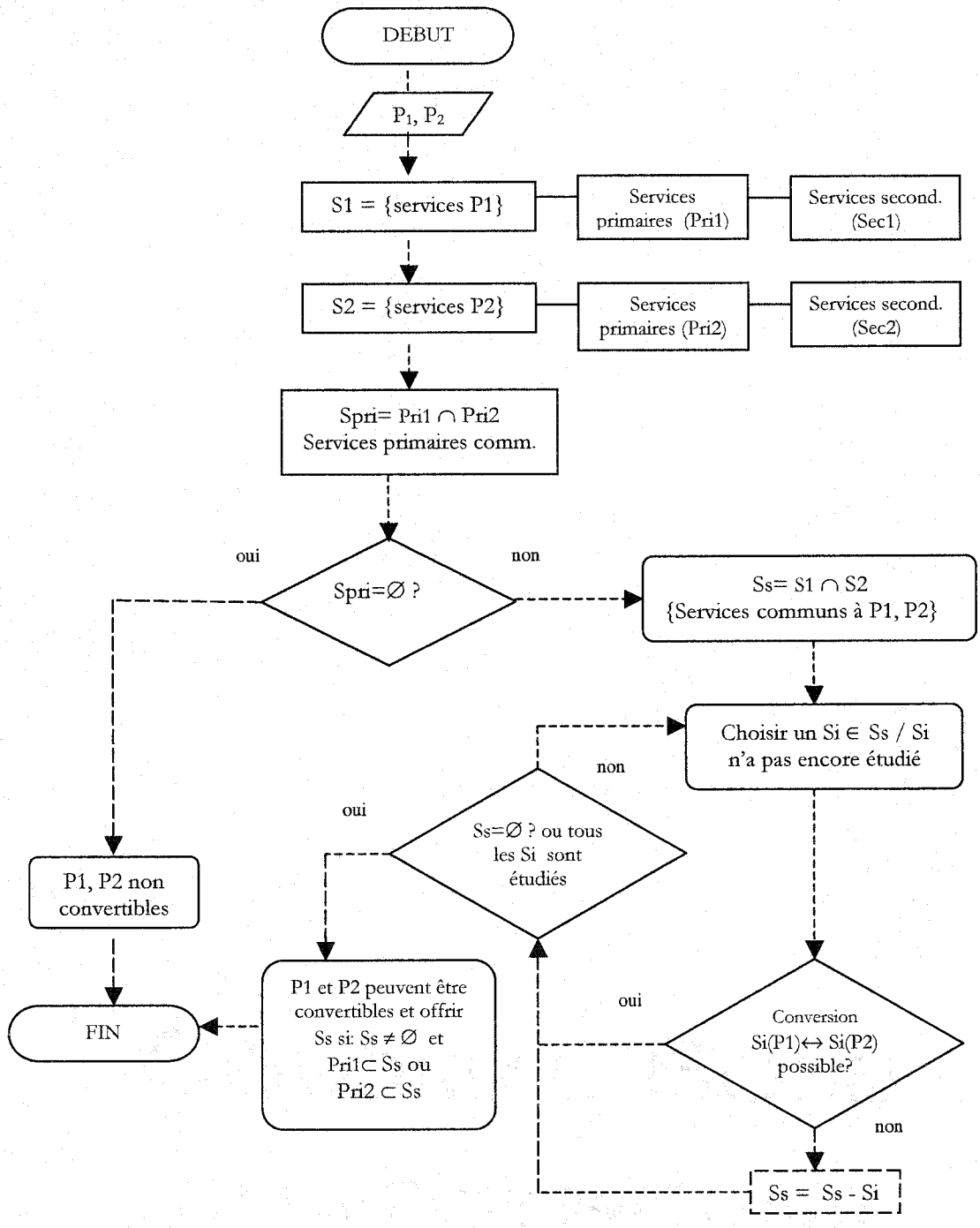


Figure 3.9 Algorithme de convertibilité proposé

- Identifier les éléments du PCI (informations de contrôle de protocole) qui permettent d'offrir cette (ou ces) fonction(s) principale(s). Si P est orienté bit, il existe certainement dans l'entête un (ou des) champ(s) permettant d'offrir cette (ces) fonction (s) de base. Les services associés à ces champs sont alors considérés comme primaires. En revenant aux protocoles considérés dans le point précédent, l'adressage logique dans IP ou MPLS, l'adressage physique dans Ethernet et Firewire sont autant de services primaires. Si P est orienté caractère, ce seront généralement des commandes et réponses (ex : commandes d'établissement ou terminaison de connexion, de transfert de données, etc.) que l'on considérera.
- Tout autre service offert par P qui n'est pas primaire est secondaire.

3.3.4.4 Détermination des services convertibles

Certains services communs aux protocoles peuvent ne pas être convertibles entre eux, c'est-à-dire qu'il n'existe pas de mécanisme permettant d'établir automatiquement la correspondance des informations concernant ce service. Pour illustrer, nous pouvons reprendre l'exemple des adressages MAC et ATM dans le cas des protocoles Ethernet et ATM. ATM utilise des numéros de circuit virtuel et il n'y a normalement pas de correspondance automatique possible entre circuit virtuel et adresse physique MAC. Ainsi, bien que le service d'adressage soit commun, il n'est pas convertible.

La détermination des services convertibles consiste alors à identifier, pour chaque service commun, la possibilité de convertir les éléments de service correspondants (contenus dans les entêtes ou les informations PCI des protocoles).

Nous n'avons pas de méthodologie générale à proposer quant à la correspondance des éléments de service des PCI, car elle peut varier d'un protocole à l'autre selon le type de service considéré. Les services convertibles peuvent être mentionnés dans une base de données par exemple.

3.3.5 Exemple d'application de la méthode proposée

Nous donnons dans cette section un exemple de la manière dont l'algorithme proposé peut être utilisé en pratique pour déterminer si la conversion est possible entre deux protocoles. Il est appliqué aux protocoles Ethernet et ATM.

Convertibilité des protocoles Ethernet et ATM

1. Identification des services de chaque protocole

Protocole Ethernet802.3

Ethernet est un protocole LAN; il assure la transmission des trames sur lien physique et opère en mode datagramme. Il dispose d'un mécanisme d'adressage physique MAC pour adresser les machines d'un réseau LAN et d'un mécanisme de contrôle des données transportées. Il opère en mode full-duplex, et le format de la trame montre aussi des champs pour la délimitation, la longueur des données. On se limite aux services offerts à la couche liaison, la spécification englobant aussi la couche physique.

- Ensemble de services S = {datagramme, adressage physique source et destination, contrôle d'erreur des données, délimitation de trame, full-duplex, longueur données};
- Services primaires *Spri1* = {datagramme, adressage physique source et destination, contrôle d'erreur des données};
- Services secondaires *Ssec1* = {délimitation trame, full-duplex, longueur données}.

Protocole ATM

ATM spécifie l'interface entre l'usager et le réseau d'une part, et entre deux éléments du réseau d'autre part (UNI, NNI). Il opère à la couche liaison.

- Ensemble de services S = {circuit virtuel, adressage circuits virtuels VPI/VCI, contrôle d'erreur d'entête, contrôle congestion (priorité des paquets), *QoS*};
- Services primaires *Spri2* = {circuit virtuel, adressage VCI, contrôle erreur entête};
- Services secondaires *Ssec2* = {contrôle congestion, QoS}.

2. *Identification des services primaires communs aux deux protocoles*

Spricomm = {adressage, contrôle d'erreur}. On remarque qu'il y a différence entre les modes de fonctionnement des deux protocoles, l'un offrant un service *datagramme* et l'autre, un service *circuit virtuel*, ce qui est critique pour la conversion des protocoles. Une fonction de complémentation pourrait permettre de résoudre cette disparité.

3. *Déterminer les services convertibles*

L'adressage MAC, est-il convertible en adressage VPI-VCI de ATM? Nous en avions déjà discuté plus haut, ce sont deux adressages n'ayant aucun rapport. La conversion n'est pas possible. Et même si nous envisageons une possibilité de travailler avec les adresses physiques ATM, on sait qu'ils sont formatés sur 20 octets et n'ont pas de rapport avec les adresses MAC Ethernet. Quant au contrôle d'erreur, dans ATM il porte sur l'entête tandis que, dans Ethernet, c'est toute la trame qui est considérée. Cette différence peut aussi être résolue par l'addition d'une fonction (complémentation) permettant de réaliser le contrôle sur tout le paquet ou sur l'entête seulement selon le protocole de départ.

4. *Conclusion*

On voit que le service d'adressage qui est un service primaire n'est pas convertible, donc il ne peut être supporté dans une conversion de Ethernet à ATM. La conclusion normale serait que la conversion n'est pas possible. Cependant, une fonction de complémentation peut permettre d'offrir la conversion des adresses. En pratique, dans un contexte où les deux protocoles utilisent les services du protocole IP, les adresses IP peuvent être utilisées pour réaliser la conversion des adresses entre les nœuds ATM et Ethernet (*IP sur ATM*). Chaque nœud ATM et chaque nœud Ethernet dispose d'une adresse IP. Ceci permet à chaque machine de l'un des réseaux de pouvoir communiquer avec une autre sur l'autre réseau. Dans ce cas, les protocoles seraient convertibles puisque l'ensemble de services primaires *Spri* serait inclus dans l'ensemble des services convertibles.

3.4 Schéma de convertisseur générique de protocoles

Nous proposons dans la suite un schéma de convertisseur que nous voulons générique. Ce convertisseur va supporter un ensemble de protocoles des basses couches et des couches hautes satisfaisant à certaines propriétés que nous préciserons par la suite. Sachant que l'on ne peut envisager une convertibilité qu'entre des protocoles offrant des services similaires (correspondant aux mêmes couches du modèle de référence), nous proposons un modèle qui s'inspire des couches du modèle OSI. Chaque protocole appartient à une couche précise, et une couche est dans notre modèle une abstraction contenant un nombre prédéfini de services (correspondant aux services fréquemment offerts par les protocoles courants opérant à cette couche).

Notre but est de concevoir une architecture qui va offrir la conversion "directe" entre des protocoles usuels. Ses caractéristiques sont les suivantes:

- flexible, de manière à permettre des ajouts de nouveaux protocoles, aussi bien normalisés que propriétaires;
- facile d'utilisation grâce à une décomposition modulaire pour permettre l'ajout de nouvelles fonctions de conversion entre services, non encore programmées;
- supporter des protocoles de toutes les couches (2 à 7), même si l'accent est pour le moment mis sur ceux opérant dans les basses couches;
- totalement programmable et facilement modifiable pour l'adapter à des besoins spécifiques.

L'approche choisie, schématisée à la Figure 3.10, est basée essentiellement sur quatre modules quasi-indépendants: un analyseur de protocole, un module principal de conversion, un convertisseur de données de service et enfin une base de données de service.

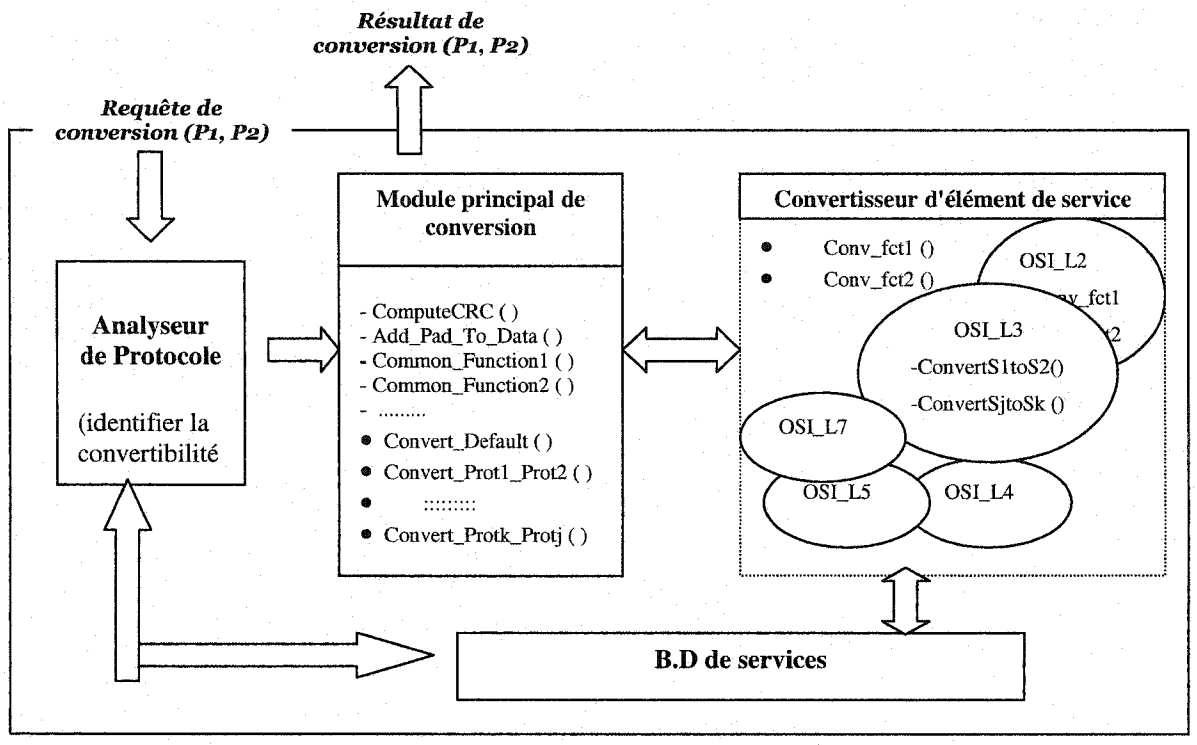


Figure 3.10 Schéma simplifié des modules de notre convertisseur de protocoles

Brièvement, l'analyseur identifie la convertibilité des protocoles, le module principal de conversion réalise la conversion en s'aidant des fonctions de conversion spécifiques du convertisseur des données de service. Ces modules vont en général collaborer avec la base de données qui contient un ensemble de services à chaque couche OSI et maintient une association entre les éléments de services convertibles entre eux d'une part, et une autre association entre les éléments de service de même type d'autre part.

3.4.1 Description et fonctionnement des éléments de notre modèle

Lorsqu'il y a une requête de conversion de deux protocoles *req (P1, P2)*, l'analyseur de protocoles est appelé. Si les protocoles sont prouvés convertibles, le module principal réalise la conversion avec l'aide du convertisseur des services et de la base de donnée de services.

L'analyseur

Il identifie d'abord les couches OSI des protocoles (si elles sont spécifiées) et applique l'algorithme de convertibilité proposé. Il appelle une fonction de comparaison qui va identifier les services communs aux deux protocoles en consultant les champs de l'entête. Un élément de service commun aux deux protocoles sera enregistré dans une table de correspondance avec les positions correspondantes dans chacune des entêtes. Lors de la conversion, il suffit alors de faire la correspondance entre les éléments de service contenus dans cette table. Bref, ce module applique l'algorithme de convertibilité proposé et retourne, dans le cas où les protocoles sont convertibles, la liste des correspondances entre champs convertibles, et la couche à laquelle ils opèrent. Ces paramètres sont ensuite passés au module principal de conversion de protocole.

Le module principal

C'est l'entité qui réalise la conversion proprement dite. Il peut être considéré comme une classe qui dispose d'un ensemble de fonctions génériques, fréquemment utilisées comme le calcul de CRC, l'attribution d'une valeur par défaut à un champ, etc., (*Compute_CRC(), Add_Padding(), setdefaultvalue()*), mais aussi d'autres fonctions particulières correspondant aux ajouts de l'usager. C'est dans ce module que l'on pourra définir des fonctions propres à la conversion de deux protocoles précis, si l'on estime que la traduction des informations requiert des ajouts considérables que la fonction de conversion générique ne pourrait pas supporter.

Le convertisseur d'éléments de service

Il contient plusieurs sous-modules correspondant chacun à une abstraction d'une couche donnée du modèle de référence. Chaque sous-module dispose d'un ensemble prédéfini de fonctions de conversion entre éléments de service, mais d'autres fonctions peuvent être ajoutées à tout moment. Des exemples de fonctions qu'on retrouvera dans le sous-module correspondant à la couche 3 sont: *Convert_IPv4addr_to_IPv6addr(), Convert_Check16_to_Check32(), Get_Pkt_Len(), etc.*

La base de données de service

Elle maintient un ensemble de services prédéfinis qui peuvent être mis à jour par l'usager. Les services peuvent être définis sous forme de *flags* (valeur constante) correspondant chacun à un type précis d'information. On pourrait par exemple associer à toute donnée d'adressage le préfixe ADDR. Une information d'adressage de nœud source réseau pourrait correspondre à L3_SRC_ADDR et celle d'un nœud destination serait par exemple L3_DEST_ADDR. Pour IPv4 par exemple, on aurait SRC_ADDR_IP4 ou DEST_ADDR_IP4. De plus, la BD maintient pour chaque type de service la liste des formes possibles d'unité de données. Par exemple, en considérant L3_SRC_ADDR comme un type de service (adressage nœud source, niveau réseau), une liste partielle des formes possibles d'unité de données pourrait être: SRC_ADDR_IP4, SRC_ADDR_IP6, SRC_ADDR_VC, SRC_ADDR_X121, etc. Enfin, comme mentionné précédemment, la BD contient aussi une liste d'associations entre services convertibles. Par exemple, il y aurait une association <SRC_ADDR_IP4, SRC_ADDR_IP6>, ou <PKT_LENGTH, DATA_LENGTH> pour indiquer qu'ils sont convertibles.

Nous ne définissons pas de format particulier pour les éléments de la base de données. Un format simple (fichier texte par ex.) peut être utilisé. Cependant, on peut aussi opter pour des technologies plus modernes comme XML.

3.4.2 Définition du modèle de protocole

L'algorithme et le convertisseur générique proposés, prennent en entrée deux protocoles avec des informations bien spécifiques. On suppose que les spécifications de chaque protocole sont inclues dans un fichier portant son nom. La Figure 3.11 montre les informations principales chargées dans la structure de donnée *Protocole* et contenues dans chaque fichier.

Un protocole est caractérisé par un ensemble d'attributs propres que nous citerons par la suite. Il intègre aussi les paramètres de services primaires et secondaires définis par l'utilisateur. En effet, vu la diversité et le nombre sans cesse croissant des

protocoles, il est difficile de trouver une méthode qui éviterait à l'usager de spécifier ces paramètres. Les principaux attributs sont les suivants :

Prot_name	: nom du protocole
Type	: nature du protocole (orienté bit ou caractère)
Architecture	: nom de l'architecture associé
Osi_layer	: couche OSI du protocole
Nb_packet	: nombre de paquets
Liste_des_champs	: liste des champs du paquet (format du paquet)
Nb_primary_serv	: nombre de services primaires
Primary_serv	: liste de services primaires
Nb_secondary_serv	: nombre de services secondaires
Secondary_serv	: liste de services secondaires

Il faut préciser que le format du paquet (*Liste_des_champs*) est répété pour chaque type de paquet du protocole et que, pour chaque champ, on précise les informations suivantes : numéro dans le format, nom, nombre de bits alloués, service offert, valeur statique ou variable, lorsqu'on est en présence d'un protocole orienté bit.

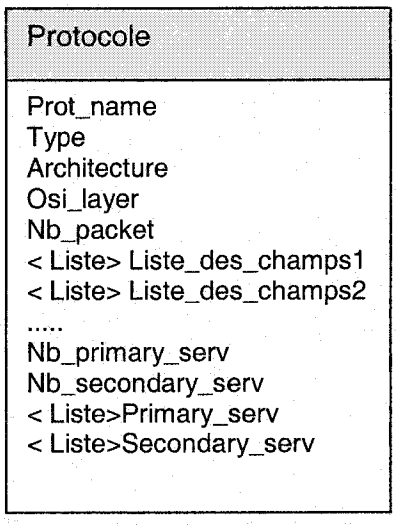


Figure 3.11 Modèle de l'objet protocole

3.4.3 Algorithme de conversion

L'algorithme présenté à la Figure 3.12 peut être utilisé comme algorithme général de conversion au niveau du module principal. On suppose que les protocoles à convertir sont soit orientés bit, soit orientés caractère. On considère aussi qu'ils ont chacun un format bien précis lorsqu'ils sont orientés bit, et que les champs convertibles de P1 et P2 sont enregistrés dans une structure *Convertible_Fields_List*. Lorsqu'ils sont orientés caractère, les correspondances entre commandes, réponses et contrôles sont dans *Convertible_Msgs_List*.

ENTREES:

- P1 : objet protocole (source), P2 : objet protocole (destination)
- Pkt1: paquet à modifier au format de P1 (ou Msg1 pour protocole orienté caractère)
- Convertible_Fields_List : liste des champs convertibles entre P1 et P2

SORTIE: - Pkt2: paquet modifié au format de P2 (ou Msg2 pour protocole orienté caractère)

DÉBUT:

Si il n'existe pas de fonction de conversion de P1 et P2, **Alors**

Si P1 et P2 sont des protocoles *orientés bit*, **Alors**

Obtenir le format du protocole destination P2 et créer un paquet Pkt2 de P2

Pour chaque champ *Fi* de Pkt2, faire:

Obtenir le nom *Si* du service fourni par *Fi*.

Si *Si* est convertible entre P1 et P2, **Alors**

Lire le champ *Ej* de P1 corresp. à *Fi* dans Convertible_Fields_List

Obtenir le nom *Sj* du service fourni par le champ *Ej*.

Si une fonction de conversion *Convert_Sj_to_Si( )* existe, **Alors**

call: *Convert_Sj_to_Si (Fi, Ej) // convertir Ej en Fi.*

Sinon *// convertir Ej en Fi avec la fonct. de conversion génériq.*

call: *Convert_Gener_Serv(Fi, Ej).*

Fin **Si**

Sinon

Chercher *Fi* dans la liste des champs à valeur constante de P2

Si *Fi* est trouvé, **Alors**

Attribuer à *Fi* la valeur constante prévue.

