Titre: Title:	Séparation entre la partie acheminement et la partie contrôle dans les routeurs
Auteur: Author:	Rachid Nait Takourout
Date:	2003
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Nait Takourout, R. (2003). Séparation entre la partie acheminement et la partie contrôle dans les routeurs [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7224/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7224/
Directeurs de recherche: Advisors:	Samuel Pierre
Programme: Program:	Génie électrique

UNIVERSITÉ DE MONTRÉAL

SÉPARATION ENTRE LA PARTIE ACHEMINEMENT ET LA PARTIE CONTRÔLE DANS LES ROUTEURS

RACHID NAIT TAKOUROUT
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES (M.Sc.A.)
(GÉNIE ÉLECTRIQUE)
AOÛT 2003

Your file Votre référence
ISBN: 0-612-89218-2
Our file Notre référence
ISBN: 0-612-89218-2

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

SÉPARATION ENTRE LA PARTIE ACHEMINEMENT ET LA PARTIE CONTRÔLE DANS LES ROUTEURS

présenté par : NAIT TAKOUROUT Rachid
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. BOUCHENEB, Hanifa, Ph.D., présidente
M. PIERRE, Samuel, Ph.D., membre et directeur de recherche
M. QUINTERO, Alejandro, Doct., membre

REMERCIEMENTS

Je tiens à remercier M. Samuel Pierre, mon directeur de recherche, pour son soutien tout au long de ma recherche. De même, je remercie M. Laurent Marchand, M. Jon Maloy et toute l'équipe du Centre de Recherche LMC de Ericsson Canada pour les conseils et toute l'aide qu'ils m'ont apportée.

Surtout, je voudrais remercier mes parents, mes sœurs et mes amis pour m'avoir toujours supporté dans les moments difficiles.

RÉSUMÉ

Depuis l'avènement de Internet, les éléments de réseau, tels que les routeurs, se sont vus adjoindre beaucoup de fonctionnalités. Ils tendent à intégrer de plus en plus de services comme le support de la Qualité de Service ou encore des mécanismes de sécurité. Parallèlement à l'évolution des services implantés dans les éléments de réseau, les performances techniques des processeurs ont continué à croître rapidement. Ainsi, ces deux types d'évolution ont amené le concept de la séparation entre la partie *Contrôle* et la partie *Acheminement* dans les éléments de réseau. De plus, l'émergence de la technologie des processeurs réseau, capables de traiter les paquets avec de hautes performances tout en restant profondément programmables, apporte un véritable support à cette séparation.

Cependant, ce nouveau domaine de la réseautique nécessite le développement d'une architecture globale. Plusieurs organismes comme l'*Internet Engineering Task Forces* (IETF) se sont attelés à cette séparation; ils travaillent à définir les requis et les spécifications de ce concept naissant.

Dans le cadre de ce mémoire, nous étudions cette séparation en nous focalisant sur l'architecture et les requis formulés par l'IETF. Nous nous sommes intéressés plus particulièrement à la robustesse de l'architecture ainsi qu'à la problématique de la modélisation des données. En effet, nous proposons une solution aux requis relatifs à la robustesse en utilisant un protocole sous-jacent à l'architecture. De plus, nous apportons une syntaxe en XML pour modéliser d'une part les éléments propres à la partie *Contrôle* et à la partie *Acheminement* et, d'autre part, les types de données que ces deux parties échangent.

Nous avons mis en exergue cette architecture par le biais d'un prototype implémentant nos propositions de protocoles et de modélisation. Puis, nous avons testé cette plate-forme en fonction de cas de figure représentatifs des contraintes de robustesse et de flexibilité relativement à la modélisation.

Ainsi, nous obtenons des résultats probants de la fonctionnalité de nos propositions. Pour cela, nous avons simulé des pannes de connexion au sein d'un élément de réseau. Et, nous avons vérifié que les mécanismes de survivabilité que nous avons proposés ont pleinement réussi à assurer le fonctionnement de l'architecture.

De plus, nous avons réalisé des tests de performance sur les divers aspects de nos propositions. Nous avons mesuré les temps de reconnexion au sein de l'élément de réseau après une panne. Toujours dans le cadre de nos tests, nous avons quantifié l'effet sur les performances de l'utilisation de IPSec pour sécuriser les communications de notre architecture. Finalement, Nous avons mesuré l'impact sur les performances induit par notre modélisation en XML. Bien que ce type de syntaxe pour la modélisation amène une perte de performance, il nous permet d'avoir une plate-forme flexible qui soit véritablement capable de gérer l'hétérogénéité du matériel implémentant les fonctionnalités de la partie *Acheminement*.

ABSTRACT

Since the emergence of Internet, the network elements, such as the routers, were seen adding much functionalities. They tend to integrate more and more services like the Quality of Service support or security mechanisms. In the same way, the processors performances continued to grow quickly. Thus, these two evolution types brought the concept of the separation between the Control plane and the Forwarding plane in the network elements. Moreover, the developpment of the new Network Processors technology able to treat packets with high performances while remaining programmable, brings a real support to this separation.

However, this new networking area requires the development of a complete architecture. Several organizations as the IETF were interested in this separation, and work to define the requirements and the specifications of this incipient concept.

In this thesis, we will study this separation and more particulary the architecture and the requirements formulated by the IETF. We were interested in the architecture robustness and in the problems of data modeling. Indeed, we propose a solution to the robustness problematic by using a subjacent protocol for this architecture. Moreover, we bring a XML syntax to model on the one hand, the suitable elements of the Controle plane and the Forwarding plane and on the other hand, the data types that these two parts exchange.

We created a functional prototype implementing our protocols and a modeling proposals. Then, we tested this platform according to cases of figure representative of the robustness and modeling flexibility constraints.

Thus, we obtain significant results on the functionality of our proposals. Finally, we quantified the impact on the architecture performances which our XML modeling induces. Although this syntax type for modeling brings a loss of performance, it enables us to have a flexible platform which is truly able to manage the heterogeneity of the material implementing the Forwarding Plane.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES SIGLES ET DES ABRÉVIATIONS

Sigle ou abréviation	Signification
API	Application Program Interface
CE	ForCES Control Element
DiffServ	Differentiated Services
DOM	Document Object Model
ForCES	FORwarding and Control Element Separation working group
FE	ForCES Forwarding Element
Gb/s	Gigabit par seconde
IETF	Internet Engineering Task Force
IntServ	Integrated Services
IP	Internet Protocol
IPSec	IP Security Protocols
Mb/s	Megabit par seconde
MIB	Management Information Base
ms	Milliseconde
NPF	Network Processing Forum
OSI	Open System Interconnexion
PE	ForCES Protocol Element
QdS	Qualité de Service
RSVP	Resource reSerVation setup Protocol
SAX	Simple API for XML
TIPC	Telecom Inter Process Communication
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
W3C	World Wide Web Consortium
XML	Extensible Markup Language
µs	Microseconde

CHAPITRE I
INTRODUCTION

Depuis l'émergence d'Internet, les réseaux tendent à intégrer de plus en plus de fonctionnalités à chacun de leurs nœuds. En effet, des services appartenant traditionnellement au domaine de l'utilisateur tel que les serveurs Pare-Feu (*'Firewall'*) ou bien les services de Proxy se voient implémenter au niveau des éléments de réseaux. L'évolution des réseaux et de leurs protocoles remet en question l'aspect monolithique des éléments tels que les routeurs et nous amène à les concevoir comme des systèmes hiérarchiques et évolutifs. Pour l'exemple des routeurs, les équipementiers doivent leurs apporter des services tels que les mécanismes de Qualité de Service (IntServ et DiffServ), l'utilisation d'algorithmes de sécurité, de contrôle de congestion et de filtrage. D'autre part, ils doivent aussi prendre en compte les évolutions des protocoles de routage gérés par les organismes de normalisation. Ainsi, le concept de réseau programmable a pu se développer pour apporter une architecture qui soit à la fois dynamique, facilement extensible et performante [MERU 2000]. L'objectif, en fait, est de pouvoir faire évoluer les services apportés aux nœuds des réseaux tout en étant capable de travailler sur une architecture matérielle qui puisse être hétérogène, ce qui est l'objet de ce mémoire.

Dans ce chapitre d'introduction, nous présenterons d'abord quelques concepts de base nous permettant de préciser les éléments de la problématique, puis nous formulerons nos objectifs de recherche avant d'esquisser le plan du mémoire.

1.1 Définitions et concepts de base

L'évolution des routeurs et des services qu'ils proposent nous amène à nous intéresser à la structure logique d'un élément de réseau. Ce dernier peut être vu comme deux entités logiques coopérantes : la première nommée partie *Acheminement* qui reçoit, décompose, traite et réexpédie les paquets reçus, et la seconde appelée partie *Contrôle* qui s'occupe de contrôler les fonctionnalités et services applicables à ces données. La partie *Contrôle* se charge plus spécifiquement de la gestion des flots de données, des interfaces, de la fragmentation des paquets ou encore des statistiques. Il arrive que la gestion de haut niveau des éléments de réseau soit représentée en dehors de la partie *Contrôle* dans une partie dite de gestion. La Figure 1.1 illustre la représentation logique d'un nœud en tant qu'élément de réseau.

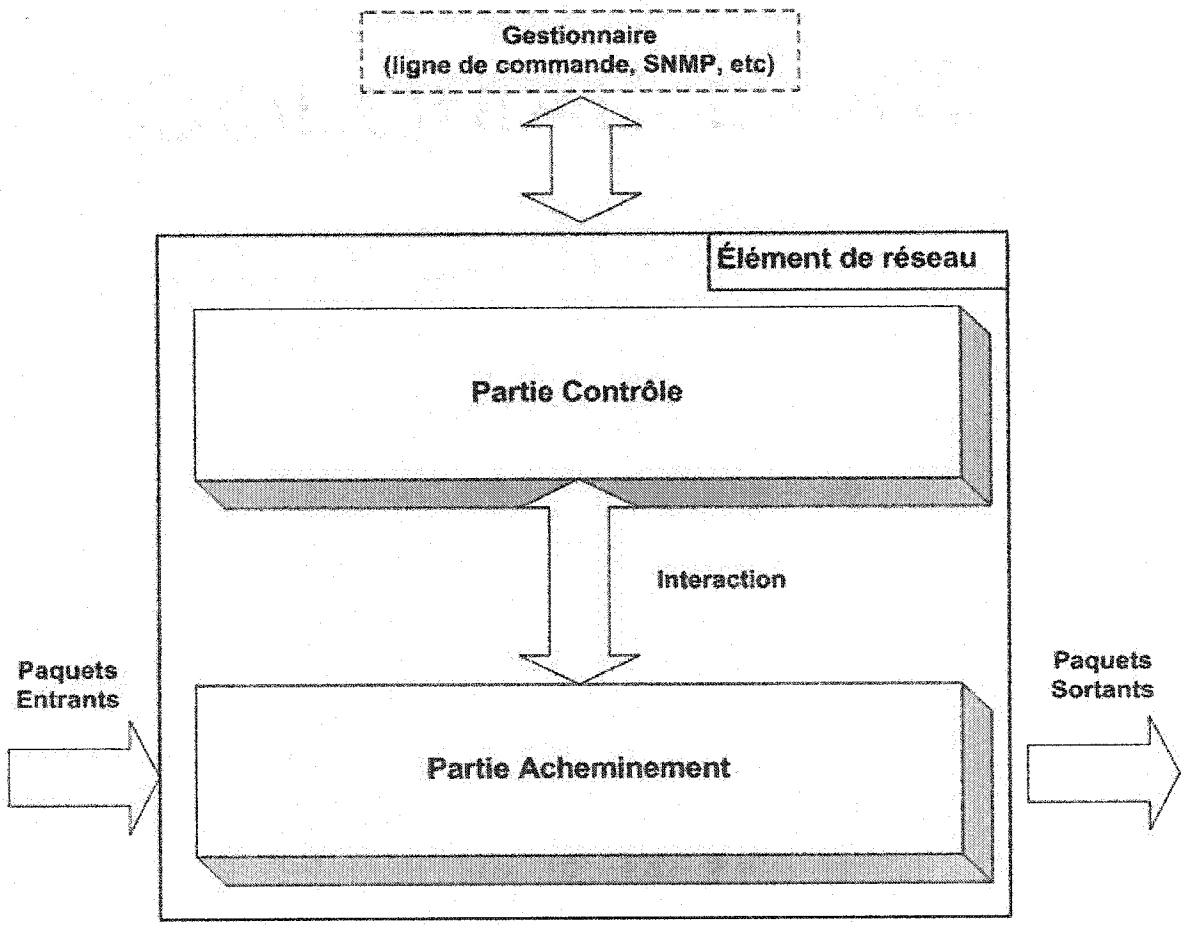


Figure 1.1 Représentation logique d'un élément de réseau

Sur cette illustration, nous retrouvons le fait que la partie *Acheminement* soit la seule entité à avoir un accès aux données que sont les paquets et qu'elle est entièrement assujettie à la partie *Contrôle*.

Généralement au cœur de la partie *Acheminement*, nous retrouvons des processeurs baptisés ASIC (*Application-Specific Integrated Circuits)* qui sont des circuits intégrés dont le fonctionnement est personnalisable. Ces circuits fournissent donc des fonctionnalités pour des tâches spécifiques comme la manipulation de paquets. Leur architecture leur permet d'être plus performant et de consommer moins d'énergie que les processeurs standard.

1.2 Eléments de problématique

Les parties *Contrôle* et *Acheminement* possèdent chacune leurs spécificités, et leurs priorités sont différentes. En effet, il est important de noter que la partie *Contrôle* doit être la plus flexible possible et que la partie *Acheminement* nécessite un traitement rapide, sachant qu'une connexion de type OC-48 a une bande passante de 2.488 Gb/s et celle de type OC-192 une bande passante de 9.953 Gb/s. L'architecture conventionnelle des routeurs basée sur les circuits intégrés de type ASIC permet d'obtenir des systèmes très rapides mais ils sont difficiles et très chers à développer. De plus, leurs spécialisations font qu'ils n'offrent que de faibles possibilités de programmation.

Il existe deux autres types d'architecture dans le sens d'un choix de support matériel combiné à une certaine implémentation logicielle, permettant d'apporter plus de flexibilité et d'évolutivité aux routeurs : les architectures basées sur des processeurs standards et celles basées sur les processeurs réseau.

L'architecture basée sur des processeurs standards conjointement liés à un support logiciel spécifique [YAN 2001] permet d'apporter la flexibilité recherchée au niveau de la partie *Contrôle*. Cependant, les performances de cette architecture en termes de nombre de paquets traités par seconde ne sont pas propices à son usage pour des routeurs très sollicités (pour des paquets d'une taille de 64 octets, un routeur logiciel tel que la plate-forme Pronto en traite 53500 à la seconde [HJAL 2000]). En effet, la structure des

processeurs standard (Pentium par exemple), bien que hautement programmable, n'est pas adaptée au traitement réseau. En effet, leurs capacités de calcul en virgule flottante ne sont pas vraiment utilisées et leurs largeurs de bande d'accès à la mémoire trop faibles.

L'architecture basée sur les processeurs réseau (*Network Processors*) est une évolution de l'architecture précédente. Elle se trouve à mi-chemin entre celle à base d'ASIC et celle de processeurs standards. En fait, un processeur réseau sert à manipuler les paquets à une vitesse élevée, tout en implémentant un ensemble de fonctions de Qualité de Service, de chiffrage et autres [DECA 2000][SPAL 2001]. Le principal attrait de ce genre de processeur vient de l'implémentation de ces fonctions. En effet, le processeur réseau n'implémente qu'une partie des piles de protocoles utilisés dans le nœud, uniquement celles qui requièrent un accès direct aux données et laisse à un processeur standard le reste de la gestion qui est plus complexe. Ce type de processeur possède une architecture matérielle caractérisée par :

- plusieurs processeurs amenant ainsi la possibilité d'avoir un traitement en parallèle ;
- des entités matérielles spécialisées pour les opérations réseaux ;
- un accès rapide à la mémoire ;
- un accès rapide aux périphériques réseaux ;
- un accès au processeur standard.