Sinon attribuer une valeur par défaut à *Fi*.

Fin **Si**

Fin **Si**

Fin **Pour**

Retourner Pkt2.

Fin **Si** *// protocoles orientés bit*

```
// Si les protocoles sont orientes caract., on suppose qu'un message est soit une
// commande, réponse, contrôle ou données.
    Si Protocoles orientés caractère, Alors
        Identifier le type du message Msg1 de P1
        Si le type de Msg1 est : commande ou contrôle ou réponse, Alors
            Obtenir le message correspondant Msg2 de P2.
            Si  Msg1 ⊂ Convertible_Msgs_List, Alors
                Lire Msg2 de P2 équivalent
                Retourner Msg2.
            Sinon
                Retourner un message par défaut.
            Fin Si
        Fin Si

        Si le type de Msg1 est : données, Alors
            Msg2 = Msg1.
            Retourner Msg2.
        Fin Si
    Fin Si // protocoles orientés caractère
Fin Si  // s'il n'existe pas de fonction de conversion entre P1 et P2

FIN
```

Figure 3.12 Algorithme de la fonction de conversion (P1→ P2)

CHAPITRE IV
IMPLÉMENTATION, MISE EN ŒUVRE ET ANALYSE DE PERFORMANCE

Ce chapitre expose les détails d'implémentation et la mise en oeuvre du convertisseur générique de protocoles ainsi que des résultats d'application de la procédure proposée pour identifier la convertibilité. Nous y présenterons d'abord le diagramme des classes de notre convertisseur. Puis, nous décrirons les différentes structures de données ainsi que les principaux algorithmes. On montre comment la structure proposée peut supporter facilement la conversion de plusieurs protocoles. Les résultats obtenus de l'application de notre algorithme à quelques protocoles d'usage courant y sont présentés. Après avoir évalué la performance du modèle proposé, nous étudierons la convertibilité des protocoles Firewire (IEEE1394) et Ethernet.

4.1 Implémentation du convertisseur générique de protocoles

Nous présenterons, dans la suite, une vue générale du convertisseur à travers un diagramme général des classes, puis le détail des structures de données utilisées. Nous décrivons aussi l'algorithme utilisé dans le programme principal. Par la suite, nous indiquerons comment le modèle proposé pourrait être utilisé pour supporter facilement la conversion de plusieurs protocoles. Enfin, en guise de mise en œuvre, nous l'appliquerons à quelques cas de conversion.

4.1.1 Diagramme de classes

Nous avons implémenté notre convertisseur en langage C++ dans un modèle orienté objet intégrant 13 classes. La Figure 4.1 présente une vue globale du diagramme UML de ces différentes classes. Comme on peut le constater, tous les modules définis au chapitre 3 y sont présents sous forme de classe.

Principalement, on y retrouve les classes suivantes : *Protocol,* qui définit la structure de l'objet *Protocole* à utiliser, *ProtocolAnalyser* qui implémente les méthodes pour identifier la convertibilité des protocoles considérés, *ProtConversionModule* qui réalise la conversion proprement dite de deux protocoles quelconques, *ServiceConverter* qui regroupe les méthodes utilisées pour la conversion des services à chaque couche OSI ainsi que certaines fonctions utiles, *BDService* qui regroupe les méthodes et attributs relatifs à la base de données de service.

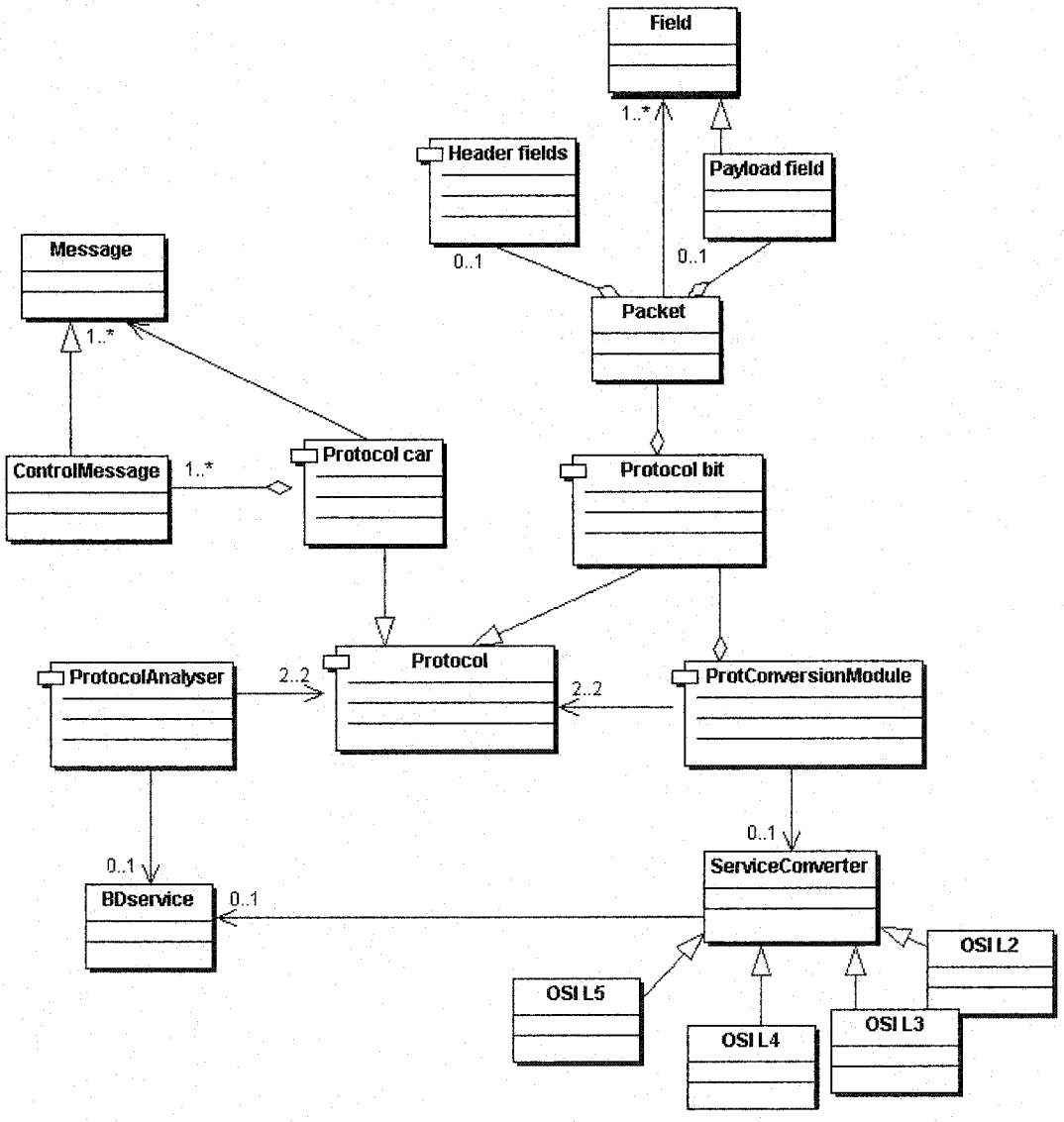


Figure 4.1 Diagramme UML des différentes classes

La classe *Protocol* est une superclasse dont hérite les classes *Protocol_bit* et *Protocol_car* qui regroupent respectivement les attributs et opérations relatifs aux protocoles orientés bit et aux protocoles orientés caractère. Pour les protocoles bit, les classes *Packet*, *Field*, *Header_fields* et *Payload_field* définissent respectivement les structures et opérations sur les paquets, les champs d'un paquet, les champs de contrôle et ceux des données utiles. Enfin, les classes *Message* et *ControlMessage* définissent les types d'information manipulés par les protocoles orientés caractère. *ControlMessage* est une classe dérivée de la classe *Message* et est utilisée pour les messages de contrôle du protocole.

Chacune des classes définies possède, outre ses attributs et opérations associés, un constructeur pour l'initialisation des variables et un destructeur qui libère la mémoire allouée aux attributs de la classe. Le programme principal défini dans une fonction *main()* appelle un objet de type *ProtocolAnalyser* pour l'analyse des protocoles, puis une instance de la classe *ProtConversionModule* pour la conversion. Il reçoit en argument deux protocoles dont les définitions sont incluses dans des fichiers de même nom, ainsi que les paquets à convertir que nous supposons initialement contenus dans un fichier.

4.1.2 Implémentation des classes et structures de données

Dans cette section, nous donnons un bref aperçu des attributs et principales méthodes de chacune des classes de notre modèle.

- **Classe *Protocol***

Elle comprend les principaux attributs d'un objet *Protocol* tel que présenté ci haut (voir Figure 4.1), et les méthodes qui serviront à manipuler les protocoles. Les principales méthodes cette classe sont :

- *get_primary_serv( )* qui retourne les fonctions primaires du protocole ;
- *get_secondary_serv( )* qui retourne les fonctions secondaires du protocole ;
- *get_all_serv( )* qui retourne tous les services du protocole ;

- *get_Osi_Layer( )* qui retourne la couche OSI du protocole ;
- *find_Common_S( )* qui identifie les services communs avec un autre protocole;
- *find_Common_PrimaryS( )* qui identifie les services primaires communs aux protocoles.

Les trois premières fonctions retournent des pointeurs sur des chaînes de caractères du type char**. Nous avons choisi cette structure car elle permet d'allouer dynamiquement la mémoire et de la libérer après, selon le nombre d'éléments que nous aurons identifiés.

La fonction de détermination des services primaires *find_Common_PrimaryS()* utilise en entrée un objet *Protocol P*, représentant le protocole de destination. Elle retourne l'ensemble des services primaires communs au protocole actuel et l'objet *P*, sous la forme d'un pointeur sur des chaînes de caractères (char** en C++).

La fonction *find_Common_S( )* reçoit en entée un objet *Protocol P* identifiant le protocole selon lequel on veut déterminer les services communs, et un objet *BDservice* qui est une instance de la base de données de service. Elle retourne, d'une part, les services communs aux protocoles, et d'autre part, les correspondances entre les positions des champs de contrôle permettant d'offrir ces services communs, sous la forme d'un pointeur à deux dimensions sur des entiers (int**). Par exemple, si le champ *checksum* du premier protocole est à la position 5, et qu'il existe aussi à la position 8 dans le second protocole, on aurait, à une ligne *i* de notre table de correspondance : tab[*i*][0] = 5 et tab[*i*][1] = 8.

- **Classe *Protocol_bit***

Elle hérite de la classe *Protocol* et contient les attributs et opérations propres aux protocoles orientés bit. Les attributs propres à cette classe sont:

nb_packet : le nombre de paquets du protocole ;

mypacket : la définition des différents paquets du protocole.

La variable *nb_packet* est entière, alors que *mypacket* est un pointeur vers un ensemble d'objets de type *Packet* (Packet*). Les méthodes suivantes ont été définies:

- *addpkt( )* qui ajoute au protocole un paquet dont la définition est contenue dans un fichier ;
- *extract_pkt( )* qui extrait les valeurs des champs d'un protocole à partir d'un paquet ;
- *find_pkt_ind( )* qui trouve l'indice correspondant à un type de paquet donné dans la liste des paquets du protocole ;
- *get_Packet( )* qui retourne un paquet correspondant à un indice ou à un nom donné ;
- *get_nb_pkt( )* qui retourne le nombre de paquets du protocole.

La fonction *addpkt()* reçoit en entrée le nom du fichier contenant la définition du paquet et retourne une valeur booléenne (*vrai* ou *faux*) indiquant si l'ajout a été bien réalisé. La méthode *extract_pkt()* reçoit en entrée le nom du paquet (important si le protocole dispose de plusieurs types de paquets), ainsi que la chaîne de caractères représentant le paquet à extraire, et retourne une valeur booléenne indiquant si l'extraction du paquet a été bien réalisée.

La méthode *get_Packet()* reçoit en entrée une valeur entière correspondant à l'indice d'un paquet ou une chaîne de caractères correspondant au nom du paquet, et retourne le format du paquet sous la forme d'un pointeur vers un objet de la classe *Packet*.

- **Classe *Protocol_car***

De la même manière que la classe *Protocol_bit*, elle hérite aussi de la superclasse *Protocol,* mais spécifie plutôt les attributs et opérations relatives aux protocoles orientés caractère. Les principaux attributs sont :

nb_msg_ctrl : le nombre de messages de contrôle du protocole ;
msg_ctrl : liste des champs de contrôle dans l'ordre ;
msg_data : les données utiles encapsulées dans les messages de contrôle.

La variable *msg_data* est un pointeur vers un objet de type *Message*, alors que *msg_ctrl* est un pointeur vers un ensemble d'objets de type *ControlMessage*. Les méthodes qui y sont définies sont :

- *get_nb_message( )* qui retourne le nombre de messages de contrôle du protocole ;
- *addmessage( )* qui charge la définition des différents types de messages du protocole à partir d'un fichier reçu en paramètre ;
- *get_ControlMessage( )* qui retourne un message de contrôle correspondant à un indice donné reçu en paramètre ;
- *get_lentot( )* qui calcule et retourne la longueur d'un message reçu en paramètre.

Il faut préciser que les messages manipulés par les trois premières fonctions sont des objets de type *Message* que nous définirons par la suite. Les deux classes, que nous venons de décrire, qui dérivent de *Protocol,* peuvent utiliser aussi les méthodes et attributs de cette dernière. La Figure 4.2 montre les attributs et méthodes des classes *Protocol*, *Protocol_bit* et *Protocol_car*.

- **Classe *Message***

La classe *Message* définit la structure des messages manipulées par les protocoles dans les hautes couches. Les principaux attributs sont :

Msg_min_size : la taille minimale du message ;
Msg_max_size : la taille maximale du message ;
Msg_size : la taille du message ;
Msg_name : le nom du message ;
Value : la valeur du message ;
Order : ordre du message, indique le nombre de messages de contrôle envoyés avant ce message-ci.

Le nom et la valeur sont des chaînes de caractères, tandis que les autres attributs sont des valeurs entières. Les méthodes définies sont *setmessage_value()*,

getmessage_value(), *getmessage_len()* qui permettent respectivement de mettre à jour, de retourner la valeur du message, ou d'en obtenir la longueur.

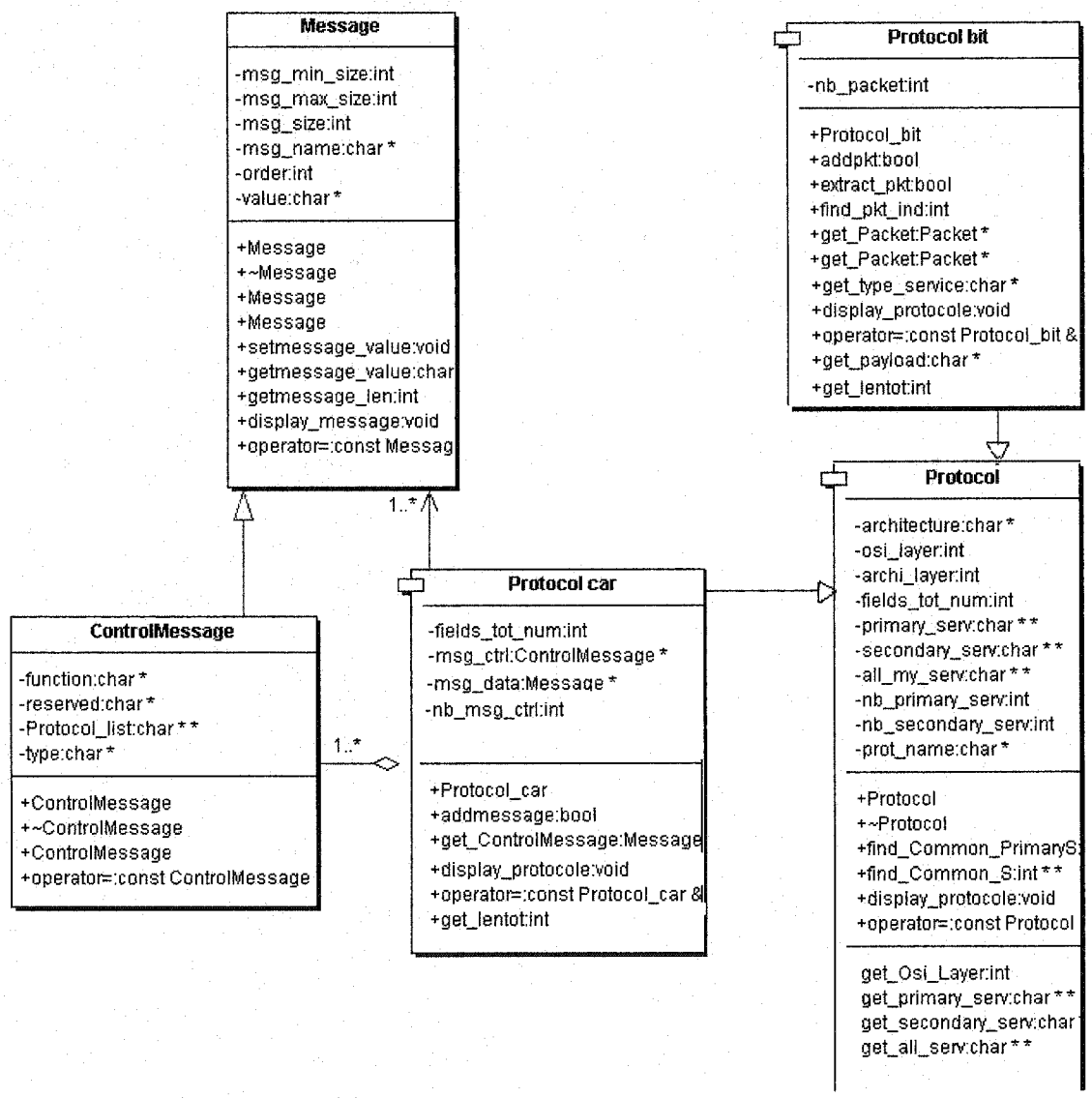


Figure 4.2 Structure des classes *Protocol*, *Protocol_bit*, *Protocol_car* et *Message*

- **Classe *Packet***

Utilisée par les protocoles orientés bit, la classe *Paquet* est constituée d'un ensemble de champs d'entête, de charge utile (payload) et possiblement d'une queue (champs après la charge utile). Cette classe dispose des attributs privés suivants :

lnkheader_fields : l'ensemble des champs d'entête (type Header_fields *) ;
trailer : l'ensemble des champs de queue (type Header_fields *) ;
lnkpayload_field : le champ contenant les données utiles du paquet ;
pkt_name : le nom du paquet ;
pkt_min_size : la longueur minimale du paquet ;
pkt_max_size : la longueur maximale du paquet ;
nb_fields : le nombre de champs du paquet ;
pktfields : tous les champs du paquet dans l'ordre.

Les variables *lnkheader_fields* et *trailer* sont des pointeurs vers des objets de type *Header_fields*, alors que *pktfields*, de type Field*, est un pointeur vers un ensemble d'objets de type *Field*. Les valeurs de *pkt_min_size*, *pkt_max_size*, *nb_fields* sont entières. Les méthodes définies dans la classe *Packet* sont les suivantes :

- *setpkt_min_size( )* qui attribue une taille minimale à un paquet ;
- *setpkt_max_size( )* qui attribue une taille maximale à un paquet ;
- *setpkt_name( )* qui attribue un nom (entré en paramètre) au paquet ;
- *setnbfields( )* qui attribue une valeur au nombre de champs du paquet ;
- *setfields( )* qui copie dans le paquet la liste des champs entrée en paramètre ;
- *getnbfields( )* qui retourne le nombre de champs de paquet ;
- *getpkt_min_size( )* qui retourne la taille minimale du paquet ;
- *getpkt_max_size( )* qui retourne la taille maximale du paquet ;
- *getpkt_name( )* qui retourne le nom du paquet ;
- *setpktheader( )* qui attribue au paquet l'entête passée en paramètre ;
- *settrailer( )* qui attribue au paquet la queue passée en paramètre ;
- *addfield( )* qui ajoute un champ au paquet à l'indice spécifié ;
- *getfield( )* qui retourne le champ à l'indice spécifié ;
- *getfields( )* qui retourne tous les champs du paquet pointeur vers un ensemble d'objets de type *Field* ;
- *setpktpayload( )* qui attribue au paquet les données de charge utile entrées en paramètre comme un objet de type *PayloadField* ;

- *getpayload( )* qui retourne la charge utile du paquet ;
- *getheader( )* qui retourne l'entête du paquet ;
- *gettrailer( )* qui retourne la queue du paquet ;
- *getpkt_len( )* qui calcule et retourne la longueur actuelle du paquet ;
- *getheader_len( )* qui calcule et retourne la longueur de l'entête ;
- *getpayload_len( )* qui calcule et retourne la longueur de la charge utile ;
- *gettrailer_len( )* qui calcule et retourne la longueur de la queue ;
- *has_header( )* qui indique si l'entête est vide et retourne une valeur booléenne : *faux* si l'entête est vide et *vrai* dans le cas contraire ;
- *has_payload( )* qui indique si la charge utile est vide ;
- *has_trailer( )* qui indique s'il y a une queue dans le paquet, i.e. des champs de contrôle après les données utiles ;
- *resetpkt( )* qui permet de réinitialiser tous les champs du paquet ;
- *extract_pkt( )* utilisée pour extraire d'une chaîne de caractères les valeurs des différents champs du paquet.

- **Classe *Field***

Elle définit les attributs et opérations possibles sur un champ dans un paquet. Les différents attributs sont:

field_name : le nom du champ ;

field_value : la valeur du champ ;

field_type : le type du champ, indique s'il s'agit d'un champ d'entête ou de queue, et précise si le champ a une valeur fixe ou variable ;

service : le service offert par le champ;

field_pos : la position du champ dans le format du paquet ;

field_size : la taille du champ (i.e. le nombre de bits alloués).