L'architecture à base de processeurs réseau répond aux exigences d'évolutivité. Mais comme cette technologie est très récente (les premiers processeurs réseau commercialisés ont vu le jour en 1999), le manque de standard pose le problème de la manière de gérer et de faire cohabiter différents types de processeurs réseau. De manière plus précise, le besoin de flexibilité dans l'évolution des parties *Contrôle* et *Acheminement* nécessite la création d'une véritable interface entre ces deux parties qui soit performante et capable de gérer l'hétérogénéité du matériel incarnant la partie *Acheminement*.

Dans la suite de ce mémoire, nous nous baserons sur l'architecture reposant sur les processeurs réseau. En effet, ce choix est naturellement motivé par les capacités de traitement et la programmabilité de cette architecture.

1.3 Objectifs de recherche

L'objectif principal de ce mémoire est de concevoir une architecture de communication flexible entre la partie *Contrôle* et la partie *Acheminement* des éléments de réseau émergeants intégrant des processeurs réseau. De manière plus précise, ce mémoire vise à :

- définir les requis essentiels d'un protocole de communication supportant efficacement la séparation des fonctions de contrôle et d'acheminement des équipements constituant un élément de réseau ;
- décrire formellement les fonctionnalités à implémenter dans la partie *Acheminement*, en vue d'une plus fine granularité dans la représentation de cette partie résultant en interactions plus efficaces avec la partie *Contrôle* ;
- implémenter et tester une architecture fonctionnelle synthétisant la séparation entre les parties *Contrôle* et *Acheminement*, architecture qui devra mettre en œuvre les mécanismes nécessaires à l'évolutivité mais aussi à la robustesse de la séparation de ces deux parties.

1.4 Plan du mémoire

Ce mémoire s'organise en cinq chapitres. Le second chapitre a pour vocation de définir les requis de la séparation des parties *Contrôle* et *Acheminement* et ce, par une étude des différentes composantes à la fois logicielles comme les piles de protocoles mais aussi matérielles comme les architectures des processeurs réseau disponibles.

Le troisième chapitre introduit les propositions formulées pour répondre à la problématique de la modélisation de la partie *Acheminement,* mais aussi de la robustesse des interactions entre les deux parties. Ensuite, dans le quatrième chapitre, nous verrons

la mise en œuvre des protocoles issus de nos propositions ainsi que les résultats de nos choix en termes de cas d'utilisation et de performances.

Enfin, nous conclurons ce mémoire par le cinquième chapitre qui résume nos principaux résultats et met l'accent sur les limitations de notre étude et les différentes possibilités de recherche à venir.

CHAPITRE II
ARCHITECTURE GÉNÉRALE DES PARTIES CONTRÔLE ET ACHEMINEMENT

La dissociation entre la partie *Contrôle* et la partie *Acheminement* passe préalablement par une étude détaillée des fonctionnalités et des contraintes inhérentes à chacune d'elles. Ainsi, dans le présent chapitre, nous nous focaliserons sur chacune de ces parties ainsi que sur les différents requis de leurs interactions. En effet, nous verrons qu'au-delà de la séparation en tant que tel, le cœur de la problématique motivant ce mémoire se trouve être la définition d'un médium de communication entre ces deux parties et d'apporter les mécanismes de flexibilité nécessaires à l'émergence de routeurs programmables à haute performance. Ainsi, dans un premier temps, nous allons définir l'architecture générale de cette séparation, ensuite nous discuterons des particularités de la partie *Acheminement* et des caractéristiques de la partie *Contrôle*. Puis, nous présenterons les interactions entre ces deux parties. Enfin, nous discuterons de technologies connexes permettant d'apporter une solution à notre problématique.

2.1 Architecture générale

La séparation effective entre les parties *Contrôle* et *Acheminement* nécessite de définir une architecture générale ainsi qu'une terminologie adaptée. Dans ce sens, le groupe de travail ForCES de l'IETF a proposé une architecture [YANG 2003a], dont nous présentons une illustration à la figure 2.1. Celle-ci est basée sur un ensemble d'éléments de base :

- '*Forwarding Element*' (FE): C'est une entité logique utilisant les fonctionnalités du matériel afin de pouvoir manipuler et traiter chaque paquet, conformément aux exigences du 'Control Element' et implémentant le protocole ForCES. Ce dernier est la notation de l'ensemble des protocoles régissant les interactions entre la partie *Contrôle* et la partie *Acheminement.*
- '*Control Element*' (CE) : C'est une entité logique utilisée pour fournir au(x) FE(s) la manière de traiter les paquets et implémentant aussi le protocole ForCES.
- '*FE Manager*' (FEM) : Entité logique opérant durant une phase de pré-association (période durant laquelle on détermine quel FE et CE appartient au même élément réseau), responsable de déterminer quel FE et CE peuvent communiquer ensemble. Le FEM doit pouvoir connaître les capacités des CEs disponibles.
- '*CE Manager*' (CEM) : De la même manière, cette entité logique est responsable aussi du choix des CEs et FEs pouvant communiquer, et doit pouvoir connaître les capacités des FEs disponibles.

Les CEs et FEs sont aussi dénommés *Protocol Elements* (PE). Ces éléments sont bien entendus amenés à communiquer entre eux. Les différents types de communication entre éléments peuvent être vus de la manière suivante :

- Interaction entre CEs :

  L'usage de plusieurs CEs dans le même élément de réseau permet d'introduire des mécanismes de redondance d'informations (pour plus de fiabilité en cas de panne d'un CE), de contrôle distribué (lorsqu'une requête par exemple venant d'un FE ne peut être traitée que par un CE spécifique), ou bien de partage de charge (si une requête peut être traitée par plus d'un CE, il faut décider lequel s'en chargera). Ainsi, ces mécanismes requièrent une communication entre les CEs.

- Interaction entre FEs :

  D'une manière semblable, il peut y avoir des communications entre FEs. En effet, un paquet arrivant par un certain FE peut devoir être renvoyé par un autre FE.

- Interaction entre un CEM et un ou des CEs :

  Un CEM doit être capable de configurer un CE en fonction des informations sur les FEs.

- Interaction entre un FEM et un ou des FEs :

  Le FEM assigne à un ou des FE(s) un CE avec lequel ils puissent communiquer.

- Interaction entre CEM et FEM :

  Naturellement, l'opération d'assignation des relations entre CE et FE introduit une communication entre FEM et CEM où ils échangent leurs listes de CEs et FEs disponibles.

- Interaction entre un CE et un FE :

  Après leur configuration à l'aide des informations pour se contacter (via les FEM/CEM), un CE et un FE doivent communiquer entre eux selon deux cas de figure. Dans le premier dit *établissement de l'association*, le FE doit informer le CE de ses propres capacités et de sa topologie vis-à-vis des autres FEs, afin de définir le genre de traitement de paquet qu'il est susceptible de fournir et de connaître ses paramètres configurables ainsi que les statistiques sur son état. Cet échange introduit la notion de modèle servant à la description des capacités du FE, sur laquelle nous reviendrons plus tard. Le second cas de figure correspond à l'échange d'information relative à la manipulation du FE par le CE (ajout de route dans la table de routage, état d'un lien du FE, statistiques du FE, etc.). Nous pouvons aussi noter que les paquets de contrôle (RIP, OSPF, etc.) doivent être redirigés par le FE vers le CE et vice versa.

- Interaction entre un FE et le lien réseau :

  Ce type de communication représente simplement l'interaction entre un FE et ses interfaces liées au réseau.

Afin de bien synthétiser cette architecture, nous pouvons la schématiser comme à la Figure 2.1 pour le cas simple d'un élément de réseau possédant deux CEs et deux FEs. Les différentes interactions présentées précédemment y sont indiquées par des lignes en pointillé.