Les variables *field_name*, *field_value*, *field_type* et *service* sont des chaînes de caractères alors que *field_pos* et *field_size* sont des valeurs entières. Quant à la variable *field_type*, elle est composée de trois caractères dont le premier indique s'il s'agit d'un champ

d'entête, de données ou de queue (H, P ou T) et le troisième indique si sa valeur est fixe ou variable (S pour les champs à valeur fixe). Les opérations possibles dans cette classe sont:

- *setfieldname( )* qui attribue le nom entré en paramètre au champ ;
- *setfieldvalue( )* qui attribue la valeur entrée en paramètre au champ ;
- *setfield( )* qui copie dans le champ actuel les valeurs des attributs du champ passé en paramètre ;
- *getfield( )* qui retourne une copie du champ ;
- *getfieldname( )* qui retourne le nom du champ ;
- *getservice( )* qui retourne le service offert par le champ ;
- *getfieldtype( )* qui retourne le type du champ ;
- *getfieldsize( )* qui retourne la taille du champ (nombre de bits alloués).

La classe *PayloadField* dérivée de *Field* dispose, en plus, des variables *field_max_size* et *field_min_size* indiquant respectivement la taille maximale et la taille minimale des données utiles, ainsi que des méthodes *setpayload(), getpayload(), getpayload_len(), extract_payload()* utilisées respectivement pour attribuer une valeur, obtenir le contenu, la longueur du champ des données utiles ou extraire les données utiles d'une chaîne de caractères représentant un paquet.

- **Classe *Header_fields***

Elle permet de stocker dans une structure un ensemble de champs qui se suivent dans le paquet. Ainsi, on l'utilise pour les champs de l'entête ainsi que ceux de la queue. Ses attributs sont le nombre de champs *nb_fields*, ainsi que l'ensemble des champs *lnkField* (de type Field *). Les principales méthodes sont : *getheader()* qui retourne la liste des champs enregistrés, *extract_h()* qui extrait les champs d'une chaîne de caractère, *get_nbfields()* qui retourne le nombre de champs, et *addfield()* pour ajouter un champ.

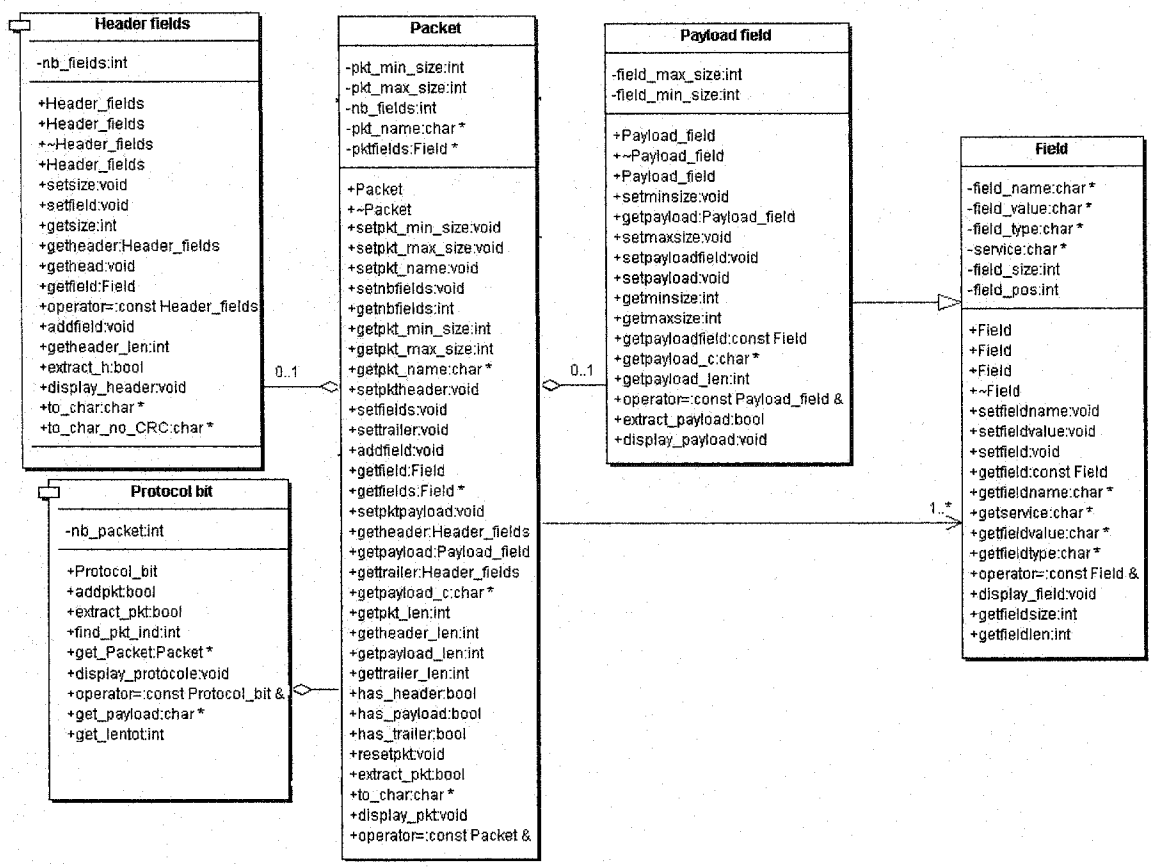


Figure 4.3 Structure des classes *Field, Header_Fields, PayloadField* et *Packet*

- **Classe *ProtConversionModule***

C'est la classe qui contient les fonctions de conversions des protocoles. Elle possède comme attributs deux objets de type *Protocol* identifiant les deux protocoles à convertir, respectivement *myProt1* et *myProt2*, ainsi qu'un objet *myServiceConverter* de type ServiceConverter représentant notre convertisseur de service. Elle dispose de deux méthodes par défaut pour la conversion de deux protocoles quelconques *Convert_1_to_2( )* et *Convert_2_to_1( )*, selon que l'on convertit du premier protocole vers le second ou vice-versa. Ces méthodes reçoivent en entrée le paquet à convertir sous forme de chaîne de caractères, ainsi que la liste de correspondance entre les champs convertibles des deux protocoles. Elles retournent une chaîne de caractères représentant

le paquet converti. Une fonction *set_defaultvalue( )* permet d'attribuer une valeur par défaut à un champ et reçoit en entrée un objet de type *Field*.

- **Classe *ProtocolAnalyser***

Cette classe a pour attributs deux objets de type *Protocol*, soit *protocole1* et *protocole2* représentant les protocoles à convertir, un objet de type *BDservice*, et un double pointeur d'entiers *mapping_tab* (type int**) identifiant les positions des champs convertibles entre ces protocoles. Les méthodes *load_first_prot()* et *load_second_prot()* permettent de lire dans un fichier la définition de chaque protocole, *pkt_is_convertible( )* détermine la convertibilité de deux types de paquets et *prot_is_convertible( )* identifie la convertibilité des protocoles. La méthode *pkt_is_convertible( )* reçoit en entrée les noms des types de paquets et retourne une valeur booléenne indiquant s'ils sont convertibles. Enfin, une fonction *get_Convertible_Fields_List( )* permet de récupérer la liste des champs convertibles entre les protocoles.

- **Classe *ServiceConverter***

Elle est utilisée pour implémenter les fonctions de conversion entre les champs offrant des services convertibles. Quelques méthodes y sont définies comme *convert_generic( )* qui est une fonction de conversion générique entre deux champs quelconques, pour convertir *compute_pkt_len( )* qui calcule la longueur d'un paquet, *int2str( )* pour convertir un entier en chaîne de caractères, *compute_hdr_len( )* pour obtenir la longueur de l'entête et enfin *convert_FCS16_to_FCS32( )*. D'autres méthodes comme *convert_IP4_to_IP6( ), convert_IP6_to_IP4( )* peuvent être utilisées pour la conversion des adresses IP par exemple.

- **Classe *BDservice***

Elle contient les attributs et opérations associées à la base de données de service. Les attributs *BD_serv_type* et *BD_conv_serv* sont des tableaux de chaînes de caractères à deux dimensions qui maintiennent respectivement les associations entre les services et

leur type, et entre les services convertibles. Les méthodes *load_BD( )* pour charger les données d'un fichier, *get_type_service( )* pour obtenir le type associé à un service donné, et *serv_is_convertible( )* pour identifier la convertibilité de deux services quelconques entrés en paramètre y sont définies.

La Figure 4.4 illustre la structure interne des classes *BDservice*, *ServiceConverter*, *ProtocolAnalyser* et *ProtConversionModule*.

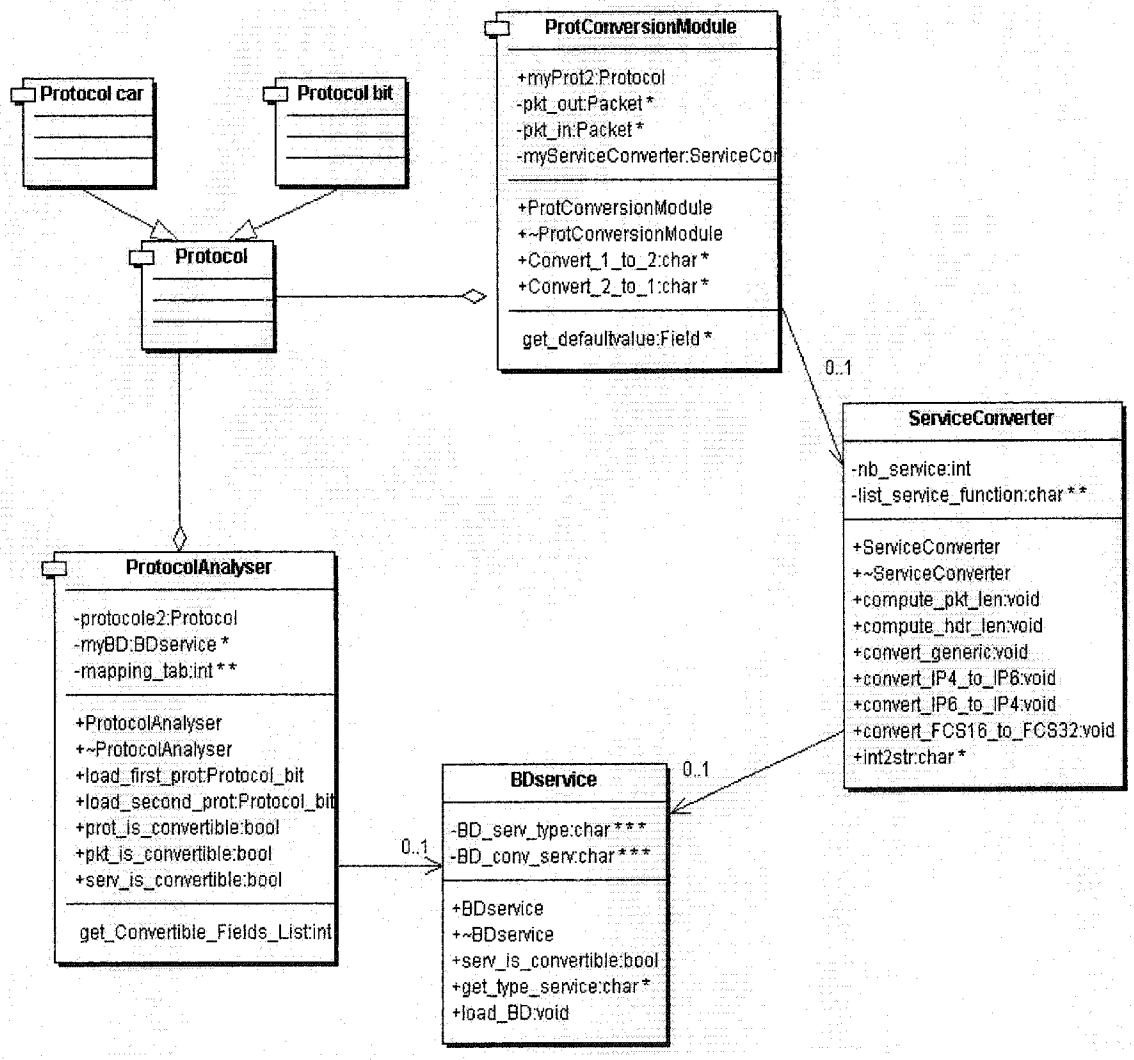


Figure 4.4 Structure interne des classes principales du convertisseur

4.1.3 Algorithme du programme principal

Le programme principal coordonne les interactions entre les différents modules de notre convertisseur. Il est défini dans une fonction *main( )* appelé au début du programme avec comme arguments les deux protocoles à convertir.

Il utilise deux objets *Protocol*, *P1* et *P2*, représentant les protocoles à convertir, une variable *tab_pkt* contenant les paquets à convertir éventuellement chargés d'un fichier, un objet *myAnalyser* de type *ProtocolAnalyser*, un objet *myConvModule* de type *ProtConversionModule* et un tableau d'entiers à deux dimensions, *f_list*, pour maintenir les correspondances entre les champs convertibles. La Figure 4.5 décrit l'algorithme utilisé dans cette fonction.

```
DEBUT
    Initialiser les variables: P1, P2, f_list, tab_pkt, myAnalyser, myConvModule
    Lire les définitions de P1 et P2 dans les fichiers d'entrée.
    Déterminer leur convertibilité (myAnalyser : prot_is_convertible()).
    Si P1 et P2 ne sont pas convertibles, Alors
        Afficher message de fin. Retourner. FIN.
    Sinon
        Obtenir f_list (myAnalyser : get_Convertible_Fields_List()).
    Fin Si.
    Pour chaque paquet à convertir
        Si il existe une fonction spécifique de conversion P1-P2, Alors
            Convertir le paquet avec cette fonction spécifique
            (myConvModule: Convert_P1_to_P2 (paquet, f_list )).
            Retourner le paquet converti.
```

Figure 4.5 Algorithme de la fonction principale

4.1.4 Mise en œuvre du convertisseur

Après avoir exposé les détails d'implémentation de notre convertisseur, nous montrons dans cette section comment il pourrait être utilisé pour supporter la conversion de plusieurs protocoles.

Nous choisissons de réaliser la conversion de Ethernet à FDDI. Nous utiliserons les formats SNAP, i.e. Ethernet II et FDDI SNAP. Ces deux protocoles offrent des services similaires et, comme nous le verrons dans la suite, ils sont prouvés convertibles par notre algorithme. La plupart des services sont supportés dans la conversion. Nous utilisons alors la fonction de conversion par défaut. Les champs qui ne figurent pas dans les deux protocoles et qui ne contiennent pas des valeurs calculables dans le convertisseur de service (comme la longueur de l'entête ou du paquet, ou le checksum qui sont calculables), se verront attribuer des valeurs par défaut. C'est le moment de préciser que lorsqu'on se retrouve avec des protocoles pour lesquels certains champs importants doivent recevoir des valeurs par défaut, on pourrait mettre à jour le convertisseur de service en incluant une fonction pour définir la valeur à inclure dans ce champ.

De même, nous utilisons notre convertisseur pour réaliser la conversion entre les protocoles Ethernet et Token-Ring. Nous utilisons cette fois ci le format 802.3. Pour vérifier l'exactitude des résultats fournis par notre convertisseur, nous réalisons une fonction de conversion simple qui permet de convertir un paquet Ethernet-SNAP en FDDI-SNAP d'une part, et une autre qui réalise la conversion de Ethernet 802.3 en Token-Ring. Dans les deux cas, les valeurs des paquets convertis, obtenus à l'aide du convertisseur générique sont identiques à celles obtenues à l'aide des fonctions de conversions simples. Nous reviendrons à la section 4.3 sur notre convertisseur de protocole pour évaluer sa performance. Mais avant, nous détaillerons, dans la section suivante, l'implémentation que nous avons réalisée pour notre algorithme de convertibilité au niveau de la classe *ProtocolAnalyser* et montrerons quelques résultats.

4.2 Application de notre algorithme de convertibilité

Dans cette section, nous présentons d'abord quelques aspects relatifs à l'application de notre algorithme. Par la suite, nous montrons et analysons les résultats obtenus.

4.2.1 Détails d'application de l'algorithme

Pour montrer l'efficacité de la méthode proposée au chapitre 3 pour l'identification de la convertibilité, nous allons l'appliquer à un ensemble de protocoles d'usage courant et analyser les résultats. Les protocoles considérés sont : IPv4, IPv6, Ethernet (802.3 et SNAP), FDDI (SNAP), ATM, X25PLP, TCP, NSP, UDP et Token-Ring.

La diversité des services offerts dans les réseaux nous a conduit à prédéfinir un ensemble de services à chaque couche en se basant sur les services fréquemment rencontrés au niveau des protocoles opérant à ces couches là. Les tableaux 4.1 à 4.3 résument les différents éléments de service prédéfinis que nous utiliserons.

Tableau 4.1 Définition des services à la couche 2 du modèle OSI

Type service	Services associés	Signification
PHY_ADDR	DST_ADDR_MAC, SRC_ADDR_MAC	Adressage MAC
	DST_1394_NODE_ID, SRC_1394_NODE_ID	Adressage Firewire
	DST_ADDR_VAR8, DST_ADDR_BCAST	Adresse variant à chaque nœud, Adresse Broadcast
PKT_DELIMIT	START_DEL, END_DEL, PREAMBLE	Délimitation du paquet
ERR_CTRL_DATA	HDR_CHK, DATA_CHK, PKT_CHK	Contrôle d'erreur des données
LENGTH	HDR_LEN, DATA_LEN, PKT_LEN	Longueur entête, des données ou du paquet
OTHER	PKT_PRIO	Priorité du paquet
	UP_PROTID	Protocole encapsulé
	FLOW_ID	Identificateur du flot
	CTRL, RESERVED, TRANS_CODE, DSAP, SSAP, OUICODE	Contrôle, réservé, code de transaction, Champs SNAP

Nous nous sommes limités à la couche 4 tout d'abord parce que la plupart des protocoles considérés opèrent dans les basses couches, mais aussi parce qu'au-delà de cette couche, les types de services rencontrés dépendent de l'application opérant au plus haut niveau et sont assez diversifiés. Le Tableau 4.1 résume les services prédéfinis à la couche 2, les tableaux 4.2 et 4.3 montrent les services définis aux couches 3 et 4 respectivement.

Tableau 4.2 Définition des services à la couche 3 du modèle OSI

Type service	Services associés	Signification
NET_SRC_ADDR NET_DST_ADDR	DST_ADDR_IP4, SRC_ADDR_IP4,	Adressage IPv4
	SRC_ADDR_IP6, SRC_ADDR_IP6	Adressage IPv6
	SRC_ADDR_X121, DST_ADDR_X121	Adressage X.121
	ADDR_VC	Adressage circuit virtuel
	SRC_ADDR_DEC, DST_ADDR_DEC	Adressage type DECNET
FRGMT	OFFSET, FLOW_ID, MSG_LEN,	Informations de fragmentation
ERR_CTRL_DATA	HDR_CHK, DATA_CHK, PKT_CHK	Contrôle d'erreur de données
LENGTH	HDR_LEN, DATA_LEN, PKT_LEN	Longueur entête, données ou du paquet
ACK	ACK_NUM, SEQ_NUM,	Champs d'accusé de réception
OTHER	PKT_PRIO	Priorité du paquet
	UP_PROTID	Protocole encapsulé
	DONT_FRGMT, MORE_FRGMT	Bit indiquant la fragmentation
	CTRL, RESERVED,	Champs de contrôle, réservé
	PKT_LIFETIME, TOS, PKT_TYPE	Durée de vie, type de service, type de paquet

Pour pouvoir bien appliquer notre algorithme de convertibilité, il faut associer à chaque service, éventuellement, un type de service si possible. Dans le cas particulier de l'adressage, les types suivants ont été définis :

- PHY_SRC_ADDR, PHY_DST_ADDR : adressage physique de noeud source et destination respectivement ;
- NET_SRC_ADDR, NET_DST_ADDR : adressage logique réseau de nœud source et destination respectivement ;
- PROC_SRC_ADDR, PROC_DST_ADDR : adressage de processus source et destination.

Tableau 4.3 Définition des services à la couche 4 du modèle OSI

Type service	Services associés	Signification
PROC_SRC_ADDR PROC_DST_ADDR	SRC_ADDR_PORT	Adresse de port source
	DST_ADDR_PORT	Adresse de port destination
	ADDR_VC	Adressage circuit virtuel
FLOW_CTRL	WINDOW	Taille de fenêtre utilisée
ERR_CTRL_DATA	HDR_CHK, DATA_CHK, PKT_CHK	Contrôle d'erreur de données
LENGTH	HDR_LEN, DATA_LEN, PKT_LEN	Long. entête, données, paquet
ACK	ACK_NUM, SEQ_NUM	Numéro : ACK, séquence
OTHER	URG_BIT, ACK_BIT, PSH_BIT, RST_BIT, SYN_BIT, FIN_BIT	Bit indiquant l'urgence, ACK, reset, synchronisation, fin
	CTRL, RESERVED,	Champs de contrôle, réservé

Les tableaux 4.1 à 4.3 montrent aussi les associations entre les services et leur type. Pour chaque type d'adressage sont indiqués les éléments de service possibles associés. Pour nous aider dans la détermination des services primaires, nous en avons identifié un certain nombre à chaque couche, en suivant la méthodologie présentée au chapitre 3.