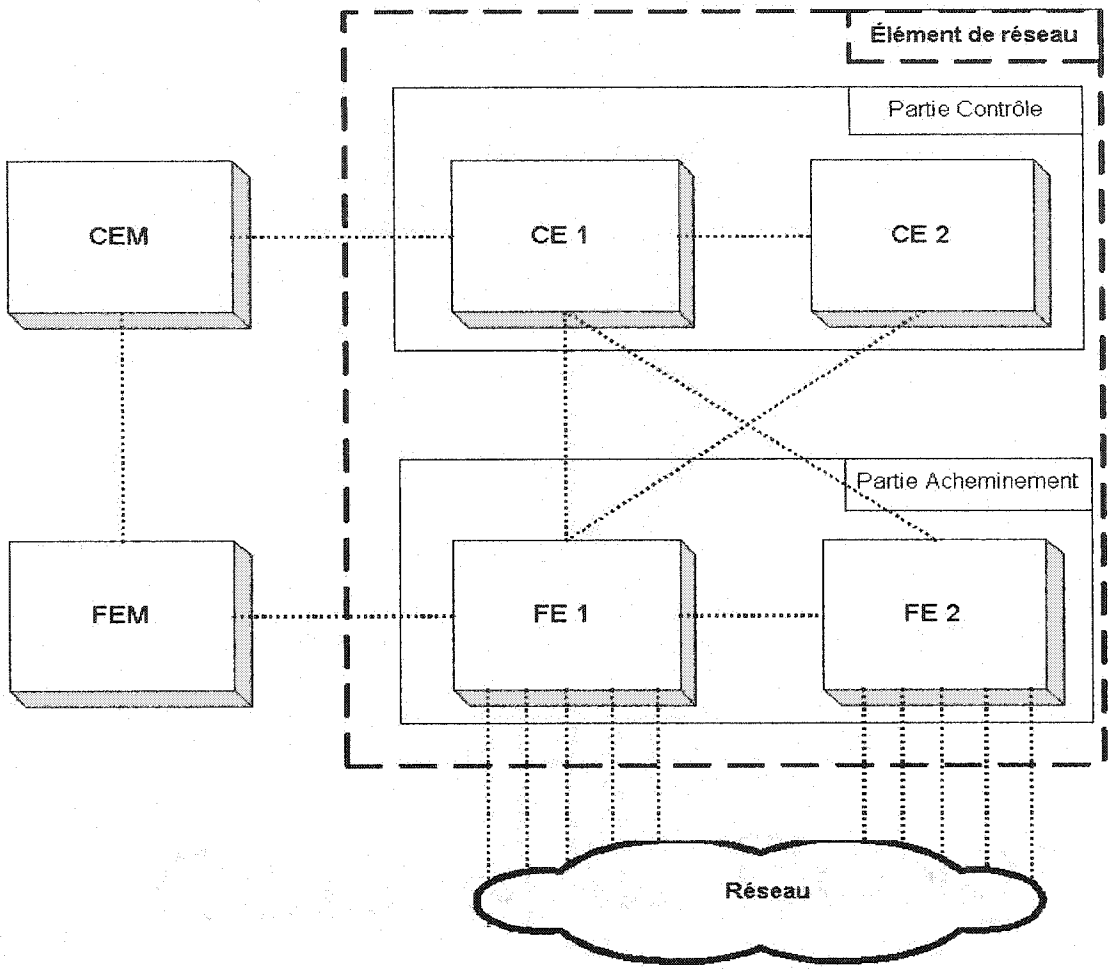


Figure 2.1 Exemple d'architecture d'un élément de réseau

2.2 Fonctionnalités de la partie *Acheminement*

La partie *Acheminement* doit pouvoir répondre à un ensemble de caractéristiques fonctionnelles. La difficulté du concept de la partie *Acheminement* vient en fait de la variété de ses fonctionnalités. En effet, la partie *Contrôle* doit être en mesure de comprendre la manière dont la partie *Acheminement* traite les paquets afin de la gérer. Le groupe de travail ForCES de l'IETF a ainsi défini un modèle de fonctionnalité pour un élément de la partie *Acheminement* [YANG 2003b]. Il faut noter que c'est une proposition de modèle, car il peut en exister plusieurs en fonction du domaine

d'utilisation du FE. Nous pouvons néanmoins définir un ensemble minimum de fonctions qu'un modèle de FE doit fournir, tout en sachant que tous les FEs répondant à ce modèle à l'intérieur d'un élément réseau n'ont pas obligatoirement à les implémenter en totalité.

Maintenant que nous avons vu le concept de FE et quelques unes de ses fonctionnalités, nous allons nous intéresser aux processeurs réseau remplissant en fait ce rôle. Tout d'abord, il faut remarquer que ce genre de processeur est soumis à d'importantes contraintes de traitement. En effet, ils doivent pouvoir traiter un paquet en un temps très limité. Un processeur cadencé à 232 MHz gérant des paquets IP d'une taille de 40 octets avec une liaison OC-48, en reçoit tous les 35.5 cycles d'horloge. Ainsi, bien qu'il existe de nombreuses architectures de processeur réseau, ils possèdent tous en commun le fait d'être multi-tâches pour gérer cette contrainte de temps. Pour cela, ils disposent de plusieurs entités de traitement fonctionnant en parallèle qui sont en fait des pico-processeurs. Les principaux processeurs réseau sur le marché sont le C-Port™ (C-5 et C-5e) de Motorola©, le IXP 28xx/24xx™ d'Intel©, l'AgËre™ de Lucent©, le IQ2000™ de Vitesse© ou encore le PowerNP™ de IBM©. Le modèle de la firme Intel© est l'un des processeurs réseau les plus répandus, grâce aux performances de son architecture (*'Internet eXchange Architecture'*) et aussi du fait que Intel© documente de manière libre ce produit. Dans le cadre de cette présentation des processeurs réseau, nous allons nous restreindre au cas du IXP 2800 d'Intel© [ADIL 2002].

Comme l'illustre la Figure 2.2, ce processeur combine un processeur XScale™ avec seize Micro-Machines (*'Micro-Engine'* noté ME sur la figure) 32-bits RISC indépendantes, des interfaces vers des mémoires SRAM/SDRAM, d'un bus PCI ainsi que d'une interface vers le lien physique (SPI-4) ou vers un *Switch Fabric* (CSIX) regroupés sous le terme *Media Switch Interface* (MSF). Ces Micro-Machines peuvent traiter des tâches généralement réservées aux processeurs ASIC. Le processeur XScale™ basé sur une architecture de type RISC 32 bits cadencé à 700 MHz est destiné aux tâches plus complexes telle que la maintenance des tables de routage.

L'interface appelée MSF sert à connecter le IXP aux périphériques du niveau physique et le cas échéant à un *Switch Fabric*. Ce dispositif se décompose en deux interfaces, une qui reçoit et l'autre qui renvoie. Cette interface supporte 15 Gb/s de trafic entrant et 10 Gb/s de trafic sortant (ou bien 15 Gb/s sortant et 10 Gb/s entrant); ce déséquilibre de performance semble due au design du *Switch Fabric*.

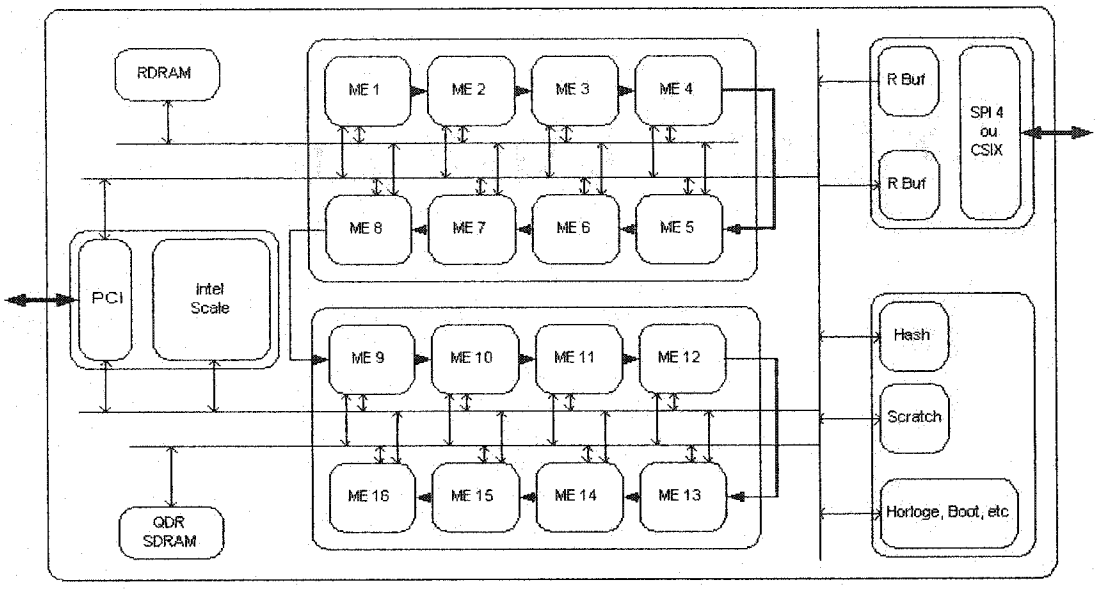


Figure 2.2 Architecture du processeur réseau IXP 2800 de Intel©

La particularité de ce processeur réseau est l'utilisation des Micro-Machines, dont les spécifications ont été guidées par les requis suivants :

- une fréquence d'horloge élevée pour permettre suffisamment d'instructions par paquet ;
- un grand nombre de registres pour minimiser le nombre de variables à faire passer des registres à la mémoire et vice-versa ;
- une mémoire locale avec une faible latence d'accès pouvant servir à garder les paquets ou encore l'état d'un port ;
- un mécanisme de communication performant entre les Micro-Machines ;
- la gestion multi-tâches.

Ainsi, chaque Micro-Machine possède 4 kilo-octets de mémoire dédiée à son micro-code ainsi que quatre types de registres de 32 bits : 256 registres à usage général,

512 registres de transfert servant les unités SRAM et SDRAM, ce qui réduit la latence des instructions, 128 registres d'accès à la Micro-Machine précédente et la mémoire locale de 640 mots de 32 bits. Nous pouvons aussi noter qu'il n'existe pas, au niveau matériel de FIFO d'entrée-sortie pour les paquets, en fait c'est au programmeur d'utiliser des registres pour les implémenter.

Les Micro-Machines supportent chacune huit contextes matériels pouvant commuter en une seule instruction. Ceci permet d'obtenir un bon recouvrement des calculs avec les accès mémoire mais divise par huit le nombre de registres par contexte. Leur programmation s'avère complexe, nécessitant pour le codage le recours au langage assembleur ('*IXA Mev1 Symbolic Microcode*') ou bien en utilisant un compilateur C propriétaire d'Intel©. Cependant, il existe des solutions logicielles permettant de faciliter la programmation des Micro-Machines en générant le micro-code de manière automatique, notamment la plate-forme Teja NP™ de la firme Teja© [TEJA 2002]. Autre particularité de cette architecture, les seize Micro-Machines sont organisées en deux grappes, comme illustré à la Figure 2.2.

D'autre part, nous pouvons noter qu'il existe une version de ce processeur réseau nommé IXP 2850™ disposant de fonctionnalités cryptographiques nécessaires pour l'usage par exemple d'IPSec ou de VPNs. En effet, le IXP 2850™ dispose de deux unités cryptographiques couplées au contrôleur de la RDRAM. Chacune de ces unités possède deux blocs basés sur l'algorithme 3DES/DES (*Data Encryption Standard*) et un sur AES (*Advanced Encryption Standard*) pour le chiffrage et le décryptage des paquets, ainsi que deux blocs basés sur SHA-1 (*Secure Hash Algorithm*) pour l'authentification des données.

2.3 Fonctionnalités de la partie *Contrôle*

L'élément central de la partie *Contrôle*, comme nous avons pu le voir, est le 'Control Element' (CE). En effet, le CE a pour objectif de contrôler le comportement du ou des FEs et ainsi doit implémenter les protocoles de routage [BAKE 1995]. Or ces protocoles nécessitent que les routeurs communiquent entre eux. Ce type de

communication, en fait entre CEs, peut transiter par le biais de FEs (ils redirigent les paquets de données destinés aux routeurs).

La partie *Contrôle* doit supporter les mécanismes de routage, de Qualité de Service et de gestion de liste d'accès. Dans la suite de cette section, nous allons voir les principales d'entre eux.

2.3.1 Protocoles de contrôle et de signalisation

Au-delà du cas des réseaux de très petites tailles ne nécessitant qu'un routage statique, le routage se base sur des mécanismes dynamiques que l'on peut classer en deux grandes familles, en fonction de la topologie du réseau : ce sont les protocoles intra-domaines ('*Interior Gateway Protocol*') et les protocoles inter-domaines ('*Exterior Gateway Protocol*').

Les protocoles intra-domaines se limitent au routage à l'intérieur d'un Système Autonome (SA) qui est une zone de routage autonome. Ces protocoles sont basés sur l'algorithme de vecteur distant (parfois appelé algorithme de Bellman-Ford) ou bien d'état de liens (qui est une adaptation de l'algorithme de Dijkstra). Deux protocoles sont généralement utilisés RIP et OSPF.

Le protocole RIP ('*Routing Information Protocol*') [HEDR 1988] utilise l'algorithme de vecteur distant (dont la métrique de choix est ici le nombre de sauts entre deux routeurs sans tenir compte de la qualité de la liaison). Un routeur utilisant RIP envoie sa table de routage toutes les 30 secondes à ses voisins qui mettent à jour la leur et les transmettent aux suivants. Ce processus continue jusqu'à la convergence du réseau. Il existe deux versions de ce protocole (sa seconde version [MALK 1998] apporte surtout le support des masques de longueurs variables) pour IPv4 et une version adaptée à IPv6 RIPng. RIP a comme principaux désavantages de consommer beaucoup de bande passante (du à l'envoie des tables de routage) et de converger lentement.

Le protocole OSPF ('Open Shortest Path First') [MOY 1998] utilise l'algorithme d'état de lien (dont la métrique est fonction de bande passante; pour l'illustrer, les routeurs Cisco© utilisent une métrique égale à 10^8 divisé par la bande passante du lien).

Avec OSPF, les Systèmes Autonomes sont divisés en zone et dans chacune de ces zones, les routeurs possèdent la même table de routage. Ce découpage en zones de taille restreinte s'explique par le fait que l'algorithme d'état de lien est très gourmant en ressources (en général une zone ne contient pas plus de 50 routeurs). OSPF possède la particularité de consommer peu de bande passante et de permettre d'avoir une hiérarchie au niveau du routage. Ce protocole en est à sa deuxième version et remplace dans l'industrie le protocole RIP.

D'autre part, les protocoles inter-domaines sont des protocoles utilisés pour le routage entre Systèmes Autonomes. Ils distribuent les paquets de manière optimale en considérant chaque SA traversé comme une boite noire, tout en tenant compte de leurs politiques de routage. Chaque Système Autonome spécifie, dans sa politique de routage, la liste des réseaux qui peuvent l'utiliser comme SA de transit. Le principal protocole de routage inter-domaine est BGP (*'Border Gateway Protocol'*) [REKH 1995]. L'algorithme de ce protocole est un algorithme de vecteur distant modifié. Ainsi, les routeurs calculent pour chaque destination la route la plus courte en terme de SA traversés en fonction des politiques qu'ils ont annoncées à leurs voisins. Il existe deux sortes de connexions BGP, les eBGP (entre routeurs de deux SA différents) et iBGP (tout routeur d'un SA est connecté avec les autres routeurs de ce SA pour se communiquer les routes reçues des autres SAs).