Tableau 4.4 Définition des services primaires

Couche	Services primaires	Signification
Couche 2	PHY_SRC_ADDR	Adresse physique de nœud source
	PHY_DST_ADDR	Adresse physique de nœud destination
	FRGMT	Fragmentation
	ERR_CTRL_DATA	Contrôle d'erreur des données du paquet
	VIRTUAL_CIRC	Service circuit virtuel
	DATAGRAM	Service datagramme
	ERR_CTRL_PKT (ou ACK)	Contrôle d'erreur des paquets (perte)
Couche 3	NET_SRC_ADDR	Adresse logique nœud source
	NET_DST_ADDR	Adresse logique nœud destination
	FRGMT, ERR_CTRL_DATA, VIRTUAL_CIRC, DATAGRAM, ACK	Idem qu'à la couche 2
Couche 4	PROC_SRC_ADDR	Adresse de processus source
	PROC_DST_ADDR	Adresse de processus destination
	FRGMT, ERR_CTRL_DATA, ERR_CTRL_PKT VIRTUAL_CIRC, DATAGRAM	Idem qu'à la couche 2

Le Tableau 4.4 résume quelques services primaires prédéfinis à partir de la couche 2. Notre avons choisi les paires de protocoles suivants: (IPv4, X.25PLP), (IPv4, IPv6), (Ethernet802.3, FDDI-SNAP), (FDDI-SNAP, Ethernet-SNAP), (Ethernet 802.3, ATM), (TCP, NSP), (UDP, TCP), (Ethernet 802.3, Token-Ring). Les résultats obtenus sont présentés au Tableau 4.5. On y retrouve, pour chaque paire de protocoles, le résultat de notre algorithme de convertibilité, i.e. si les protocoles sont convertibles ou pas. Les services qui sont supportés après la conversion sont retournés, de même que ceux qui ne sont pas convertibles.

4.2.2 Analyse des résultats

L'algorithme termine par un échec pour les protocoles IPv4 et X25. On peut voir que l'adressage qui est un service primaire n'est pas supporté. De même, le mode de fonctionnement est différent entre les deux protocoles, l'un utilisant un service datagramme sans ACK, et l'autre fonctionnant en mode circuit virtuel avec ACK. En pratique, ces protocoles ne sont pas convertibles, de sorte que le seul moyen pour résoudre l'interconnexion des réseaux IP via les réseaux publics X25 est d'utiliser un mécanisme d'encapsulation des paquets IP dans la charge utile de X25, et on décapsule à destination. On ne modifie donc ni les paquets IP, ni les paquets X25.

Considérant la paire IPv4 et IPv6, on peut voir que tous les services primaires sont supportés, en conséquence les protocoles sont convertibles. On sait que IPv6 est une extension de IPv4, et supporte donc tous les services de ce dernier. La conversion des services entre ces deux protocoles a même été discutée dans les RFC 2529 et 3142.

De même, on peut voir que la conversion est possible pour IPv4 et CLNP. Ce dernier est le protocole de la couche réseau spécifié par l'ISO. En effet, il a de grandes ressemblances avec IP au niveau du format et du mode de fonctionnement et offre quasiment les mêmes services que ce dernier.

Tableau 4.5 Résultats de l'application de l'algorithme de convertibilité

Protocole 1	Protocole 2	Résultat	Services supportés	Services non convertibles
IPv4	X.25 PLP	CNP	-------	NET_SRC_ADDR NET_DST_ADDR
IPv4	IPv6	CP	NET_SRC_ADDR, UP_PROTID NET_DST_ADDR, FLOW_ID, TOS PKT_LIFETIME.	------
Ethernet 802.3	ATM	CNP*	-------	PHY_SRC_ADDR PHY_DST_ADDR
Ethernet-SNAP	FDDI-SNAP	CP	DST_ADDR_MAC_48, CTRL SRC_ADDR_MAC_48, UP_PROTID ERR_CTRL_DATA, DSAP, SSAP	------
Ethernet 802.3	Token-Ring	CP	DST_ADDR_MAC_48, SRC_ADDR_MAC_48, ERR_CTRL_DATA	ACESS_CTRL, FR_STATUS
TCP	UDP	CP* TCP→UDP	L4_SRC_ADDR, L4_DST_ADDR, ERR_CTRL_DATA	ACK
TCP	NSP	CP	PROC_SRC_ADDR, PROC_DST_ADDR ERR_CTRL_DATA, ERR_CTRL_PKT	-------
IPv4	CLNP	CP	NET_SRC_ADDR, NET_DST_ADDR, FRGMT, HDR_CHK, PKT_LIFETIME	------

La convertibilité entre Ethernet et ATM a déjà été discutée antérieurement. Comme on peut le voir dans le Tableau 4.5, ils ne sont pas convertibles directement dû essentiellement aux différences d'adressage et de mode de fonctionnement, cependant ils peuvent l'être si l'on implémente une fonction de complémentation permettant de résoudre l'adressage entre les deux protocoles, car alors les services primaires communs seraient convertibles.

Les protocoles Ethernet-SNAP et FDDI-SNAP sont assez similaires du point de vue des services offerts. Comme on peut le constater, notre méthode montre qu'ils sont convertibles. De plus, la plupart des services sont supportés dans la conversion. De même, il se termine par un succès pour Ethernet et Token-Ring, ce qui reflète bien la réalité puisque les protocoles LAN Ethernet, Token-Ring et FDDI sont convertibles

entre eux et il existe des relais qui permettent de réaliser l'interconnexion de ces différents types de réseaux.

Les protocoles TCP et NSP (de DECNET) offrant des services similaires sont prouvés convertibles. On peut voir que dans le cas UDP et TCP, la conversion n'est pas réalisable dans les deux sens. Ceci est dû essentiellement aux différences de modes de fonctionnement de ces deux protocoles. On peut juste opérer dans les sens de TCP vers UDP, l'ensemble des services primaires de UDP étant inclus dans celui de TCP.

Aux hautes couches, comme nous l'avions mentionné, les services primaires dépendent du protocole et de l'application considérés; il faut considérer chaque type d'application et étudier les services de protocole. En considérant comme application la messagerie électronique, l'étude des deux protocoles SMTP et X.400 nous a montré que des informations de services primaires identiques se retrouvent dans chaque protocole. En pratique, il existe une passerelle applicative permettant de convertir ces deux systèmes de messagerie.

On peut donc déduire que la méthode proposée permet d'identifier de manière adéquate la convertibilité de deux protocoles quelconques en se basant sur les services offerts.

4.3 Analyse de performance

Dans cette section, nous évaluons la performance du modèle générique proposé par rapport à un modèle simple traditionnel de convertisseur de deux protocoles bien définis. Nous avons instancié un convertisseur de protocole de FDDI à Ethernet, puis un autre de Token-Ring à Ethernet. Cela nous permet d'évaluer l'impact du modèle proposé par rapport aux modèles classiques de convertisseur spécifique. Le principal paramètre considéré est le délai de conversion des paquets.

4.3.1 Influence du modèle générique

Pour chaque cas de conversion, nous avons considéré des situations où se présentent respectivement 1000, 10.000, 50.000 et 100.000 paquets dans la file d'attente du convertisseur. Nous avons pris les mesures aussi bien au niveau du convertisseur simple qu'au niveau du convertisseur générique proposé.

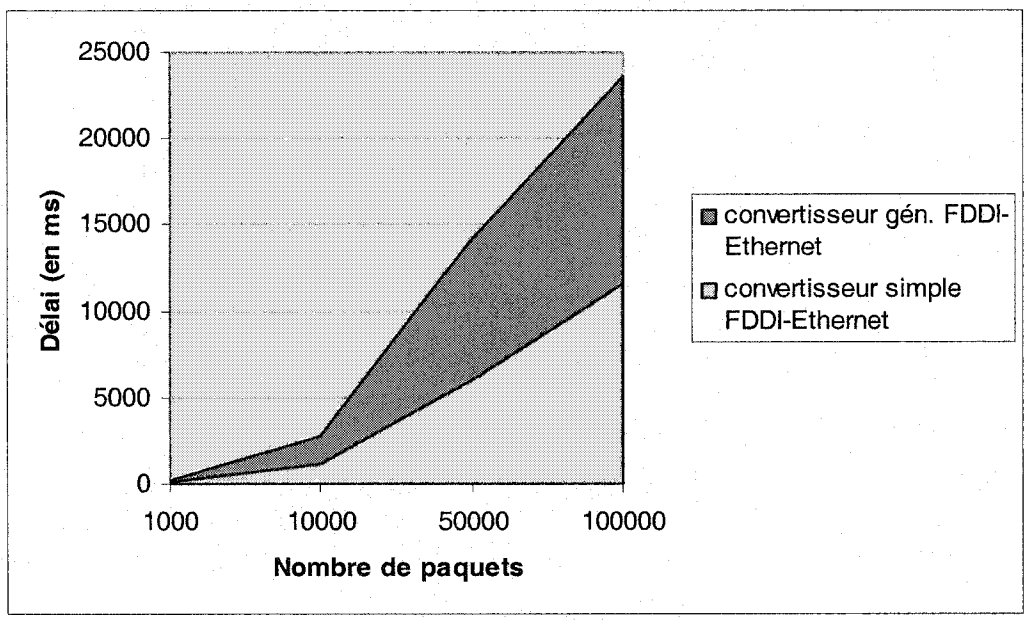


Figure 4.6 Délai de conversion des paquets (FDDI--Ethernet)

Les figures 4.6 et 4.7 montrent les résultats obtenus dans le cas de la conversion de FDDI à Ethernet et Token-Ring à Ethernet respectivement. On remarque que, pour chaque cas de conversion, un délai supplémentaire est induit au niveau du convertisseur générique. Ceci est normal et s'explique par le fait que le convertisseur générique dispose de plusieurs structures et fonctions, crée plusieurs objets, réalise plusieurs opérations de test, et passe dans plusieurs boucles avant de réaliser la conversion d'un champ. Le modèle générique fait gagner en flexibilité (on peut supporter plusieurs protocoles), mais fait perdre en performance. Cependant, des espoirs de gain en performance sont encore permis quand on sait que les processeurs réseaux peuvent intégrer plusieurs processeurs travaillant en parallèle. Notre modèle a été conçu pour

opérer dans ces types de processeurs.

Dans la prochaine section, nous évaluerons notre modèle sur des machines incluant plusieurs processeurs.

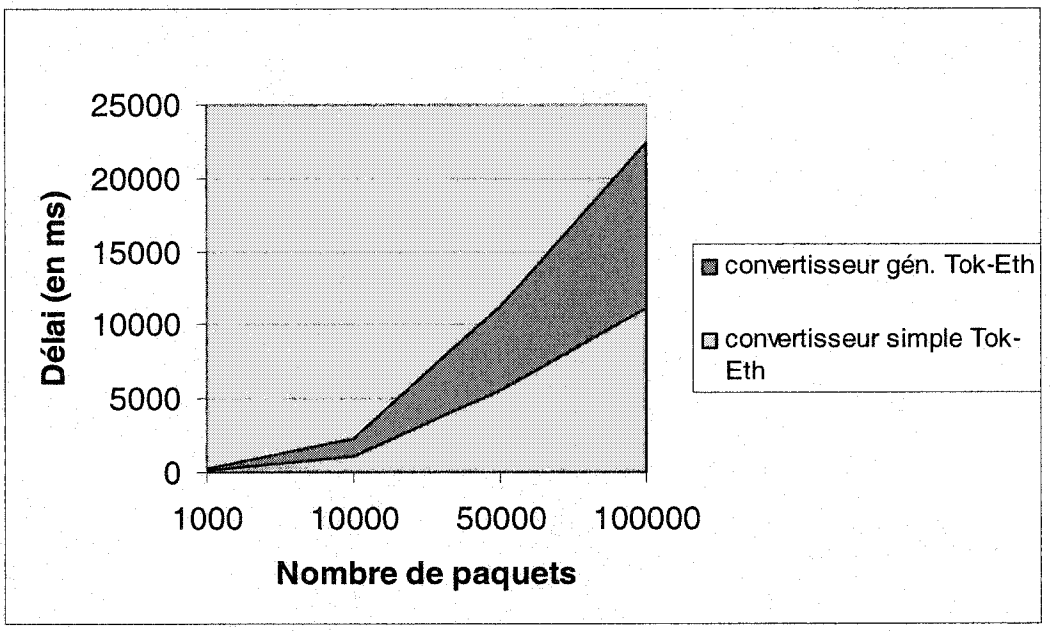


Figure 4.7 Délai de conversion des paquets (Token-Ring à Ethernet)

4.3.2 Architecture à processeurs multiples

Les unités de traitement à processeurs multiples offrent généralement des possibilités de traitement parallèle des données et, ce faisant, peuvent considérablement diminuer les temps d'exécution des programmes par rapport aux systèmes disposant d'un seul processeur. Sachant que les processeurs réseaux peuvent incorporer plusieurs processeurs capables de fonctionner simultanément, il convient alors d'étudier l'impact de ce type de topologie sur notre convertisseur.

4.3.2.1 Présentation des machines parallèles

Une machine parallèle peut être vue comme une agglomération d'unités de traitement semblables à celles que l'on retrouve dans un système classique de traitement

de données. Le principal aspect est la coopération, c'est-à-dire que ces unités de traitement peuvent communiquer entre elles et traiter ensemble un même problème.

Plusieurs types de parallélisme peuvent être adoptés dans la conception d'une application parallèle. Les plus utilisés sont :

- le parallélisme des données ;
- le parallélisme de contrôle.

Le *parallélisme des données* s'applique dans les cas où les mêmes actions sont répétées sur plusieurs données. On alloue alors une tranche de données à chaque processeur, de sorte que la tâche est réalisée de façon simultanée sur plusieurs données à la fois.

Le *parallélisme de contrôle* provient du fait que certaines instructions de l'application peuvent être exécutées simultanément et de manière indépendante par des unités de traitement différents, sans affecter le résultat final. Après avoir évalué les dépendances entre les instructions d'un programme, on peut alors associer à chaque processeur un lot d'instructions. On diminue ainsi le temps d'exécution total de l'application.

Deux modèles de contrôle sont souvent utilisés selon le type d'architecture : SIMD (Single Instruction Multiple Data) et MIMD (Multiple Instruction Multiple Data). Dans le modèle SIMD, on a une unité globale de contrôle qui gèrent tous les processus, tandis que dans le modèle MIMD, on en a plusieurs (souvent un par processeur).

Il convient aussi de préciser que la communication entre les processeurs peut se réaliser selon un *passage de messages* d'un processeur à l'autre ou selon le principe de *mémoire partagée.* Dans le premier cas, le processeur, après avoir traité les données, envoie les résultats aux autres processeurs qui en ont besoin, ou à tous les processeurs (en utilisant un « broadcast » par exemple). Dans le second cas, tous les processeurs lisent et écrivent dans une zone mémoire commune. Il faut alors faire gérer les accès aux données partagées pour éviter les conflits.

4.3.2.2 Impact du nombre de processeurs

Nous avions constaté que notre modèle générique engendrait un délai supplémentaire dans la conversion des paquets. Considérant les gains de temps de traitement que l'on peut tirer du modèle parallèle, nous évaluons dans cette section la performance de notre convertisseur selon le nombre de processeurs utilisés.

Notre implémentation a considéré essentiellement un parallélisme des données et de quelques instructions de contrôle. Ce qui veut dire que les performances obtenues peuvent être éventuellement améliorées avec un degré plus élevé de parallélisme des instructions de contrôle. Nous avons considéré les mêmes applications que nous avions utilisées à la section 4.3.1 pour évaluer l'influence de notre modèle générique. Les mesures ont été effectuées pour des topologies incluant 2, 4 et 8 et parfois 10 processeurs. Les figures 4.8 et 4.9 montrent les résultats obtenus dans les cas de la conversion des paquets Token-Ring à Ethernet puis FDDI à Ethernet respectivement.

Comme on peut le constater, l'influence du parallélisme se fait nettement sentir. En effet, dans tous les cas, avec plusieurs processeurs, le délai de traitement des paquets au niveau du convertisseur générique proposé est plus faible que celui d'un convertisseur simple de protocoles. Les meilleures performances ont été obtenues avec 4 processeurs. Au-delà, on ne gagne plus tellement bien que les délais de conversion soient toujours plus faibles que ceux des convertisseurs spécifiques. Cela veut dire que l'ajout indéfini de processeurs n'implique pas forcément une diminution conséquente du temps de conversion. Il faut préciser qu'au-delà de 10 processeurs, on dégrade considérablement la performance. Les simulations montrent qu'en général, avec plus de 10 processeurs, le délai de conversion dépasse celui d'un convertisseur spécialisé et on ne gagne plus rien. Les mesures laissent croire que, pour chaque quantité de données, il y a un nombre limite de processeurs au-delà duquel on n'améliore plus tellement la performance, même qu'on la dégrade plutôt.

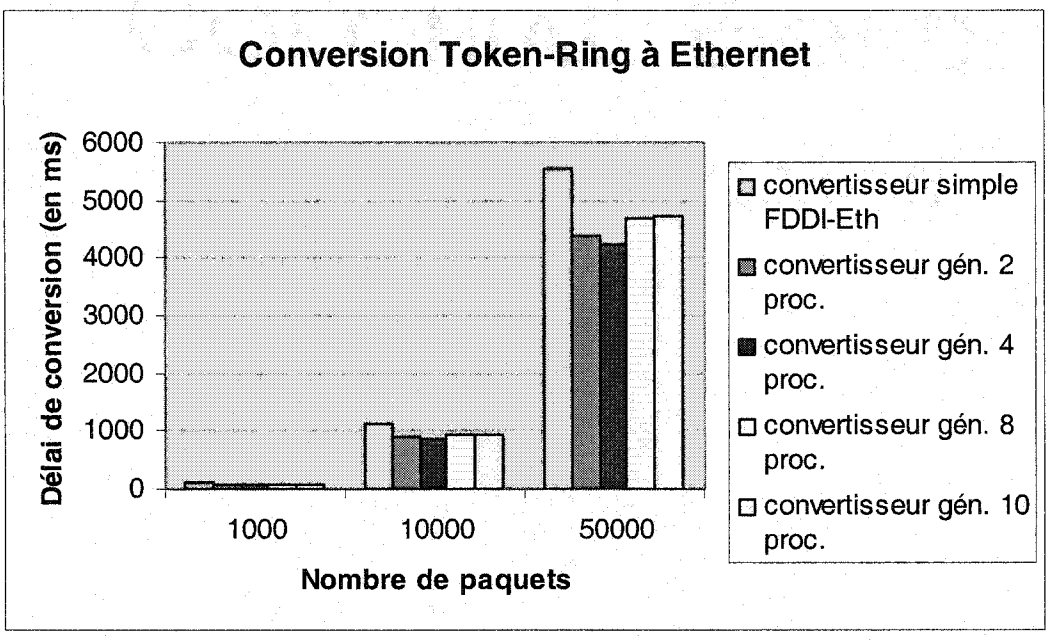


Figure 4.8 Impact du nombre de processeurs sur le délai de conversion Token-Ring à Ethernet

Les figures 4.8 et 4.9 montrent aussi que l'influence de l'architecture à plusieurs processeurs se fait sentir pour de grandes quantités de données. En effet, les variations de délai de conversion pour 1000 paquets entre les différentes topologies sont à peine notables. Mais lorsqu'on considère de plus grandes quantités de paquets (10.000, 50.000 etc.), on voit nettement les avantages de l'approche utilisant plusieurs processeurs.

Les mesures montrent qu'à partir de 8 processeurs, on n'améliore plus la performance. On stagne au même niveau qu'avec 4 processeurs ou alors on rajoute même de petits délais. En réalité, dans les applications parallèles, il y a souvent des limites de performance qu'on peut atteindre dues aux fractions de code qui ne sont pas parallélisables d'une part, et aussi aux mécanismes de communication entre processeurs qui peuvent devenir importants lorsque le nombre de processeurs augmente d'autre part. Le temps de réponse global du système augmente alors à partir d'un certain nombre de processeurs, ce qui en dégrade la performance.

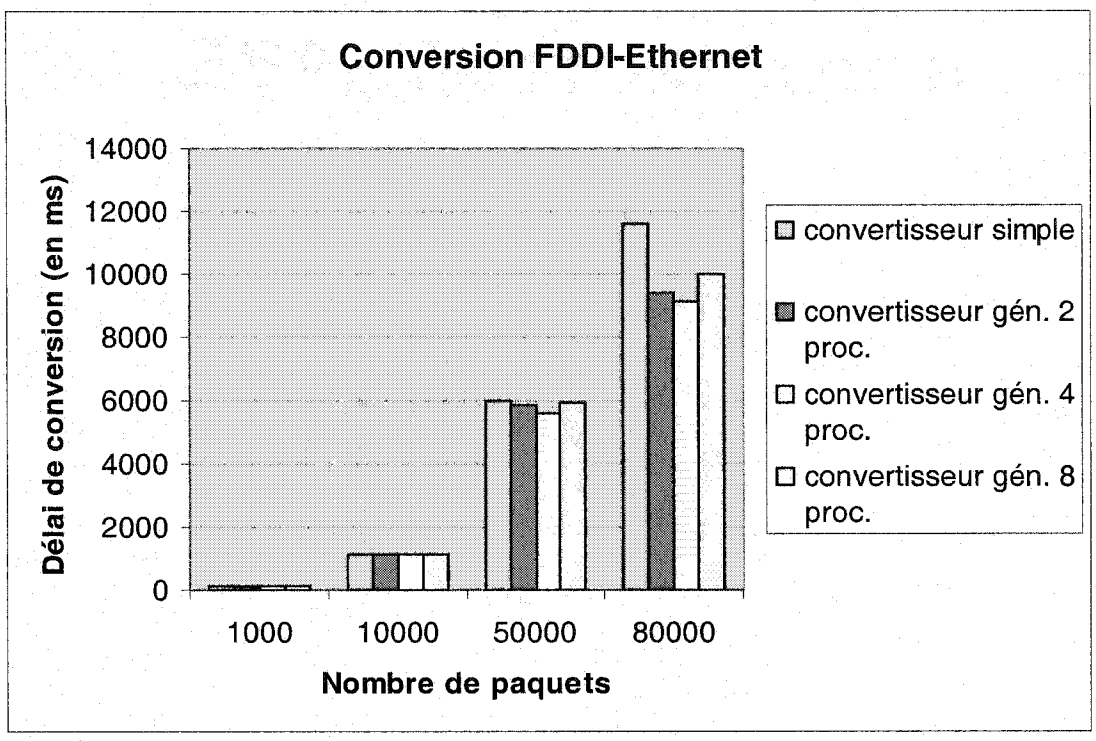


Figure 4.9 Impact du nombre de processeurs sur le délai de conversion FDDI à Ethernet

Théoriquement, en ajoutant un processeur, on devrait s'attendre à ce que la performance du système s'améliore seulement si le nombre de paquets à convertir atteint un certain seuil donné pour que les communications entre processeurs soient négligeables. Dans nos mesures, nous avons considéré 100.000 comme valeur maximale du nombre de paquets. Il est probable que des mesures sur des millions de paquets révèlent par exemple que le nombre idéal de processeurs, lorsqu'on dépasse le seuil du million, soit de 8 ou même plus.