2.3.2 Les Protocoles de Qualité de Service

La Qualité de Service (QdS) peut être définie comme l'effet d'un service qui détermine le degré de satisfaction d'un utilisateur. Les métriques nécessitant d'être contrôlées afin d'assurer un certain niveau de QdS sont de manière non exhaustive : le délai de bout en bout, la variation de délai, le taux de perte de paquets, la bande passante minimale, la disponibilité d'un réseau, etc. Les architectures pour assurer une Qualité de Service garantie sont de trois grands types : l'Intégration de Services (*'Integrated Services'* noté IntServ [BRAD 1994]), la Différentiation de Services (*'Differenciated*

Services' noté DiffServ [BLAK 1998]) et MPLS (*'Multi-Protocol Label Switching'* [ROSE 2001]).

2.3.2.1 Modèle IntServ

Le modèle de *IntServ* repose sur deux concepts fondamentaux, le premier étant que les réseaux doivent être contrôlés et soumis à des mécanismes de contrôle d'admission, le second étant que des mécanismes de réservation sont nécessaires pour réussir à fournir des services différenciés. Ainsi, *Intserv* propose deux sortes de service de trafic en plus du traditionnel *Best Effort* afin de gérer le trafic temps-réel dans un cadre *unicast* aussi bien que *multicast*. Il s'agit du Service Garanti (*'Guaranteed Service'*) qui assure une bande passante garantie et un délai d'acheminement limité, et le Service Contrôlé (*'Controlled Load'*) qui est équivalent à un service *Best Effort* dans un environnement non congestionné (correspondant aux services temps réels supportant une variation de délai). Les ressources réseau, telles que la bande passante, doivent être gérées explicitement en fonction du type d'application, ce qui induit une réservation de ressource et un contrôle d'admission. Pour cela, *IntServ* s'appuie sur la notion de flot de données correspondant à un ensemble de paquets issu d'une application utilisatrice et nécessitant une certaine QdS. Ce modèle définit donc les éléments nécessaires pour assurer ces services. Ce sont l'ordonnanceur de paquets (*'Packet Scheduler'*), le classificateur de paquets (*'Packet Classifier'*), le contrôle d'admission (*'Admission Control'*) et la réservation de ressource (*'Ressource Reservation'*). D'autre part, la réservation de ressources est réalisée par le biais du protocole RSVP (*'Resource reServation Protocol'* [BRAD 1997]). RSVP a pour objectif d'allouer dynamiquement de la bande passante (car des messages stipulant l'état du chemin à travers les routeurs sont émis périodiquement) aux applications orientées réseau. Il est particulièrement utile pour les applications multimédia, pourtant il résiste mal, de par sa structure, au facteur d'échelle et se limite donc à un réseau de petite taille.

2.3.2.2 Modèle DiffServ

Le modèle *DiffServ* consiste à classer le trafic grâce à un champ présent dans le paquet IP (pour la version 4, il s'agit du champ TOS *'Type of Service'* ou bien, pour la version 6, du champ Classe de Trafic '*Traffic Class*'). On applique ensuite des traitements adaptés aux différentes classes de trafic. Cela revient à avoir une granularité moins fine que sous *IntServ*, mais qui devient en revanche moins sensible au facteur d'échelle. Les opérations de classification, contrôle et marquage sont effectuées par les routeurs périphériques (*'Edge Router'*), tandis que les routeurs centraux (*'Core Router'*) traitent les paquets en fonction de sa classe selon un comportement spécifique, le PHB (*'Per Hop Behavior'*). Ce comportement peut être de deux types : soit *'Expedited Forwarding'* qui a pour but de garantir une bande passante avec des taux de perte, de délai et de gigue faible; soit *'Assured Forwarding'* regroupant plusieurs PHBs garantissant un acheminement de paquets IP avec une faible possibilité d'erreur (cette famille de PHB est scindée en 4 classes garantissant une bande passante et un délai minimum, chaque classe comprenant 3 niveaux de priorité appelé *'Drop Precedence'*). Ces comportements sont implémentés dans les routeurs par le biais de mécanisme de gestion de file d'attente tel que WFQ (*'Weighted Fair Queuing'*).

L'architecture de ce modèle est en fait basée sur deux notions, les *domaines DiffServ* et les *régions DiffServ*. Les domaines regroupent un ensemble de nœuds contigus partageant un service commun de règles et de PHBs. D'ailleurs, les nœuds d'un domaine comprennent soit des nœuds frontières (*DS Boundary Node*) qui peuvent être de type entrant ou sortant (*DS Ingress Node* et *DS Egress Node*), soit des nœuds intérieurs. Les nœuds frontières réalisent les opérations les plus complexes, comme la classification. Les régions DiffServ, quant à elles, se trouvent être un ensemble de domaines contigus dont la QdS doit être assurée de bout en bout. Cela nécessite de mettre en place des agréments entre domaines nommés *Service Level Agreement* (SLA) pour permettre une continuité des services aux travers des domaines. Chaque SLA définit un contrat (*Traffic Conditioning Agreement* TCA) spécifiant les règles de

classification et de manipulation du trafic servant au conditionnement à la frontière de deux domaines.

Comme nous l'avons mentionné précédemment, les fonctions les plus complexes sont celles du conditionnement du trafic dans les nœuds frontières. En effet, le comportement de tels nœuds se définit par un ensemble de blocs de conditionnement de trafic (*Traffic Conditioning Bloc* TCB). Chacun de ces TCBs consiste en un chemin de donnée logique comportant des blocs fonctions interconnectés, comme l'illustre la Figure 2.3.

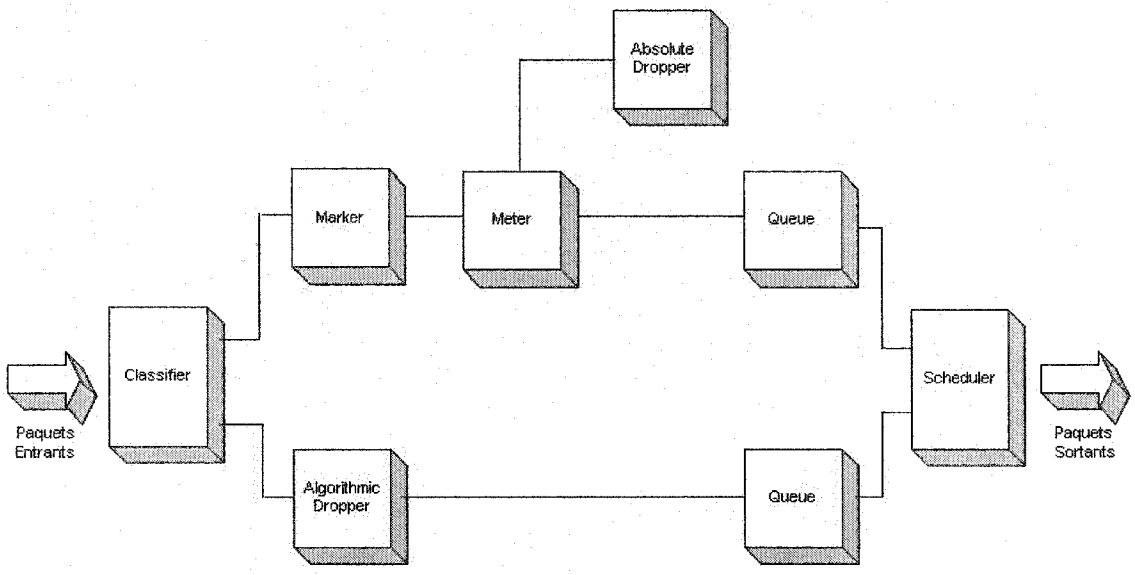


Figure 2.3 Exemple de blocs fonctionnels DiffServ

Le modèle Diffserv articule ces blocs en six grands types de fonctions traitant les paquets :

- *Classifier* : Un filtre de paquets (utilisant les champs de l'entête IP) qui divise un flux entrant en différents flux de paquets ;
- *Meter* : Un bloc qui mesure les paquets et qui si une condition préétablie est vérifiée, redirige les paquets vers un bloc spécifique ;
- *Action* : Fonction de traitement que l'on applique à un paquet. Il existe différents types de bloc *Action* : élimination d'un paquet (*Absolute Dropper*), marquage d'un paquet (*Marker*), comptage de paquets (*Counter*), multiplexage de

différents flux en un seul (*Multiplexor*) ou bien aucune action n'a lieu (*Null Action*) ;

- *Algorithmic Dropper* : Élimine un paquet relativement à un algorithme particulier ;
- *Queue* : Une queue de paquets ;
- *Scheduler* : Un ordonnanceur de paquets appliquant un algorithme donné.