Bref, avec le modèle de convertisseur proposé, on est plus performant qu'un convertisseur spécialisé sur des topologies à plusieurs processeurs. Si *Ts* désigne le temps d'exécution du programme séquentiel, et *Tp* celui du programme parallèle, la différence *Ts – Tp* nous donne le gain obtenu grâce au parallélisme. Le gain de

performance réalisé sur une architecture incluant plusieurs processeurs par rapport à un convertisseur spécifique peut être évalué par l'expression :

$$\mu = \frac{Ts - Tp}{Ts} x100 = (1 - \frac{Tp}{Ts}) x100$$

En l'appliquant à nos résultats, on peut gagner jusqu'à de 21% avec 2 processeurs et 24% avec 4 processeurs par rapport à un convertisseur simple.

En conclusion, notre modèle, en plus d'être flexible, peut être plus performant que les convertisseurs traditionnels dans un environnement de processeur réseau. Mais comme on a pu le constater, il existera souvent un nombre idéal de processeurs permettant d'obtenir les meilleures performances.

4.4 Modèle d'un processeur réseau : la plateforme Intel IXP1200

Dans cette section, nous décrivons brièvement les caractéristiques principales matérielles et logicielles d'un processeur réseau réel, ce qui nous permettra de justifier le modèle choisi dans nos simulations. La plateforme considérée est celle du processeur Intel IXP1200.

L'architecture IXP est conçue comme un ensemble de processeurs parallèles (dénommé *Microengine*) intégrant un processeur ARM. Il est composé de six microprocesseurs indépendants (32-bits RISC) et d'un processeur plus puissant dénommé StrongARM. Chaque microprocesseur dispose d'un ensemble de quatre threads pouvant travailler en parallèle et dispose de la capacité de traitement nécessaire pour réaliser des tâches d'un ASIC. Chaque thread peut réaliser plusieurs opérations asynchrones parallèles simultanément. Chaque microprocesseur a un ALU puissant dont les instructions ont été conçues pour manipuler des tâches relatives aux bits, byte, word (2 bytes) et longwords (4 bytes). Le StrongARM peut être utilisé pour des tâches plus complexes comme la résolution d'adresses, la gestion de réseau etc. Le processeur dispose donc au total de 24 threads capables de travailler en parallèle. De plus, d'après les spécifications, le changement de contexte est quasi-nul entre les threads. Dans les

applications de commutation LAN (LAN switching), les six microprocesseurs sont capables d'acheminer plus de 3 millions de paquets Ethernet à la couche 3.

Une suite d'éléments logiciels a été développée par Intel pour le développement de code, la simulation et le déboguage et peut être utilisée en conjonction avec les systèmes temps réels (RTOS) et les outils logiciels du ARM pour construire des applications embarquées complètes. Il dispose d'un bus de données pour la réception et le transport des données (paquets), d'une interface DMA entre péripheriques et processeurs, d'une SDRAM et SRAM interface et de FIFOs pour stocker les paquets. On a deux types de mémoire : la SRAM, rapide, utile pour les tables de recherche, SDRAM pour les queues de transmission et l'acheminement des données. La Figure 4.10 illustre les principales composantes matérielles précitées de l'architecture du processeur réseau. Il faut préciser que d'après les spécifications, ce processeur a été conçu spécifiquement pour les tâches de classification et d'acheminement de paquets.

4.4.1 Mode d'opération

Les microprocesseurs reçoivent les paquets sur le bus servant d'interface avec le port et vont éventuellement les stocker dans la mémoire RAM. Lors de la réception, ils peuvent parallèlement vérifier les entêtes des paquets ainsi que les CRC, réaliser des recherches dans les tables ainsi que la classification de paquets. Dans les applications, quelques threads peuvent être alloués à la transmission des paquets sur le bus, tandis que d'autres peuvent être alloués à la réception de ces paquets. L'allocation des threads peut être contrôlée en logiciel, ce qui veut dire que le développeur peut choisir d'allouer certains threads à des tâches spécifiques.

L'environnement de développement construit par Intel repose essentiellement sur Windows NT, mais peut être aussi utilisé pour des applications développées sous Linux. Un ensemble de fonctions permettant de contrôler les threads (Microengine C library) a été aussi proposé (fichier *ixp_thread.c*). Un PC est utilisé comme plateforme de développement et peut supporter l'un ou l'autre des systèmes d'exploitation précités. Le port COM de ce dernier est relié par câble sériel au processeur réseau. Nous pouvons

donc réaliser sur la plateforme relié au processeur réseau, toute opération possible sur un PC normal, en particulier l'ajout de librairies.

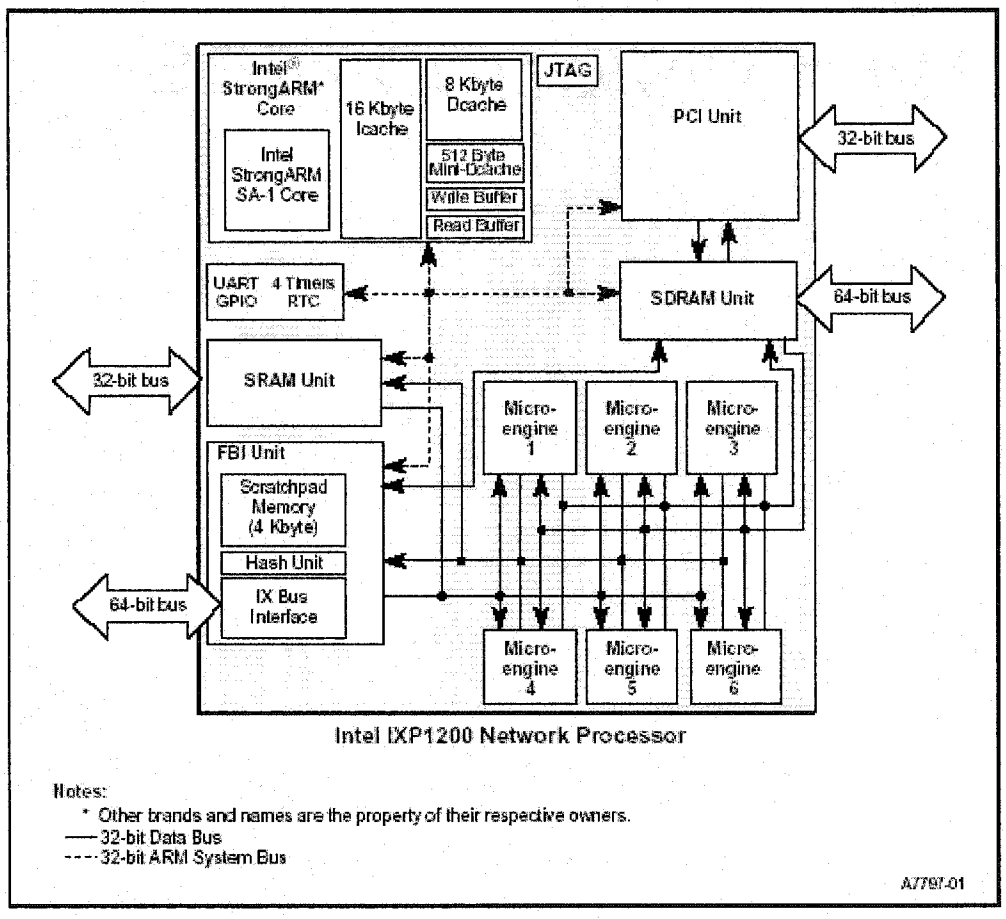


Figure 4.10 Architecture du processeur réseau Intel IXP1200

D'autre part, d'après les spécifications, l'environnement de développement du ARM IXP1200 supporte le langage C/C++, les outils utilisés peuvent être entre autres Wind River Tornado 2.0.1, et le système d'exploitation, Windows NT ou Linux. Au niveau matériel, il faut la plateforme *Intel IXDP1200 Advanced Development*, Vxworks, Lynuxworks ou Cygmon éventuellement et des interfaces Ethernet et sérielle. La Figure 4.11 montre la connexion entre le processeur réseau et la plateforme de développement.

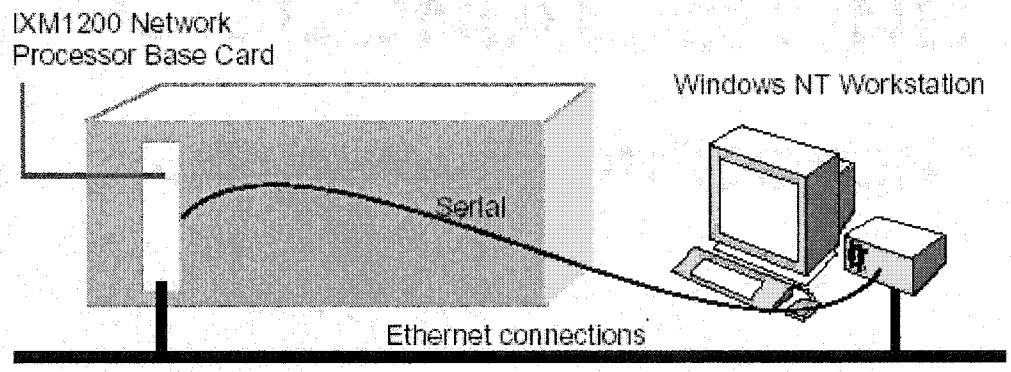


Figure 4.11 Connexion entre le processeur réseau et le PC

4.4.2 Modèle de Thread

Le modèle du processeur réseau présenté peut être donc assimilé à celui d'un système disposant de six processeurs, chacun d'entre eux pouvant déployer quatre threads travaillant en parallèle. Le modèle de programmation pour l'architecture IXP1200 implique donc des programmes s'exécutant sur plusieurs microprocesseurs, chacun comportant plusieurs threads. Le "multithreading" est explicite, c'est à dire que l'usager peut partitionner les tâches entre les threads. Il doit aussi partitionner le programme entre les microprocesseurs et éventuellement même gérer les communications interprocessus. Dans l'application ATM-Ethernet donné dans la documentation du processeur par exemple, on suggère d'allouer un total de 8 threads pour la réception des paquets sur les ports ATM et Ethernet, 6 threads pour la transmission de ces paquets, et les autres pour le calcul du CRC et autres tâches.

4.4.3 Approche de simulation utilisée

L'étude des caractéristiques du processeur réseau Intel IXP1200 nous a permis de mieux cerner les composants matériels et logiciels constituant un processeur réseau. Pour évaluer la performance de notre application de conversion de protocoles sur ce type d'architecture, nous l'avions parallélisé en affectant aux endroits stratégiques plusieurs processeurs, selon le nombre disponible, ce qui correspond bien au modèle réel du

processeur, puisque l'allocation des threads peut être réalisée de manière explicite par l'usager. Pour ce faire, nous avions utilisé la librairie *MPI*. L'architecture physique sur lequel nous avons réalisé nos tests est un serveur principal relié à plusieurs machines (PC). Pour l'instant, la plupart des applications développées concernent la classification et l'acheminement des paquets.

D'après les résultats de notre simulation, une allocation d'un maximum de dix threads à notre module de convertisseur de protocoles est un choix judicieux. Ce dernier peut être inséré dans une application quelconque destinée à être exécutée sur le processeur réseau, les autres threads s'occupant des tâches de réception, transmission, recherches dans les tables et de classification.

4.5 Conversion des protocoles Firewire et Ethernet

Dans cette section, nous présentons d'abord brièvement le protocole Firewire (ou IEEE1394) puis exposons quelques aspects de sa conversion en protocole Ethernet couramment utilisé dans les réseaux LAN. Nous présenterons par la suite les détails de notre implémentation et les résultats de conversion obtenus.

4.5.1 Le protocole Firewire

Firewire a été spécifié pour permettre la communication entre des équipements quelconques, connectés entre eux par le bus IEEE 1394, indépendamment de leur nature (« peer-to-peer network »). Ainsi, si on considère une caméra, un VCR, une imprimante, un PC et un DVD-RAM connectés entre eux, le protocole permet à n'importe quel équipement de communiquer avec n'importe quel autre sans l'aide d'un intermédiaire : la caméra pourrait directement envoyer ses images au VCR et au DVD-RAM sans l'aide du PC (Figure 4.12). De plus, il supporte d'assez grandes vitesses de transfert, de l'ordre de 100 à 400 Mbps. Avec la nouvelle spécification, on pourra atteindre 3200 Mbps. Enfin, il permet d'avoir de très faibles délais et est facile d'utilisation. Tous ces

avantages font de Firewire un protocole intéressant, auquel tous les concepteurs s'intéressent aujourd'hui.

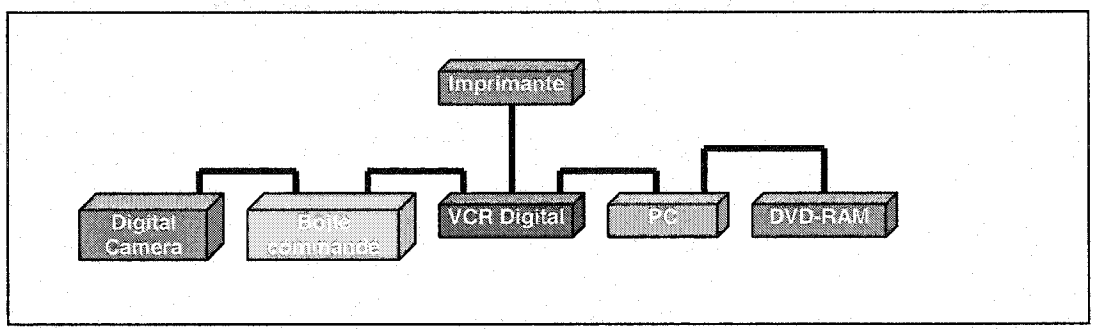


Figure 4.12 Un Bus Firewire

L'architecture du protocole Firewire

La spécification définit trois couches : *Transaction*, *Liaison* et *Physique* dont les actions sont coordonnées par un « bus manager » (Figure 4.13). La couche *Physique* s'occupe des signaux physiques, l'interface physique avec le bus et la reconnaissance des nœuds.

La couche *Liaison* fournit un service datagramme confirmé à la couche *Transaction*. Elle s'occupe de l'adressage, du contrôle d'erreur et de la fragmentation pour la transmission et la réception des paquets. On a deux types de service :

- le transfert asynchrone des données (service avec accusé de réception qui utilise la couche *Transaction*) ;
- le transfert synchrone qui communique directement avec l'application sans passer par la couche *Transaction*.

Quant à la couche *Transaction*, utilisée seulement dans le cadre des communications asynchrones, elle utilise un mécanisme de requêtes-réponses avec accusé de réception. Trois types de transaction sont définies : *Read*, *Write* et *Lock* respectivement pour la lecture, l'écriture, la lecture suivie de l'écriture des données.

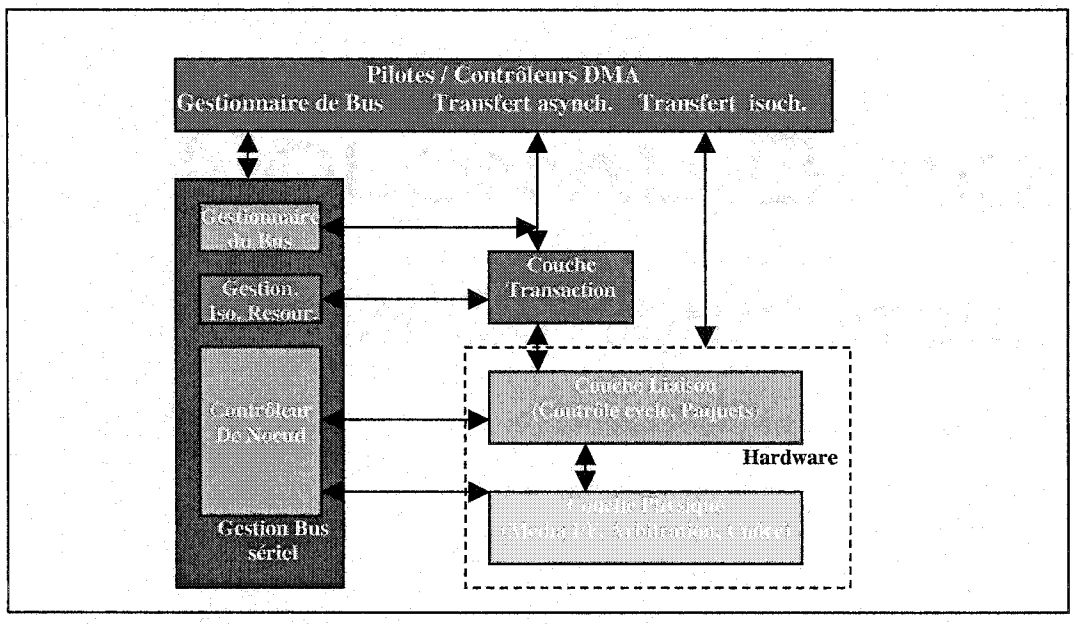
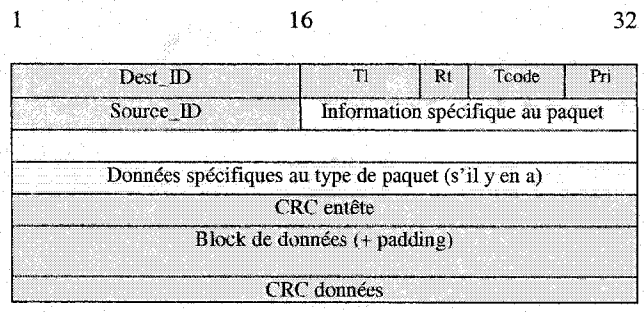


Figure 4.13 Le modèle en couches de Firewire

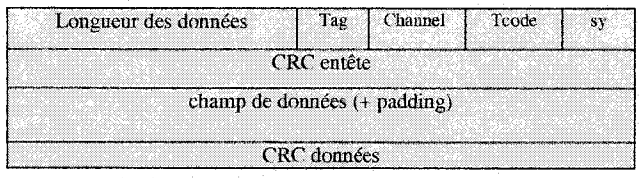
Firewire dispose principalement de trois types de paquet : les paquets isochrones, les paquets asynchrones et les paquets d'accusé de réception (1 byte). Les formats des paquets isochrones et asynchrones sont présentés à la Figure 4.14. Il faut remarquer que le format présenté pour les paquets asynchrones est générique, car on a neuf types de paquets asynchrones et certaines informations dépendent du type du paquet (champ « packet-specific-information »). Les différents types de paquets sont résumés au Tableau 4.6, avec leur signification.

On peut constater que les paquets asynchrones disposent de l'information concernant la source et la destination du paquet, tandis que les paquets isochrones disposent seulement de l'information concernant le numéro de canal à utiliser.

Il faudrait aussi préciser que, dans le cas isochrone, il y a d'abord un cycle d'initialisation (utilisant les paquets de type « cycle start ») au cours duquel le canal est alloué en considérant les adresses des nœuds source et destination. Le format du paquet d'initialisation est celui des paquets asynchrones. Par la suite, tous les paquets isochrones utiliseront ce numéro de canal. C'est un peu l'équivalent d'un circuit virtuel.



a. Format général d'un paquet asynchrone



b. Format général d'un paquet isochrone

c. Signification des champs

Champ	Signification
Tl	Label de transaction
Rt	code de retransmission
Tcode	Transaction code
Pri	Priorité du paquet
Sy	Synchronisation code
Source_ID	Identificateur de source
Dest_ID	Identificateur de destin.

Figure 4.14 Format des paquets asynchrones et isochrones

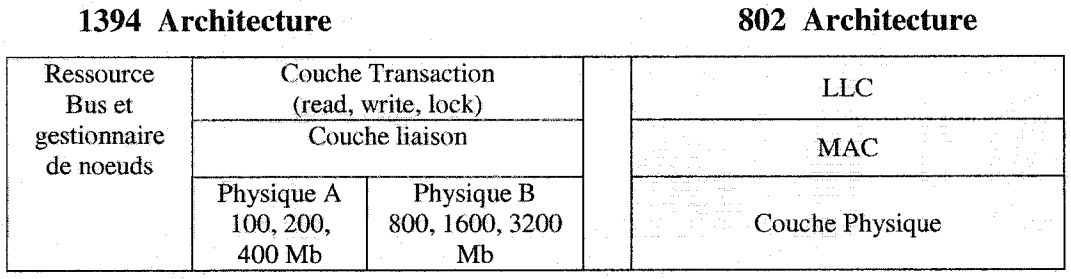
Tableau 4.6 Différents types de paquets asynchrones de Firewire

Nom du paquet	Caractéristiques
Requête d'écriture pour quadlet de données	Requête, payload de type quadlet[1]
Requête d'écriture pour block de données	Requête, payload de type block[2]
Réponse d'écriture	Pas de charge utile
Requête de lecture pour quadlet de données	Requête, pas de charge utile
Requête de lecture pour block de données	Requête, charge utile type quadlet
Réponse de lecture pour quadlet de données	Réponse, données de type quadlet
Réponse de lecture pour block de données	Réponse, données de type block
Cycle Start	Initialisation de cycle isochrone
Requête de type Lock	Requête, block de données

[1] Réfère à un ensemble de quatre octets, terme souvent utilisé avec les paquets de Firewire.
[2] Désigne un ensemble de quadlets de taille variable.

4.5.2 Principales différences entre les protocoles Firewire et Ethernet

Firewire est un protocole assez complexe, et il est difficile de pouvoir le faire correspondre à une couche OSI précise. Il définit une architecture à la manière de X.25 qui englobe les trois premières couches du modèle OSI. Contrairement à Ethernet qui dispose seulement d'un mode de fonctionnement datagramme sans accusé de réception, Firewire dispose de deux modes : l'un qui est datagramme avec accusé de réception (asynchrone) et le second, isochrone sans accusé de réception dont le mécanisme est similaire à celui du service circuit virtuel.



Adresse nœud sur 16 bits (global sur 64 b.)	Adresse de nœud unique 48 bits
Datagrammes confirmés et non confirmés	Datagrammes non confirmés
Canaux isochrones multicast	Multicast asynchrones
Interface avec couche sup. : Transaction	Ponts transparents à couche MAC
Transaction mémoire read/write	Utilisation de trames

Figure 4.15 Principales différences entre les protocoles Firewire et Ethernet

De plus, dans le mode asynchrone qui ressemble à celui d'Ethernet, on a plusieurs types de paquets, certains contenant des données. La longueur maximale de la charge utile peut être de 512, 1024, ou 2048 octets selon la vitesse du lien Firewire utilisé, alors qu'Ethernet peut accepter jusqu'à 1500 octets.

Enfin, l'adressage des nœuds est faite de manière aléatoire sur 16 bits (obtenu à chaque initialisation du bus) dans Firewire, ce qui veut dire qu'une machine peut changer d'adresse physique dans un intervalle de 5 minutes s'il y a eu une

réinitialisation du bus, due à l'ajout d'un équipement au réseau Firewire par exemple. Il faut préciser que chaque nœud dispose d'un offset de 48 bits pour adresser un équipement qui lui est connecté, ce qui donne une marge globale de 64 bits pour l'adressage d'un équipement quelconque. La Figure 4.15 résume les principales différences entre les deux protocoles.

4.5.3 Étude de la convertibilité de Firewire et Ethernet

Dans la suite, nous considérerons le protocole Ethernet tel que spécifié par la norme IEEE 802.3. Nous en avons déjà fait référence à plusieurs reprises dans les chapitres précédents, raison pour laquelle nous n'en ferons plus un rappel. Nous appliquerons tout simplement la méthodologie proposée au chapitre 3 pour déduire si une conversion est possible entre les deux protocoles considérés. Pour cela, il nous faut identifier les services primaires de chaque protocole.

Services primaires de Firewire

À la couche liaison du modèle OSI, les champs d'adressage physique et de contrôle d'erreur sont ceux qui généralement offrent les services primaires (voir définition de services primaires au 4.2.1). On peut aussi remarquer que, dans Firewire, l'adressage physique est utilisé pour communiquer avec un nœud précis, donc c'est un champ critique.

- Mode asynchrone

Dans ce mode, le paquet est acheminé sur la base des informations d'adressage physique contenues dans les champs destination_ID et source_ID. Les champs CRC permettent au destinataire de s'assurer de l'intégrité des données. Ces champs seront donc considérés comme primaires. Nous noterons 1394_ID l'adressage physique de Firewire (chaque nœud obtient une adresse aléatoire). Le service SRC_1394_ID (offert par source_ID) est donc de type PHY_SRC_ADDR et DST_1394_ID de type

PHY_DST_ADDR. C'est alors un nouveau type d'adressage physique. L'information est acheminée sous la forme de datagramme. L'ensemble de services primaires offerts sera alors: PHY_SRC_ADDR, PHY_DST_ADDR, ACK, ERR_CTRL.

- Mode isochrone

Le numéro de canal et les champs de CRC constituent les informations d'adressage et de contrôle d'erreur. Ils sont les champs primaires. L'information est acheminée en utilisant une connexion. L'ensemble des services primaires dans ce mode sera : ADDR _VC, ERR_CTRL.

Services primaires de Ethernet 802.3

L'ensemble de services primaires tel qu'il était identifié est : PHY_SRC_ADDR, PHY_DST_ADDR, ERR_CTRL.

Comme on peut le constater, le mode asynchrone de Firewire semble offrir tous les services primaires du protocole Ethernet et de plus opère selon le mode datagramme. Il faudrait cependant étudier la convertibilité de chaque service avant de tirer des conclusions. À notre connaissance, il n'y a pas de mécanisme qui permette de convertir de manière automatique une adresse MAC Ethernet quelconque de 48 bits en une adresse Firewire quelconque de 16 bits sans implémenter d'autres fonctions. On sait de plus que l'adresse MAC Ethernet est unique, tandis que celui de Firewire est dynamique et ne peut caractériser de façon unique un équipement du réseau Firewire. La conclusion évidente est que les adresses physiques ne sont pas convertibles directement et, étant des services primaires, entravent la convertibilité des protocoles.

On peut donc espérer qu'en résolvant ce problème, l'on pourra supporter la conversion des services primaires d'Ethernet car l'ensemble des services primaires d'Ethernet est un sous-ensemble de celui de Firewire.

4.5.4 Résolution de l'interconnexion Firewire - Ethernet

Comment réaliser la conversion d'adresses de nœuds attribués aléatoirement en adresse MAC Ethernet? Ces deux systèmes d'adressage n'ont rien de commun. Nous nous retrouvons dans un cas semblable à celui de la conversion ATM - Ethernet. On rappelle que la conversion des adresses était aussi le principal problème à surmonter. Pour le résoudre, une option a été de construire une couche IP sur le réseau ATM, pour avoir un adressage commun. Nous trouvons cette approche appropriée au cas présent. C'est en réalité la méthode de la complémentation. Nous allons donc construire au niveau de chaque nœud Firewire une couche IP (IP sur 1394). Il y aura alors un adressage commun IP à tous les systèmes aussi bien sur le réseau 1394 que sur le réseau Ethernet. Le protocole IP permet d'identifier un nœud de façon unique avec son adresse. Il pourra ainsi permettre de résoudre le problème des adresses de nœuds dynamiques de Firewire. Une version adaptée du protocole ARP pourra alors permettre de faire la résolution d'adresses. C'est l'occasion de préciser que la RFC2734 propose une implémentation de IP sur Firewire (IP over 1394) ainsi qu'une adaptation du protocole ARP (nommé ARP1394). La Figure 4.16 schématise le concept de IP sur 1394 et montre la correspondance avec le IP classique sur Ethernet.

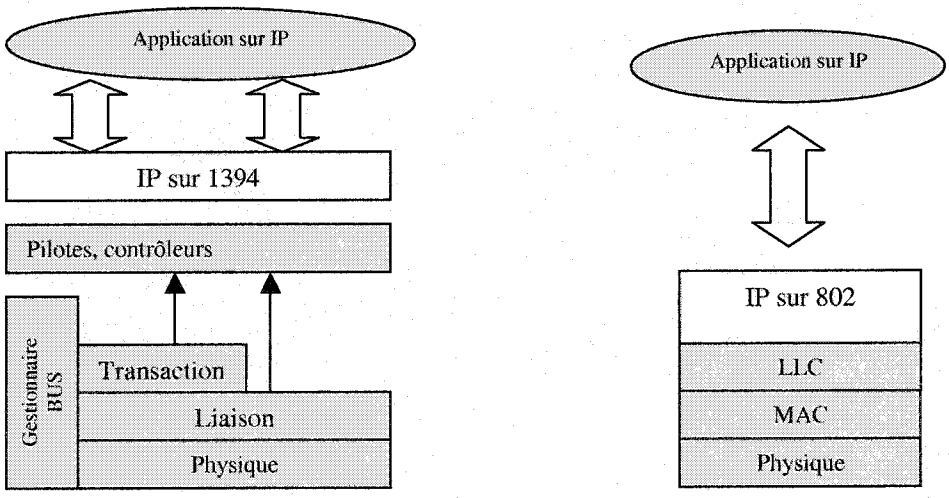


Figure 4.16 IP sur Firewire et IP sur Ethernet

Au-delà de l'adressage, nous avons le problème des accusés de réception. Le protocole Firewire attend des accusés de réception à chaque envoi de paquet asynchrone. S'il n'en reçoit pas, il considère que le paquet n'a pas été délivré correctement et ce dernier est retransmis. Il faut alors implémenter une couche virtuelle au niveau du convertisseur pour gérer ces paquets de commande et d'accusé de réception.

Dans le cas où nous avons des équipements comme des caméras qui fonctionnent en mode isochrone, le problème de la conversion est légèrement différent. Supposons qu'une caméra du réseau Firewire doive envoyer des données numériques à un équipement se situant dans un réseau Ethernet. Nous savons que la transaction entre une caméra et un PC sur Firewire est similaire à un circuit virtuel, et nous nous retrouvons dans une situation similaire à celle de l'interconnexion IPv4 et X25. Il n'y a plus d'accusé de réception à gérer. Il faut alors distinguer deux cas dans la conversion de Firewire à Ethernet, selon les types de machines utilisés :

- 1er cas : conversion de paquets asynchrones en paquet Ethernet. C'est le cas d'un PC de Firewire devant communiquer avec un PC du réseau Ethernet. Comme nous venons de le mentionner, la conversion est réalisable si nous implémentons la couche IP.
- 2e cas : transport de paquets isochrones vers le réseau Ethernet. C'est la situation correspondant à une caméra devant envoyer ses données à un PC du réseau Ethernet par exemple. Dans ce cas, étant donné que les paquets isochrones ne contiennent pas d'information d'adressage du nœud destinataire, nous utiliserons une approche similaire à celle considérée dans l'interconnexion des réseaux IP par un réseau X25. On encapsule les paquets IP dans des paquets X25 quand on passe de IP à X25 et on décapsule à la destination. Dans notre cas, les paquets isochrones seront alors encapsulés dans des trames Ethernet au niveau du convertisseur puis décapsulés à destination. Cette approche a été testée en pratique et fonctionne correctement. Les données reçues de la caméra sont encapsulées directement dans des paquets UDP ou TCP et transmis vers un PC situés sur le réseau Ethernet.

Notre travail dans la conversion de Firewire à Ethernet se concentre sur le premier cas. On considère que seuls les nœuds du réseau 1394 utilisant des paquets asynchrones et capables de supporter une implémentation de IP (typiquement des PC) peuvent communiquer avec un nœud quelconque du réseau Ethernet.

Bref, la complémentation est l'approche qui va nous permettre de faire communiquer des machines sur un réseau Firewire avec d'autres résidant sur un réseau Ethernet en mode asynchrone. Comme on l'a mentionné, une couche virtuelle IP devra être créée au niveau de chaque nœud Firewire pour gérer le problème d'adressage. Cela n'est possible en général que si nous avons des machines nous permettant de créer facilement cette couche comme des PC. En mode isochrone, c'est plutôt l'encapsulation qui est recommandée puisque les paquets isochrone et Ethernet ne transportent pas les mêmes types d'informations et donc ne fournissent pas les mêmes services.

4.5.5 Aperçu de notre implémentation

Nous donnons dans cette section quelques aspects de l'implémentation réalisée pour la conversion Firewire à Ethernet, et ceci dans le cas des paquets asynchrones.

Nous avons choisi d'utiliser LINUX comme système d'exploitation étant donné que le code du noyau (kernel) est accessible et que les librairies existantes peuvent être réutilisées. En effet, nous avons besoin de communiquer avec les pilotes des cartes Firewire et Ethernet. Ceci nous oblige à créer notre convertisseur en tant qu'un module qui sera inséré dans le noyau. Cette raison renforce notre choix de LINUX comme système d'exploitation.

La première étape consiste à implémenter la couche virtuelle IP permettant d'adresser chaque machine du réseau Firewire aussi bien au niveau du convertisseur que dans chaque nœud du réseau Firewire, puis d'attribuer des adresses IP à chaque machine de ce dernier réseau avec la commande *ipconfig* disponible dans LINUX. Un code en développement nommé «ip1394 » et disponible en ligne[3] réalise cette couche virtuelle

---

[3] Code disponible sur http://www.ip1394.org. Développé par Guido Fiala.

de IP sur Firewire.

La seconde étape est la construction du convertisseur. Une librairie *Ieee1394_transactions*[4] disponible dans les nouvelles versions de LINUX (noyau 2.4.18) gère les opérations de la couche *Transaction*, c'est-à-dire la formation des paquets asynchrones et l'envoi des paquets de type *quadlet*. Les accusés de réception à la couche *Liaison* sont gérés par les pilotes de nos cartes Firewire. Nous avons alors juste à nous préoccuper de la conversion des adresses et de la bonne formation des entêtes lorsque l'on passe d'un réseau à l'autre. Parmi les structures utilisées, les plus importantes sont *netdevice*[5] qui définit les attributs et méthodes relatifs à une carte réseau, *skbuff*[6] qui contient les caractéristiques d'un paquet à transmettre et dispose d'un champ permettant de choisir la carte sur laquelle le paquet doit être transmis. Un ensemble de fonctions utiles pour communiquer avec les cartes disponibles sur la machine (Ethernet et Firewire) et définies dans le fichier *dev.c*[7] comme *dev_get_by_name( )* qui retourne un pointeur sur une carte dont le nom est spécifié en paramètre, *dev_queue_xmit( )* qui permet de transmettre un paquet sur une carte, *net_dev_init()* pour l'initialisation d'une carte et enfin *netif_rx()* qui passe aux hautes couches le paquet reçu de la carte, nous permettent de sélectionner l'interface physique sur laquelle nous voulons transmettre le paquet.

---

[4] Disponible dans le répertoire /linux/drivers/ieee1394
[5] Structure utilisée dans Linux, définie dans le fichier /linux/include/netdevice.h
[6] Définie dans /linux/include/skbuff.h
[7] Ce fichier est dans le répertoire /linux/net/core/dev.c

```
DÉBUT
    Pour chaque paquet reçu
        Si un paquet asynchrone est reçu sur la carte 1394, Alors
                - Considérer que le paquet est destiné à un nœud Ethernet.
                - Enlever l'entête 1394 asynchrone, et récupérer le paquet IP
                - Lire l'adresse IP du nœud destinataire.
                - Chercher l'adresse IP dans la cache ARP
                  Si adresse IP trouvé, Alors
                      - Lire l'adresse MAC Ethernet de destination.
                      - MAC destination = adresse lu dans ARP.
                  Sinon
                      - lancer une requête ARP vers le réseau Ethernet et obtenir
                      l'adresse MAC de destination.
                  Fin Si.
                - Former une entête Ethernet : MAC source = adresse MAC du
                  convertisseur
                - Encapsuler le paquet IP dans cette entête.
                - Attribuer le pointeur vers la carte réseau Ethernet à ce
                nouveau paquet formé.
                - Transmettre le paquet.
        Fin Si
    Fin Pour
FIN
```

Figure 4.17 Algorithme de conversion Firewire-Ethernet

L'algorithme utilisé pour implémenter notre convertisseur est présenté à la Figure 4.17. Nous n'avons pas eu le temps de réaliser une implémentation complète et le simuler sur un réseau réel incluant plusieurs machines Firewire prouvant ainsi l'exactitude de la conversion des adresses.

CHAPITRE V
CONCLUSION

5.1 Synthèse des travaux

Dans ce mémoire, nous avons étudié quelques aspects de la convertibilité des protocoles de communication d'une manière générale. Après un survol de quelques protocoles d'usage courant, nous avons identifié un ensemble de propriétés qui les caractérisent et montré comment certaines peuvent entraver leur convertibilité. Des situations concrètes ont été utilisées pour illustrer notre raisonnement. Nous avons également proposé une technique permettant de prédire si la conversion est réalisable entre les protocoles. Nous avons considéré la conversion au sens large, c'est-à-dire incluant la complémentation, donc ne se limitant pas à la conversion directe.

Nous avons, par la suite, proposé le modèle d'un convertisseur générique de protocoles, capable de supporter la conversion des protocoles aussi bien orientés bit que caractère et couramment utilisés dans les réseaux de données. Le modèle a été spécifié sous forme d'un ensemble de modules et conçu de manière à permettre des mises à jour, des ajouts de protocoles, donc est flexible.

L'implémentation du schéma proposé nous a permis d'en faire une évaluation de performance. Nous avons pu constater que le gain en flexibilité obtenu nous coûte en performance. En effet, notre convertisseur générique engendre des délais supplémentaires de traitement des paquets par rapport à un convertisseur simple à fonctionnalité fixe, ce qui peut nuire dans un contexte de réseaux à haut débit où certains paquets en file transportent des données en temps réel. L'augmentation du délai de conversion peut aussi engendrer la perte des paquets lorsque les tampons de réception sont pleins.

Considérant le fait que notre schéma de convertisseur est conçu pour les processeurs réseaux, et que ces derniers peuvent disposer de plusieurs processeurs, capables de travailler en parallèle, nous avons effectué des mesures sur des topologies incluant plusieurs processeurs. Nous avons pu constater qu'on pouvait alors gagner en

performance et même que notre modèle devenait plus performant qu'un convertisseur spécifique. Cependant, le gain de performance dépend du nombre de processeurs utilisés, car, comme on l'a remarqué, la meilleure performance s'obtient avec un nombre précis de processeurs, au-delà duquel on peut obtenir une dégradation de la performance, même si on est toujours plus performant qu'un convertisseur à fonctionnalité fixe.

L'application de notre algorithme de convertibilité sur un ensemble de protocoles a permis de constater son efficacité pour les protocoles dont l'ensemble de services est facilement déductible du paquet (protocole orienté bit). Il faut cependant noter que la détermination des services primaires et secondaires des protocoles, qui est une étape cruciale, peut être très laborieuse dans certains cas. Si des protocoles sont prouvés non convertibles par notre algorithme, alors il n'existe pas de méthode permettant de réaliser leur conversion. Dans de rares cas, éventuellement, une approche utilisant l'encapsulation pourra être utilisée.

La dernière partie de notre travail a reposé sur la conversion des protocoles Firewire et Ethernet. Firewire est une nouvelle spécification capable de supporter de hauts débits comparables à ceux du « Gigabit Ethernet ». Après une brève étude de ces protocoles, nous avons souligné les principales différences existant entre eux et montré que la conversion directe n'est pas réalisable dans un cas général, principalement à cause du problème de la conversion des adresses. Le cas général, suppose en effet que l'on dispose d'un réseau Firewire, d'un réseau Ethernet, et qu'une machine quelconque du réseau Firewire veuille communiquer avec une autre quelconque du réseau Ethernet. Cependant, l'approche de la complémentation peut être utilisée pour résoudre le problème dans le cas des paquets asynchrones, dont les informations de contrôle incluent les adresses Firewire des nœuds source et destination. Nous avons suggéré l'implémentation d'une couche IP au plus haut niveau du modèle en couche de Firewire et donné un aperçu de l'implémentation pour le convertisseur Firewire-Ethernet.

Pour les paquets isochrones de Firewire, la méthode de l'encapsulation dans des trames Ethernet semble être appropriée, puisque ces paquets ne contiennent pas l'adresse du nœud destinataire et sont utilisables directement par l'application.

5.2 Limitations des travaux

Le modèle de convertisseur proposé doit être considéré comme un schéma de base qui peut être utilisé pour implémenter plusieurs conversions de protocoles, et non un convertisseur générique fini disposant de toutes les fonctionnalités pour convertir des protocoles quelconques. On n'aurait pas eu le temps d'implémenter toutes les fonctions nécessaires à la conversion de tous les services couramment utilisés. Il faut aussi préciser que dans notre implémentation, nous avons fait des tests avec des protocoles orientés bits uniquement, d'une part parce que cela était plus simple à réaliser, mais aussi parce que l'implémentation des protocoles orientés caractères induit celle de plusieurs autres services comme l'établissement de connexion, les messages de contrôle et la gestion des accusés de réception, qui, parfois, sont cruciales à leur fonctionnement.

La méthode suggérée pour identifier la convertibilité des protocoles demande une intervention humaine considérable et ce, quasiment à toutes les étapes du processus. Pour réaliser son implémentation, nous sommes obligés pour l'instant d'identifier par nous-mêmes les services primaires et secondaires de chaque protocole. En effet, une automatisation complète du processus requiert d'abord une approche mécanique d'identification de l'ensemble des services des protocoles, puis une méthode de classification automatique de ces services et enfin, une technique de détermination automatique des services convertibles.

Concernant l'identification des services, nous savons qu'elle est une étape cruciale qui peut influencer considérablement le résultat final et pour l'automatiser, il nous faut trouver pour chaque protocole une spécification formelle de son ensemble de services. L'élément dont nous disposons et que nous pouvons utiliser pour la détermination des services pour la plupart des protocoles est le format de leur(s) paquet(s). Les éléments de contrôle du protocole aident généralement à obtenir l'ensemble des services du protocole (cas de IPv4 par exemple), mais cela n'est pas toujours le cas. Pour les protocoles à formats multiples, il est parfois difficile d'obtenir l'ensemble des services du protocole à partir des formats de paquets uniquement (X25 par exemple), ce qui engendre des difficultés.

A propos de la classification entre service primaire et service secondaire, bien qu'une approche ait été proposée, il n'est pas aisé de l'automatiser du fait qu'un service qui est crucial pour un protocole peut éventuellement être optionnel pour un autre. Cette classification requiert donc un effort humain.