2.3.2.3 MPLS (*Multi-Protocol Label Switching*)

MPLS est un protocole pour commuter le trafic IP indépendamment de la technologie de niveau 2 (de la couche OSI) qui peut être entre autres, ATM ou bien Relais de Trame. La transmission de données sous MPLS se fait sur des chemins à commutation d'étiquettes (*'Label-Switched Paths'* noté LSP). Les LSPs sont une séquence d'étiquettes (*label*) à chaque nœud du chemin allant de la source à la destination. Les LSPs sont établis en fonction du type de transmission des données (*'control-driven'*) ou après détection d'un certain type de données (*'data-driven'*). Les étiquettes, qui sont des identifiants spécifiques aux protocoles des couches plus basses dans la couche OSI, sont distribuées suivant le protocole LDP (*'Label Distribution Protocol'*), RSVP ou parfois par les protocoles de routage comme BGP ou OSPF. Chaque paquet de données encapsule et transporte les étiquettes le long de leur acheminement. La commutation à haut débit est possible puisque les étiquettes de longueur fixe sont insérées au tout début du paquet ou de la cellule et peuvent être utilisées par le matériel pour commuter plus rapidement.

2.4 Interaction entre les parties *Contrôle* et *Acheminement*

La diversité des mécanismes présents dans la partie *Contrôle* et la partie *Acheminement* pose évidemment la question de leurs interactions. Dans cette section, nous verrons les requis fonctionnels essentiels mais aussi les méthodologies de modélisation ainsi que les mécanismes de communications qui ont été proposés à date.

2.4.1 Requis fonctionnels de l'architecture

À la vue des fonctionnalités requises par les parties *Contrôle* et *Acheminement*, il nous faut les synthétiser conformément aux concepts de FE et de CE. Le groupe de travail ForCES de l'IETF a défini un certain nombre de requis nécessaires pour les fonctionnalités au sein du document ietf-forces-requirement [KHOS 2003] :

- CEs et les FEs doivent être capables de s'interconnecter par plus d'une sorte de technologie (Ethernet, ATM, etc.). D'ailleurs, les FEs peuvent être interconnectés entre eux avec une technologie autre que celle utilisée pour les communications CE/FE.
- Les FEs doivent posséder un ensemble minimum de fonctionnalités afin d'établir une connectivité réseau (découverte d'interface, etc.) sans pour autant limiter le nombre et le type de fonctionnalités qu'un FE peut supporter.
- Les paquets arrivant dans un élément de réseau par un FE doivent être capables d'être acheminés par un autre FE.
- Un élément de réseau doit apparaître comme une entité fonctionnelle atomique.
- L'architecture doit comprendre des mécanismes de sécurité empêchant un élément non autorisé de se joindre à l'élément de réseau.
- Un FE doit être capable de prévenir de manière asynchrone le CE d'une panne, d'une variation de ses ressources disponibles ou bien de ses capacités. De la même manière, le FE doit gérer ses rapports d'erreurs.
- Les FEs doivent être capables de rediriger les paquets de contrôle (messages issus de RIP, OSPF, etc.) adressés à leurs interfaces vers leur CE, ainsi que les paquets spécifiques au routage (*Router Alert Option*). De même, les CEs doivent pouvoir configurer les informations et les filtres de redirection des FEs, mais aussi créer des paquets pour que les FEs les leurs délivrent.
- Chaque architecture proposée devra expliciter comment sont supportées les fonctionnalités des routeurs comme établies dans les spécifications de la RFC 1812.

- Au sein d'un élément de réseau, un FE doit pouvoir fournir aux CEs les informations relatives à sa topologie (connexions avec les autres FEs).
- L'architecture d'un élément de réseau doit supporter plusieurs CEs et FEs.
- Bien qu'en dehors des requis et spécifications actuelles du ForCES, les mécanismes relatifs à la phase de pré-association pourraient utiliser des outils standards de gestion pour les CEs et les FEs, durant la période de post-association, les tâches de gestion devraient être réalisées par le biais du CE par des outils tel que SNMP mais doivent suivre les deux prérogatives suivantes : Un outil de gestion peut être employé pour lire (mais pas pour modifier) l'état du FE, et il ne doit pas être possible que les outils de gestion changent l'état d'un FE sans notifier le CE.

En plus de ces requis fonctionnels généraux, le groupe de travail ForCES énumère trois principaux points concernant la robustesse de l'architecture :

- Support du facteur d'échelle : L'architecture doit supporter aux moins des centaines de FEs et des dizaines de milliers de ports ;
- Association dynamique : L'architecture doit permettre aux CEs et FEs de joindre ou de quitter dynamiquement un élément de réseau ;
- La redondance des CEs : L'architecture doit supporter la redondance des CEs par le biais de mécanismes de détection de perte de connexions et de restauration de communication et de resynchronisation des états.

Ce dernier point introduit la problématique de la cohérence des données et la synchronisation des états des CEs et des FEs. Mais, dans ce mémoire, nous nous limiterons à l'étude de la robustesse vis-à-vis du protocole de communication sous-jacent à l'architecture ForCES.

2.4.2 La phase de pré-association

Nous avons vu que le groupe de travail ForCES définit la phase de pré-association comme la période où un CE Manager et un FE Manager ont à déterminer quels FEs et quels CEs sont susceptibles d'appartenir au même élément de réseau. Mais, cette phase

représente en fait la politique globale régissant le comportement de l'élément de réseau vis-à-vis des capacités des FEs et CEs. En effet, comme nous l'expliquions précédemment, les CE Managers et FE Managers doivent indiquer à leurs entités respectives des propositions d'affiliation CE - FE. Au-delà de cet aspect, nous pouvons aussi y voir un moyen d'automatiser l'adjonction d'un FE ou d'un CE.

Nous pouvons d'ailleurs nous interroger sur la pertinence de dissocier la gestion des FEs et celle des CEs. Telle qu'énoncée dans le document ietf-forces-framework [YANG 2002a], cette dichotomie permettrait au CE Manager de ne divulguer les caractéristiques de ses CEs qu'à certains FE Managers, et réciproquement. Nous pouvons considérer ce type de mécanisme comme une manière pour le propriétaire d'une Partie *Contrôle* (ou *Acheminement*) de vendre ses fonctionnalités à une tierce personne. Cette vision nécessite donc des mécanismes de sécurité.

2.4.3 Modélisation d'un FE

La modélisation des FEs se trouve au centre des pré-requis essentiels afin de permettre une véritable communication entre un CE et un FE. En effet, afin qu'un CE contrôle de manière effective un FE, il se doit de connaître la manière dont ce dernier manipule et traite un paquet. Mais comme nous avons pu le constater le long de ce chapitre, il n'est pas aisé de définir un modèle évolutif, concis et exhaustif de ces fonctionnalités qui sera noté *FE Model*. Le groupe de travail ForCES a avancé certains requis sur ce thème. Nous allons ici reprendre le contenu afin de mieux synthétiser le cadre de cette modélisation sur laquelle nous reviendrons dans le troisième chapitre. La principale caractéristique de cette modélisation est la dichotomie effectuée entre la capacité de traitement (fonctions logiques) des paquets qui est appelé le modèle de capacité, et la configuration courante du FE dénommée le modèle d'état.