Enfin, considérant l'étape de l'identification des services convertibles, il se pose encore un problème pour l'automatisation : comment identifier qu'un service quelconque est convertible en un autre si cela n'est pas connu d'avance? En d'autres termes, considérant deux éléments de service quelconques, comment identifier si la conversion est réalisable entre eux de manière mécanique? Nous n'avons pas pu trouver une approche de résolution automatique. A défaut d'un processus mécanique, une méthode peut consister en une étude générale sur l'ensemble connu des services de protocole afin de dégager des paires de services convertibles entre eux, puis les inclure dans une base de données par exemple. La discussion qui vient d'être faite montre ainsi la nécessité de l'intervention humaine dans notre approche d'identification de la convertibilité.

5.3 Indications de recherches futures

De futures recherches pourront se concentrer sur l'automatisation de ce processus d'identification et de classification de services, en vue de réaliser une application permettant de déterminer mécaniquement la convertibilité des protocoles sans l'intervention humaine.

Concernant le convertisseur, des travaux futurs pourront se concentrer sur le cas des protocoles orientés caractère, et étudier principalement comment le schéma proposé pourrait supporter de manière efficace les services aux hautes couches comme les mécanismes d'établissement ou de terminaison de connexion, le transfert de données d'une application à l'autre. En effet, considérons par exemple le cas d'une conversion entre deux protocoles de transfert de fichiers. Au-delà de la simple correspondance entre les messages de contrôle, de transfert, il faut en plus que le convertisseur réalise une certaine synchronisation entre les commandes envoyées par l'un ou l'autre des

protocoles, et dispose d'un ensemble de fonctionnalités lui permettant de supporter l'ouverture et la fermeture de sessions entre applications par exemple.

Quant à l'implémentation du convertisseur de Firewire à Ethernet, des tests sur un réseau Firewire réel incluant plusieurs machines, restent à être réalisés pour confirmer l'efficacité de l'approche utilisée pour la conversion des adresses principalement.

BIBLIOGRAPHIE

Auerbach J., "A protocol conversion software toolkit", *ACM SIGCOMM Computer Communication Review*, Symposium Proceedings on Communications Architectures and Protocols, August 1989, Vol. 19, No. 4, pp. 259-270.

Baboescu F., G. Varghese, "Scalable Packet Classification", *Proceedings of ACM SIGCOMM'01*, August 2001, San Diego, California, USA, pp. 199-210.

Bailey M., Gopal B., Pagels M., Peterson L. and Sarkar P., "PathFinder : A pattern-based Packet Classifier", *Proceedings of the First Symposium on Operating Systems Design and Implementation*, Nov. 1994, pp. 115-123.

Bochmann, G.V.,"Deriving protocol converters for communications gateways", *IEEE Transactions on Communications*, Vol. 38, No. 9, Sept 1990, pp. 1298-1300.

Boggs D.R., Shoch J.F., Taft E.A., Metcalfe R.M., "Pup: An Internetwork Architecture", *IEEE Transactions on Communications*, Vol. 28, No. 4, Apr. 1980, pp. 612-624.

Calvert K. L., Lam S., "Deriving a protocol converter: a top-down method", *ACM SIGCOMM Computer Communication Review, Symposium Proceedings on Communications Architectures and Protocols*, Vol. 19, No. 4, Aug. 1989, pp. 247-258.

Calvert K.L., Lam S.S., "Formal methods for protocol conversion", *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 1, Jan. 1990, pp. 127-142.

Chang J., Liu M.T., "An approach to protocol complementation for internetworking" Systems Integration '90, *Proceedings of the First International Conference on Systems Integration*, 23-26 Apr. 1990, pp. 205-211.

Cerf V.G., Kirstein P.T., "Issues in Packet-Network Interconnection", *Proceedings of the IEEE*, Vol. 66, No. 11, Nov. 1978, pp. 1386-1408.

Dhar, P., Ganguly, R.C., Das, S., Saha, D, "Network interconnection and protocol conversion - a protocol complementation approach", TENCON '92, *IEEE Region 10 International Conference*, 1992, Vol. 1, pp. 116-120.

Feldmann A., Muthukrishnan S., "Tradeoffs for Packet Classification," *Proceedings of IEEE Infocom 2000, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, Mar. 2000, Vol. 3, pp. 1193-1202.

Fluckiger F., Gateways and converter in computer networks, *Computer Networks and ISDN Systems*, Vol. 16, No. 1-2, Sept. 1988, pp. 55-59.

François P., Potocki A., "Some Methods for providing OSI Transport in SNA", *IBM Journal of Research and Development*, Sept. 1983, Vol. 27, No. 5, pp. 452-463.

Gien M., Zimmermann H., "Design Principles for Network Interconnection", *IEEE/ACM Proceedings of the Sixth Data Communications Symposium*, November 1979, pp. 109-119.

Green P.E. Jr, "Protocol Conversion", *IEEE Transactions on Communications*, Vol. 34, March 1986, No. 3, pp. 257-268.

Gupta, P., McKeown, N., "Algorithms for packet classification", *IEEE Network*, Vol.15, Apr 2001, No. 2, pp. 24-32.

Kawa C., Bochmann G. v., "Hierarchical Multi-network Interconnection Using Public Data networks", *IEEE Proceedings of INFOCOM*, 1987, pp. 426-435.

Kristol D.M., Lee D, Netravali A.N. et Sabnani K., "A polynomial algorithm for gateway generation from formal specifications", *IEEE/ACM Transactions on Networking* (TON), Apr. 1993, Vol. 1, No. 2, pp. 217-229.

Lam S.S., "Protocol conversion", *IEEE Transactions on Software Engineering*, Vol. 14, No. 3, March 1988, pp. 353-362.

Liu M.T., Yao Y.-W, "Constructing protocol converters from service specifications", *Proceedings of the 12th International Conference on Distributed Computing Systems*, Jun. 1992, pp. 344-351.

Okumura K., "A formal protocol conversion method", *Proceedings of the ACM SIGCOMM conference on Communications architecture and protocols*, September 1986, pp. 30-37.

Peyravian M., Lea C.-T, "An algorithmic method for protocol conversion", *Proceedings of the 12th International Conference on Distributed Computing Systems*, IEEE 1992, pp. 328-335.

Piscitello D.M., Weissberger A.J., Stein S.A., Chapin A.L., "Internetworking in an OSI environment", *Data Commmunications*, Vol. 15, No. 5, May 1986, pp. 118-136.

Srinivasan V., Varghese G., Suri S., "Packet Classification using Tuple Space Search," *Proceedings of ACM Sigcomm'99*, Vol. 29, No. 4, Oct. 1999, pp. 135-146.

Sunshine C., "Network interconnection and gateways", *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 1, Jan.1990, pp. 4-11.

Sy K.K., Shiobara M.O., Yamaguchi M., Kobayashi Y., Shukuya S., Tomatsu T., "OSI-SNA Interconnections", *IBM Systems Journal*, 1987, Vol. 26, No. 2, pp. 157-173.

Svobodova L., Janson P. et Mumprecht E., "Heterogeneity and OSI", *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 1, Jan. 1990, pp. 67-79.

Shu J.C. and Liu M., "Protocols Conversion between Complex Protocols", *Proceedings IEEE. Ninth Annual International Phoenix Conference on Computers and Communications*, 1990, pp. 584-590.

Tajiwara T., Ohara Y., Ohta K., "A Study on the Protocol Conversion Method", *Review of the Electrical Communication Laboratories*, Nov. 1982, Vol. 30, No. 6, pp. 1066-1075.

Takei K., Okamura K., Araki K., "Design of Gateway system between different signalling protocols of the multimedia session on the Internet", *Proceedings of International Conference on Information Networking*, IEEE 2001, pp. 297-302.

Tao Z.P., Bochmann G.v., Dssouli R., "A top-down method for synthesizing optimized protocol converters", *Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on Computers and Communications*, Mar. 1995, pp. 270-276.

Tao Z.P., Goosens M., "Synthesizing communication protocol converter: a model and method", *Proceedings of the 1992 ACM annual conference on Communications,* April 1992, ACM Press New York, NY, USA, pp. 17-24.

Welfeld F., "Network Processing in Content Inspection Applications", *Proceedings of the international symposium on Systems synthesis 2001*, ACM, Vol. 14, Sept. 2001, pp. 197-201.

Williams J., "Architectures for Network Processing", *Proceedings of Technical Papers IEEE, 2001*, pp. 61-64.

Woo T., "A Modular Approach to Packet Classification: Algorithms and results", *Proceedings of IEEE Infocom 2000, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 3, Mar. 2000, pp. 1213-1222.

Sites Internet

Protocols for WAN, LAN, ATM data communications and telecommunications: http://www.protocols.com/

Spécifications des protocoles d'Internet : *Request For Comments* (RFC) http://www.ietf.org/rfc

Network Processing Forum (NPF): http://www.npforum.org/ .

IEEE 1394 for Linux: http://www.linux1394.org/

Project info IEEE 1394 for linux: http://sourceforge.net/projects/linux1394/

Network Linux computers over the IEEE 1394 bus: http://www.s.netic.de/gfiala/IPover1394.html

Norme:

IEEE Std 1394-1995 IEEE Standard for a High Performance Serial Bus

ANNEXES

Survol de quelques protocoles d'usage courant

Les architectures considérées sont: la suite la plus populaire TCP/IP, la suite ISO (normalisation), les architectures propriétaires Decnet, Appletalk. Nous allons aussi considérer d'autres suites de protocoles comme X25, H.323, et quelques protocoles LAN. Les protocoles étudiés sont : Ethernet, Token Ring, IPv4, IPv6, TCP, UDP, SMTP, HTTP dans TCP/IP ISO-IP, ISO-TP (0-4), ISO-SP, ISO-X400 dans ISO, DRP, NSP, SCP de Decnet, DDP, ASP, ADSP de l'architecture Appletalk et enfin X25, H323 (RTP, RTCP, H.225, H.261), ATM. Nous présentons dans la suite quelques protocoles choisis dans la liste précédente.

Les protocoles de la suite TCP/IP

IPv4

Défini par l'IETF dans le RFC791, Ipv4 est utilisé sur Internet par tous les autres protocoles de TCP/IP excepté ARP et RARP pour l'acheminement des paquets de données appelés *datagrammes* d'un nœud source à une nœud destination identifié par des adresses dont le format est fixe (adressage IP).

Fonctions principales

- *Adressage*. IP assure le routage des paquets d'une source à une destination spécifié par une adresse IP. Unicité de l'adresse IP.
- *Fragmentation* et *réassemblage* pour la transmission sur des réseaux supportant de petites tailles de paquets.

Fonctions secondaires

Quelques services élémentaires sont prévues au moyen de 4 champs de l'entête : Type de service (ToS), Time to live (TTL), Options et Checksum.

Couche du Modèle OSI

Ipv4 opère à la couche réseau (couche 3)

Mécanismes

Service utilisant des datagrammes. Pas de mécanismes de contrôle de flux, pas de mécanisme de fiabilité dans le transport, pas de connexion. Taille limite des paquets : 20-65535 octets en théorie. Protocole orienté bit.

Le champ *ToS* indique le type de service désiré et peut être utilisé par un routeur pour choisir le prochain hop. Le champ *TTL* indique la durée de vie du paquet et est décrémenté à chaque nœud. Le champ *Options* fournit des fonctions de contrôle (sécurité, timestamp, etc.) utiles dans certaines situations mais est très peu utilisé.

Le « *checksum* » permet de vérifier l'intégrité des données. IP étant un protocole de « best effort», les champs *ToS*, *Options* ne sont pas toujours importants pour l'acheminement des datagrammes. Les informations principales du PCI qui permettent d'assurer les fonctions de base sont les champs d'adressage (Source adress, Destination adress), les champs de fragmentation et d'assemblage (Flags, Total_len, Identification, Fragment_offset) et les champs de contrôle (Header_checksum, TTL).

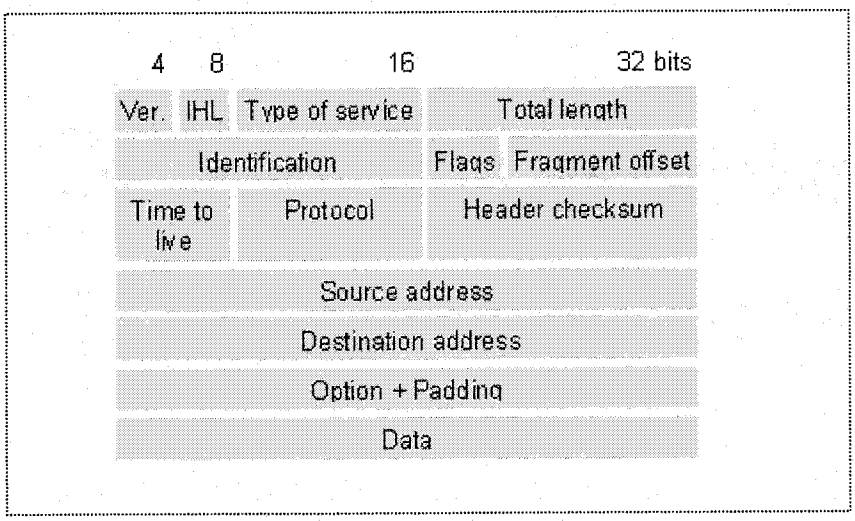


Figure A1 Structure de l'entête IPv4

IPv6

Nouvelle version de IP créée pour combler les insuffisances de Ipv4 en particulier les problèmes d'insuffisance d'adresses IP sur Internet. Les adresses

s'écrivent maintenant sur 128 bits au lieu de 32 dans la version 4, donc on peut adresser un plus grand nombre de nœuds. Elle supporte plus d'options et de champs réservés pour utilisation future. On a une flexibilité sur la longueur du champ options (pour introduire de nouvelles options).

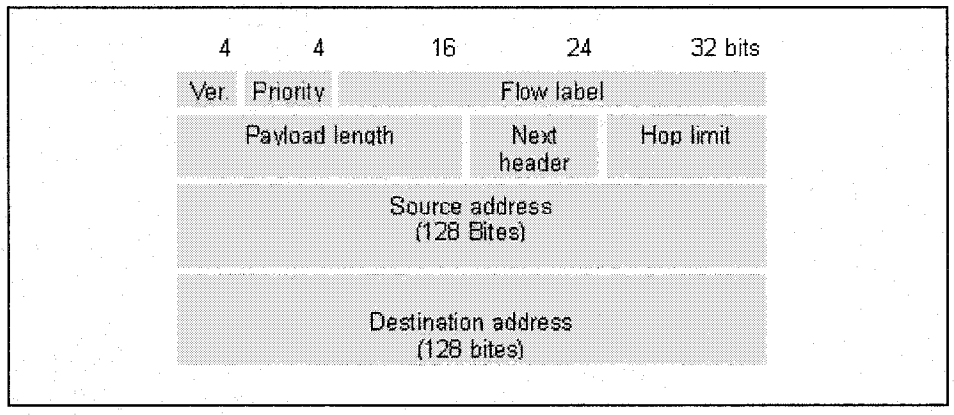


Figure A2 Structure de l'entête IPv6

Fonctions principales :

- Adressage, routage des trames basé sur l'adresse de destination (Source adress, Destination adress) ;
- fragmentation et réassemblage des paquets (Flow label, Next Header, Payload length) ;
- service utilisant des datagrammes.

Fonctions secondaires

- Informations de contrôle en cas de congestion comme priorité, le ToS voulu, le TTL (nommé Hop Limit).
- Options. Le champ *Options* existe et est placé à la fin de l'entête IP.

Mécanismes : mêmes que pour Ipv4

Couche du Modèle OSI : Ipv6 opère à la couche réseau (couche 3)

Il est aisé de remarquer que les services offerts par Ipv4 et Ipv6 sont similaires, et presque identiques à la différence que Ipv6 va permettre d'obtenir une certaine flexibilité par rapport à Ipv4. Les entêtes de ces deux protocoles ont des champs en

commun et des champs spécifiques. Mais on constate que même les champs qui sont différents ont leur équivalent au niveau de l'un ou l'autre des protocoles.

TCP

Le protocole TCP (Transmission Control Protocol), documenté dans la RFC 793, constitue, avec IP, les piliers de la suite TCP/IP. En effet, la plupart des protocoles de cette suite utilisent TCP qui à son tour se sert de IP.

Service principal offert :

- Transport fiable des données de bout en bout. Les adresses source et destination sont des numéros de port identifiant des processus sur les machines communicantes.
- Il utilise un service orienté connexion.

Service secondaire : Contrôle de flux (gère les tampons de données et coordonne le trafic)

Mécanismes :

- Flots de *segments*
- Connexion virtuelle entre applications par les mécanismes de ACK et SN
- *Adressage* grâce aux *numéros de ports* sur 16 bits.
- Usage d'*accusé réception (ACK)* pour confirmer la réception des données.
- *Numéro de séquence (SN)* pour détecter les paquets perdus, retransmission en cas d'erreurs.
- *Fenêtrage* pour contrôle de flux. TCP inclut un champ *Window Size* dans chaque paquet qu'il transmet pour indiquer combien de paquets peuvent être transmis sans ACK.
- *Flags* pour aider à la synchronisation et champ d'options.
- Contrôle d'intégrité des données grâce au *checksum*

Couche du Modèle OSI : TCP opère à la couche transport (couche 4)

TCP est un protocole orienté bit. L'entête TCP est présenté à la figure ci-dessous.

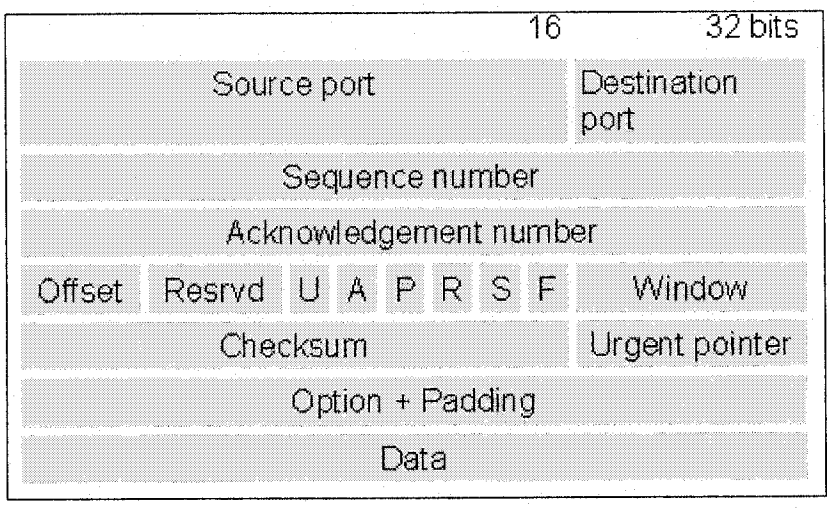


Figure A3 Structure de l'entête de TCP

UDP

Le protocole UDP (User Datagram Protocol) est défini dans RFC768 de l'IETF. Il offre contrairement à TCP un transport simple, non fiable, des datagrammes de bout en bout. Les informations pertinentes dans l'entête sont les numéros de port des applications source et destination.

Service principal offert :

- Transport des datagrammes de bout en bout (entre application source et application destination)

Service secondaire : Néant

Mécanismes :

- *Datagrammes*
- *Adressage* grâce aux numéros de ports comme dans TCP sur 16 bits
- Contrôle d'intégrité des données grâce au *checksum*

- Service *non fiable* (pas de ACK, ni de SN comme dans TCP)
- Pas d'option prévue
- Pas de Flags pour indiquer le début ou la fin des datagrammes, pas de mécanisme de fenêtrage pour le contrôle de flux (WINDOW).

Couche du Modèle OSI : UDP opère comme TCP à la couche transport (couche 4)
Le protocole de la couche réseau va se charger du routage des paquets depuis la machine source vers la machine destinataire. Souvent, il s'agit de IP.

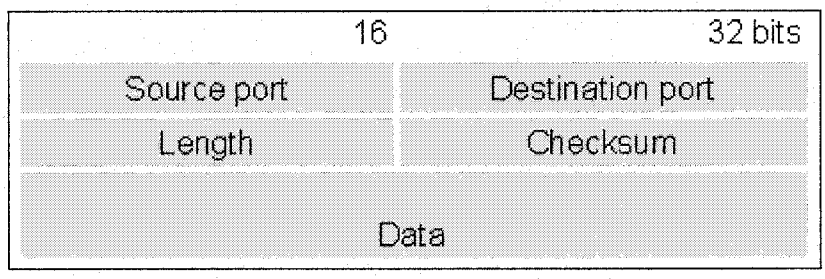


Figure A4 Structure de l'entête UDP

HTTP

Le protocole HTTP (HyperText Transfer Protocol) est le protocole le plus utilisé sur Internet depuis 1990. La version 0.9 était uniquement destinée à transférer des données sur Internet. La version 1.0 du protocole (la plus utilisée) permet désormais de transférer, des messages avec des en-têtes décrivant le contenu du message en utilisant un codage de type MIME.

Service principal offert : HTTP permet un *transfert de fichiers* (essentiellement au format HTML) localisé grâce à une chaîne de caractères appelée URL *entre un navigateur* (le client) et *un serveur Web* (appelé d'ailleurs Httpd).
Mécanismes :

- Protocole orienté caractère
- Le navigateur effectue une requête http

- Le serveur traite la requête puis envoie une réponse http
- Adressage utilisant l'URL.

Couche du modèle OSI : HTTP opère à la couche Application (couche 7)

Format des messages

Le format des messages diffère selon qu'il s'agit d'une requête ou d'une réponse.

Une requête HTTP a la syntaxe suivante	Une réponse HTTP a la syntaxe suivante
COMMANDE URL VERSION<*crlf*> EN-TETE : Valeur<*crlf*> . . EN-TETE : Valeur<*crlf*> *Ligne vide*<*crlf*> CORPS DE LA REQUETE :	VERSION-HTTP CODE EXPLICATION<*crlf*> EN-TETE : Valeur<*crlf*> . . EN-TETE : Valeur<*crlf*> *Ligne vide*<*crlf*> CORPS DE LA REPONSE

(<*crlf*> signifie retour chariot)

Les requêtes

Pour les requêtes, les commandes possibles sont: GET, HEAD (requête d'une ressource), POST (envoi de données), PUT, DELETE.

Les champs possibles de l'entête sont nombreuses. Au nombre des plus courants, on a : *Accept* (décrit le type de contenu accepté par le brownser), *Accept-Encoding* (codage de données accepté par le browser), *Accept-Language* (langage attendu par le browser), *Content-Encoding* (type de codage du corps de la requête), *Content-length* (longueur du corps de la requête), *Content-Type*, *Date*, *Forwarded*, *From*, *Link*, Orig-URL, User-agent. Voici donc un exemple de requête HTTP:

GET http://www.commentcamarche.net HTTP/1.0
Accept: text/html
If-Modified-Since: Saturday, 15-January-2000 14:37:11 GMT
User-Agent : Mozilla/4.0 (compatible; MSIE 5.0; Windows 95)

Les réponses

Une réponse HTTP est un ensemble de lignes envoyées au navigateur par le serveur. Elle comprend:

1- *une ligne de statut* précisant la version du protocole utilisé, le code de statut et un texte explicatif;

2- *les champs d'en-tête de la requête* qui sont un ensemble de lignes facultatives permettant de donner des informations supplémentaires sur la réponse et/ou le serveur. Chacune de ces lignes est composé d'un nom qualifiant le type d'en-tête, suivi de deux points (:) et de la valeur de l'en-tête (Content-Encoding, Content-Length, Content-Language, Content-Type, Date, Location, Server etc.);

3- *le corps de la réponse* qui contient le document demandé.

Voici un exemple de réponse HTTP:

HTTP/1.0 200 OK
Date : Sat, 15 Jan 2000 14:37:12 GMT
Server : Microsoft-IIS/2.0
Content-Type : text/HTML
Content-Lentgh : 1245
Last-Modified: Fri, 14 Jan 2000 08:25:13 GMT

SMTP (Simple Mail Transfer Protocol)

SMTP, spécifié dans RFC 821, est le protocole standard permettant de transférer du courrier électronique d'un serveur à l'autre en utilisant une connexion point-à-point. Il définit la structure des messages et le protocole de communication entre les agents de transfert de messages (MTA). La communication entre les agents s'effectue au niveau application.

Service principal offert *: Transfert du courrier électronique d'un noeud à l'autre* à l'aide d'une connexion point à point.

Caractéristiques du protocole

- Protocole fonctionnant en mode connecté (encapsulé dans une trame TCP/IP).

- Protocole orienté caractère. Le protocole SMTP fonctionne grâce à des commandes textuelles envoyées au serveur SMTP (par défaut sur le port 25).
- Chacune des commandes envoyées par le client (validée par un appui sur la touche entrée) est suivi d'une réponse du serveur SMTP composée d'un numéro et d'un message descriptif.

Format du message

L'entête d'un message SMTP contient les éléments suivants :

- La date et l'heure d'expédition du message (rempli par le MTA de l'expéditeur)
- Le sujet (rempli par l'expéditeur ou l'agent utilisateur de l'expéditeur)
- L'adresse du destinataire (rempli par l'expéditeur)
- La date et l'heure de réception (rempli par le MTA du destinataire)
- Le nom et l'identification de l'expéditeur
- Le nom et la version de l'agent utilisateur expéditeur
- Le type de contenu et jeu de caractères (rempli par l'agent utilisateur de l'expéditeur)
- La longueur du courrier en nombre de caractères (rempli par l'agent utilisateur de l'expéditeur)

Exemple d'en tête de message SMTP

From Yves Thue Apr 12 10 : 30 2000
Subject: End of War
To: dris@yahoo.com
Date: Thue Apr 12 2000 10 : 32: 05
From: "Dwayne Miller"dwayne@polymtl.ca
X-Mailer: ELM version 2.4
Content-type: text/plain; charset= ISO-8859-1
Content-Length:123

Adressage

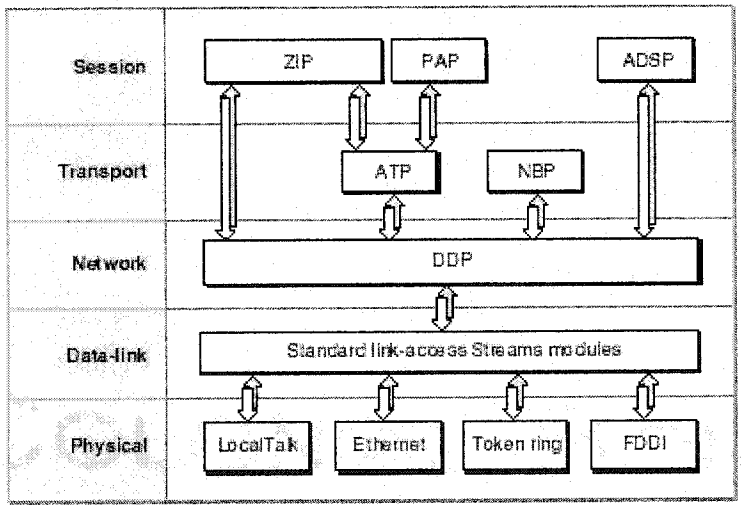
L'adressage utilisé par SMTP se base sur les noms de domaines au lieu de l'adresse IP.

La forme générale est *utilisateur@ss-domaine.domaine*.

Commande	Exemple	Description
HELO (EHLO)	EHLO 193.56.47.125	Identification à l'aide de l'adresse IP ou du nom de domaine de l'ordinateur expéditeur
MAIL FROM:	MAIL FROM: expediteur@domaine.com	Identification de l'adresse de l'expéditeur
RCPT TO:	RCPT TO: destinataire@domaine.com	Identification de l'adresse du destinataire
DATA	DATA message	Corps du mail
QUIT	QUIT	Sortie du serveur SMTP

Les protocoles de Appletalk

AppleTalk est un type de réseau de communication (spécifique à Apple) qui interconnecte différents types de machines (imprimantes, postes de travail, modems, serveurs de fichiers etc..) et permet à ces équipements de s'échanger des informations. Il est implémenté dans les postes de travail MacOS et fournit un adressage dynamique.



DDP (Datagram Delivery Protocol)

Il fournit un service d'acheminement des datagrammes et de routage aux protocoles de la couche supérieure. Dans la version AppleTalk Phase 2, l'entête des datagrammes contient les adresses réseau source et destination des nœuds de même que numéro de socket appropriés.

Fonction de base : acheminement des datagrammes d'une source à une destination.

Couche du modèle OSI: DDP opère à la couche réseau (couche 3).

Contenu des paquets

- Destination socket.
- Source socket.
- Length. Longueur totale du datagramme
- DDP type. Indique le protocole de couche supérieure
- Destination : Network/node/socket sous la forme NNNN.nn (ss),
- Source : Network/node/socket.
- Checksum.
- Hop count. Nombre de routeurs traversés par la trame.

Mécanismes:

Ce protocole opère en mode datagramme, pas d'accusé de réception. Dans l'adressage AppleTalk, un nœud a une adresse dynamique qui peut changer d'un moment à l'autre.

ASP (Appletalk Session Protocol)

Il gère les sessions pour les protocoles de plus haut niveau comme AFP. ASP fournit un identificateur de session unique pour chaque connexion logique et surveille continuellement le statut de cette connexion.

Fonction de base :

- gère les sessions (connexion logique);
- protocole orienté caractère.

Couche du modèle OSI: couche session

Formats des messages

Il existe plusieurs types de messages ASP : open session reqst, close session reqst, command call reqst, status request, session keep alive, session write reqst, write continue req, close session reply, server status reply, session write reply etc.

Les messages peuvent contenir les informations suivantes :

- Session ID (code d'identification de la session);
- Sequence number (utilisé pour maintenir les messages en ordre avec write, command etc);
- Server session socket;
- Workstation session socket;
- Version number;
- Buffer size.

ADSP (AppleTalk Data Stream Protocol)

Il fournit un canal d'acheminement de données entre les machines. C'est un protocole orienté connexion qui garantit l'acheminement des données dans l'ordre avec un contrôle de flux.

Fonction de base : Acheminement des données en ordre avec contrôle de flux.

Couche OSI : Session et Transport.

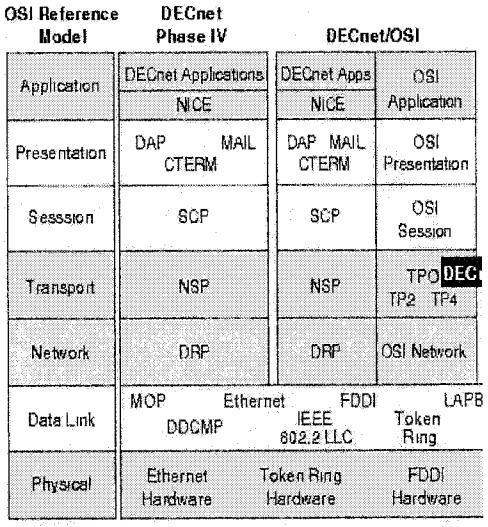
Contenu des paquets :

- Source connection ID
- Destination connection ID
- Send sequence number
- Receive sequence number
- Receive window size
- Version
- Attention sequence number
- Code
- Control flag
- Ack request flag
- End of message flag
- Attention flag

Pour résumer, les informations du PCI concernent l'adressage (ID des connexions source et destination), la fragmentation, et le contrôle des données. ADSP fonctionne en mode connecté comme TCP. Il supporte des sessions sur lesquelles les applications peuvent transporter les données en full-duplex. Il fournit un transport fiable à l'aide d'une connexion point à point. Il permet aussi aux applications de s'échanger des messages de contrôle sans interrompre le flux de données.

Les protocoles de Decnet

DECnet est un protocole propriétaire de Digital Equipment Corporation (DEC) qui a été développé pour permettre la communication entre les machines d'un réseau LAN ou WAN de type DEC. Il inclut les protocoles suivants : **DRP** (Routing Protocol) assure le routage des données d'une station à une autre sur la base des adresses logiques source et destination, **MOP** (Maintenance Operation Protocol) est utilisé pour des services comme le téléchargement de logiciel système, le test à distance de même que le diagnostique des pannes. Au niveau transport, on a **NSP** (Network Service Protocol.), puis **SCP** (Session Control Protocol) à la couche session, **DAP** (Data Access Protocol), **CTERM** - Command Terminal. , **LAT** - Local Area Transport, **STP** - Spanning Tree Protocol, **LAVC** - Local Area VAX Cluster aux plus hautes couches.



DRP (Decnet Routing Protocol)

C'est un protocole propriétaire de DECnet qui fournit l'information nécessaire pour router les paquets d'un nœud source à un nœud destination.

Fonction de base : permet le routage des données.

Caractéristiques

- Il existe plusieurs types de paquets RP mais on peut les classer en trois grandes catégories : paquets « *hello* » (envoyés par les routeurs pour la mise à jour des informations), paquets « *router msg* » donnant de l'information sur le routage dans une zone, paquets « *routed data* » qui contiennent les segments de données usager.
- Protocole orienté bit.
- Routage basé sur l'adresse de destination. Adressage DecNet : des nœuds situés dans des zones différentes peuvent avoir la même adresse.

Couche du modèle OSI: RP opère à la couche Réseau (couche 3)

Format des paquets

Les paramètres des paquets DRP dépendent de leur type. Cependant, ils ont tous le paramètre « *Node Adress*» qui indique l'adresse du nœud auquel est destiné le paquet. Les paquets de type *hello* ont les paramètres suivants : *Routing priority* (priorité sur une échelle de 100), *Hello period* (période de temps entre la mise à jour des informations de routage), *Version, Multicast status, Maximum* (taille maximum des paquets supportés par le lien).

Les paquets de données quant à eux contiennent les informations suivantes :

Node_Adress : adresse du nœud destinataire

Request return : si l'envoyeur désire que le receveur retourne la trame (1 ou 0)

Return path : indique si c'est un paquet de requête ou de réponse (1 ou 0)

Intra-Ethernet : précise si on est directement connecté à Ethernet (1 ou 0)

Version : (valeur 0)

NSP (Network Service Protocol)

C'est un protocole propriétaire de Digital. Il fournit un service orienté connexion, fiable avec contrôle de flux au protocole de la couche réseau chargée du routage des données (DRP).

Fonctions de base :

- Transport des données de bout en bout
- Création et terminaison des connexions entre nœuds, fragmentation et réassemblage de message et gestion de contrôle d'erreur.
- Contrôle de flux.

Mécanismes :

- Adressage grâce aux numéros de port
- 2 types de contrôle de flux : dans le plus simple, le receveur indique à l'envoyeur quand arrêter la transmission alors que dans le plus complexe, le receveur indique le nombre de messages qu'il peut accepter.
- Les paquets peuvent être de différents types : paquets de données, de contrôle du flux de données, accusé de réception ou de commandes (demande de connexion, de déconnexion etc.)
- Protocole orienté bit.

Couche du modèle OSI : NSP opère à la couche Transport (couche 4)

Format des paquets :

Les paramètres du paquet NSP :

Destination link address	(numéro de port destination),
Source link address	(numéro de port source),
Acknowledge number	(numéro d'accusé réception),
Acknowledge other	
Segment number	(numéro du paquet courant),
Flow control	(informations de contrôle de flot),
BOM/EOM	(début/fin de message)

SCP (Session Control Protocol)

Fonctions : gère les liaisons logiques (sessions) pour les connexions DECnet

- Requête d'une session pour un nœud terminal
- Réception de liens logiques (sessions) d'autres terminaux
- Accepte ou refuse des requêtes de session
- Convertit les noms en adresses
- Termine les liaisons logiques

Mécanismes

- Les trames SCP peuvent être de plusieurs types:
  connect data: contient les paramètres de la connexion.
  disconnect : fournit l'information sur le statut de déconnexion.
  reject data : fournit l'information sur le statut de rejet.
- Protocole orienté caractère.

Le contenu des messages est le suivant :

Group (identificateur de code de groupe), *User* (User code identifier), *Descriptor* (A user-defined string of data), *Version* (la version SCP), *Requestor ID* (User name pour l'accès), *Password* (Password pour l'authentification de l'usager), *Account* (lien ou données de service du compte usager), *User data* (données de l'usager).

Adressage des nœuds DEC

La structure générale d'une adresse DECnet est *area.node;* une adresse DECnet contient 16 bits: 6 bits pour la zone (63 max.) et 10 bits pour l'adresse du nœud (1023 max). Cette adresse logique est ensuite formatée et combinée à l'adresse physique MAC de l'équipement. L'adresse MAC est donc modifiée à chaque initialisation selon le réseau logique dans lequel le nœud se situe. Par ex., si un nœud se situe dans une zone dont l'adresse logique du nœud est 5.14 soit 0x1411. Si l'adresse physique était AA-00-06-01, alors l'adresse MAC deviendrait alors AA-00-06-01-11-14. On a donc plus besoin de garder une association entre adresse logique et physique comme dans IP.

Les protocoles ISO

La suite de protocoles ISO a été définie par l'IEEE en conformité avec les sept couches du modèle OSI. Elle comprend les protocoles suivants : IS-IS, ES-IS, ISO-IP – Internet-working Protocol, ISO-TP (Transport Protocol), ISO-SP (Session Protocol), PP (Presentation Protocol), CCITT X.400 (Consultative Committee Protocol). Dans la suite, nous mentionnerons les principales caractéristiques de ISO-IP, ISO-TP, ISO-SP, X400 respectivement.

ISO-IP

Il a été défini par l'ISO pour faciliter l'interconnexion des systèmes. Il fournit un service datagramme (non orienté connexion) et des mécanismes d'adressage, de fragmentation et de signalisation d'erreur pour aider dans le routage.

Fonctions de base :

- adressage des datagrammes, routage basé sur ces adresses;
- fragmentation et réassemblage des paquets.

Mécanismes :

- Service datagramme (CLNP).
- Adressage OSI X.121.
- Protocole orienté bit.

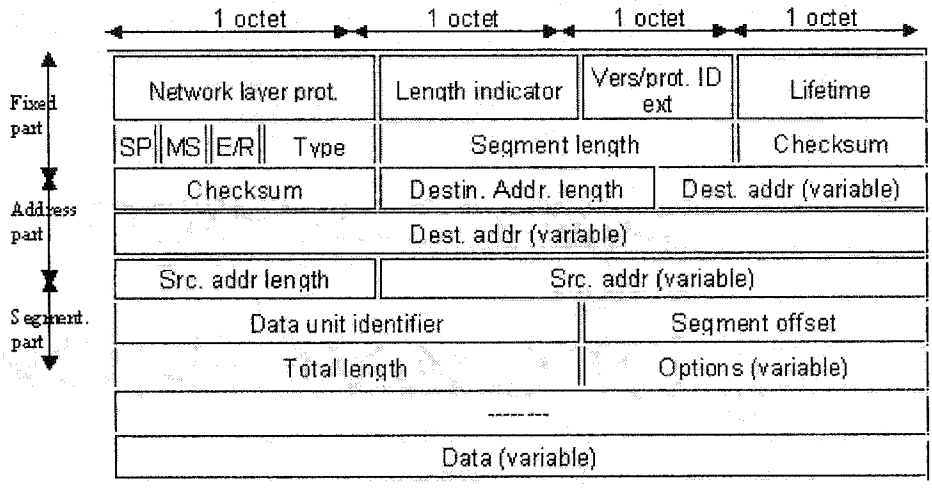
Couche du modèle OSI: ISO-IP opère à la couche réseau comme IP (couche 3).

L'information contenue dans les paquets peut être divisée en quatre grandes parties :

- une partie fixe contenant les informations comme la longueur de l'entête, la version, la durée de vie du paquet, le type de PDU, la longueur du paquet et le checksum.
- une partie d'adressage contenant les adresses source et destination.
- une partie de fragmentation indiquant l'offset du fragment courant, la longueur totale du datagramme initial.

- enfin un champ d'options qui indiquent le padding, la priorité, des paramètres de sécurité, la QoS, le routage de source (source routing), l'enregistrement des nœuds traversés (record route).

La structure de l'entête des paquets de ce protocole est illustré ci-dessous.



ISO-IP PDU

Les protocoles de la suite H.323

RTP (Real Time Protocol)

RTP fournit un transport de bout en bout approprié aux applications transportant des données temps-réel comme les données audio, vidéo, ou de simulation sur des services réseaux unicast ou multicast. Ce protocole ne fait pas de réservation de ressources et ne garantit pas la QoS pour les applications temps-réel. Il est indépendant des protocoles des couches transport et réseau.

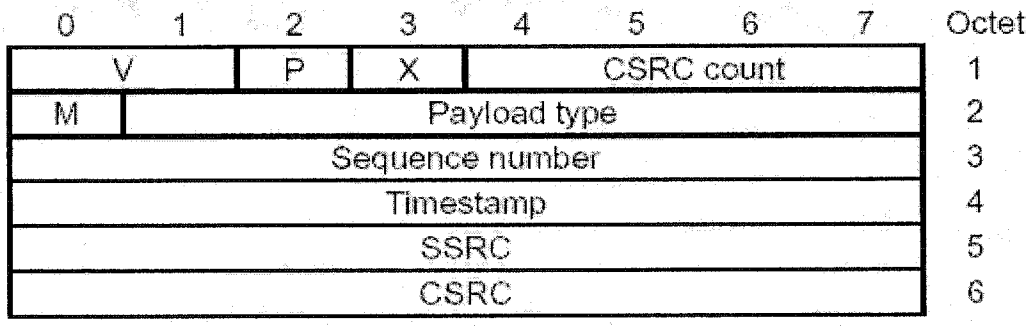
Fonctions de base : Transport de bout en bout des données audio / vidéo sur des réseaux unicast ou multicast.

Caractéristiques :

- Les informations essentielles contenues dans l'entête sont : le numéro de séquence, le temps d'envoi (timestamp), les sources de synchronisation SSRC
- Le timestamp permet à la réception la synchronisation et le calcul de gigue
- Protocole orienté bit
- Pas de mécanisme d'adressage
- RTP fonctionne au dessus du protocole de la couche transport qu'elle utilise pour acheminer les données vers l'application concernée.

Couche du modèle OSI : couche transport

Le format d'un paquet RTP est le suivant :



RTP structure

V - Version, P - Padding, X – Extension bit, M - Marker.
Les champs CSRC indiquent une liste d'identificateurs de source participant;
SSRC identifie la source de synchronisation.

RTCP

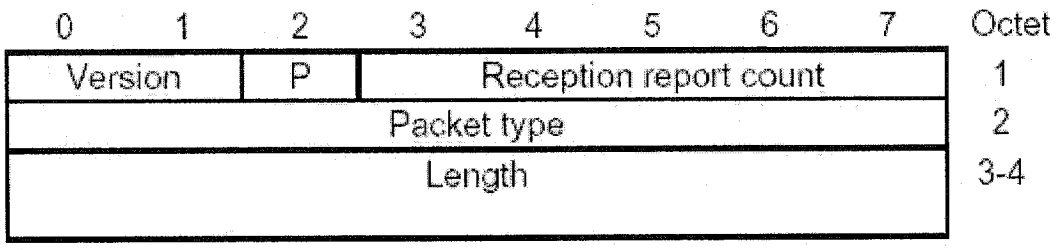
RTCP est un protocole de contrôle dont le but est d'offrir un minimum de contrôle et une fonction d'identification. Il pallie à la faiblesse de RTP en apportant quelques mécanismes de fiabilité. Il est défini dans la RFC1889.

Fonction de base : offre des *mécanismes de contrôle pour le transport des données temps-réel.*

Mécanismes

Des paquets de contrôle sont envoyés périodiquement à tous les particiants de la session en utilisant le même mécanisme que les paquets de données.

Couche du modèle OSI : couche transport



RTCP structure