Ainsi, ce modèle doit définir quels types de fonctions sont appliqués à un paquet lorsqu'il traverse un FE. Nous pouvons retrouver des fonctions tel qu'un pare-feu ou bien encore un *Shaper* Diffserv. Le groupe de travail de l'IETF a en fait défini un ensemble de fonctions logiques qui doit au minimum être implémenté :

- Fonctions de port : Le modèle se doit de décrire le nombre de ports sur un périphérique ainsi que ses attributs statiques mais aussi dynamiques.
- Fonctions d'acheminement réseau : La représentation des données servant à prendre des décisions de routage doivent être présente dans le modèle, tout particulièrement les mécanismes *unicast* et *multicast* de IPv4 et IPv6.
- Fonctions de Qualité de services : Le modèle doit transcrire les capacités de Qualité de Service des modèles *IntServ* et *DiffServ* au sein de ses fonctions.
- Fonctions de filtrage : La présence de fonctions de filtrage avancées sur les paquets est aussi nécessaire ainsi que les manipulations effectuées après le filtrage et la classification.
- Fonctions spécifiques aux vendeurs : Le modèle devrait être suffisamment flexible pour expliciter les fonctionnalités propres aux vendeurs, mais cela ne devrait pas permettre d'implémenter de manière propriétaire des fonctionnalités standards.
- Fonctions d'encapsulation et de tunnel : Il se doit aussi de pouvoir exprimer la manière d'encapsuler les données par le biais de fonctions de marquage de classe de trafic mais aussi peut implémenter des fonctions tel que NAT ou ALG.
- Fonctions de sécurité : Le modèle doit spécifier le type d'encryptage appliqué aux paquets.
- Fonctions transférées du CE vers le FE : Les FEs doivent pouvoir exécuter de manière asynchrone des fonctions aux paquets dans l'optique de transférer des fonctions du CE vers un FE. Pour illustrer ce requis, il suffit de concevoir l'exécution de mécanisme comme les messages HELLO de OSPF qui sont ordonnés indépendamment d'un événement lié à un paquet reçu.

À la vue de cet ensemble minimal de types de fonctions, il est important de noter que le modèle se doit d'offrir une certaine latitude dans leur implantation. En effet, il est facile d'imager qu'un FE implémente la table de routage de niveau 3, mais aussi la table de correspondance des adresses MAC dans les mêmes blocs de fonctions, alors qu'un autre pourrait les dissocier.

De même, il faut aussi se focaliser sur la manière dont le modèle décrit l'agencement de ces fonctions. En fait, l'ordre par lequel on applique les fonctions sur un paquet est essentiel comme on peut le remarquer par exemple pour les fonctions de conditionnement de *DiffServ*. Pour conclure sur ces requis, ce modèle se trouve dans l'obligation de pouvoir exprimer de manière générique les statistiques relatives à chacune de ces fonctions pour permettre son usage dans un environnement hétérogène.

2.4.4 La phase de post-association

Une fois que nous avons vu les fonctionnalités propres à chacune des deux parties, il est important de définir les mécanismes nécessaires à leur interaction. En effet, nous pouvons concevoir cette interaction comme un rapport maître/esclaves entre un CE et un ou plusieurs FEs. À la lumière des deux précédentes parties, nous définissons plusieurs cas de figure d'usage de ce type d'interaction :

- Contrôle des caractéristiques directes d'un FE par un CE : Dans ce cas de figure, nous retrouvons le contrôle du CE sur les FEs telle que la modification de la table de routage, des caractéristiques des files d'attente et des ordonnanceurs de paquets pour la gestion de la Qualité de Service par exemple.
- Statistiques : Le FE doit obtenir du CE certaines statistiques et l'état des interfaces et de chaque élément le constituant.
- Message de *Contrôle* : Le FE doit faire suivre au CE les messages qui lui sont destinés, tels que les messages des protocoles de routage (RIP, OSPF, etc.).

La partie *Contrôle* et la partie *Acheminement* peuvent ne pas utiliser les mêmes mécanismes et protocoles pour échanger leurs informations, tout en sachant que le médium physique les reliant peut être de natures diverses comme un port PCI ou bien un lien Ethernet. Vu l'aspect hétérogène du matériel de ces deux parties, le *'Network Processing Forum'* (NPF) [NPF 2003] a proposé des spécifications d'APIs permettant de définir une interface flexible. Bien qu'il n'existe pas pour le moment de standard pour la post-association, le groupe de travail ForCES a défini les bases d'un protocole pour la gérer, appelé FACT [AUDU 2003].

Avant de se focaliser sur la description du protocole FACT en tant que tel, il nous faut remarquer la place importante donnée aux requis de haute fiabilité inhérente à ce type de protocole. En effet, les interactions entre les CEs et FEs doivent être assurées indépendamment d'une panne d'une des deux parties. Si un CE se trouve surcharger et ne répond plus, il est essentiel qu'un autre CE puisse gérer les FEs qui lui sont assignés pour éviter une interruption de services. De manière non exhaustive, nous pouvons considérer trois grands types de gestion de disponibilité :

- La consistance forte : Plus d'un CE est actif mais seulement un est en communication avec un FE. Les autres CEs écoutent les communications (il faut d'ailleurs prendre en compte le type de liaison qui peut se trouver être un lien partagé ou bien un environnement multi-saut) entre le CE maître et le FE, et mettent à jour leurs états.
- La consistance faible : Le FE communique avec un CE qui lui est assigné, mais peut, en cas de panne de ce dernier, se connecter à un autre CE.
- Le partage de charge : Plusieurs CEs peuvent être configurés pour se partager les fonctionnalités de la partie *Contrôle*. Nous pouvons imaginer qu'un CE prenne en charge le protocole RIP, alors qu'un autre se chargerait de OSPF au sein même d'un groupe de CE.

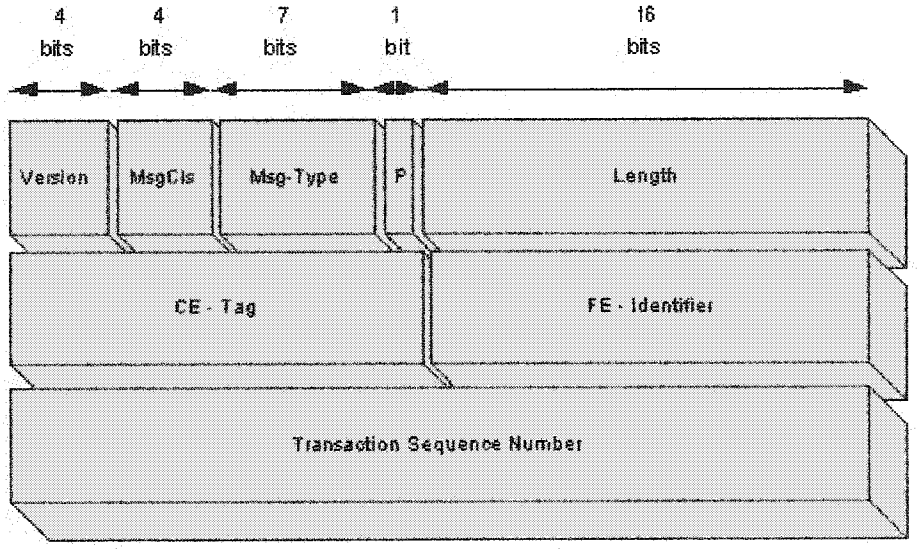


Figure 2.4 Entête des messages FACT

Les différents champs de l'entête des messages FACT, illustrés à la Figure 2.4, indiquent les deux entités CE et FE en communication et décrivent le type de commande encapsulé dans la charge utile :

- Version (4 bits) :

  Ce champ contient la version utilisée du protocole FACT. La version courante est 0x01.

- MsgCls (4 bits) :

  Ce champ indique la classe à laquelle le message appartient. Les valeurs que peut prendre ce champ sont indiquées au Tableau 2.1.

Tableau 2.1 Classe des messages du protocole FACT

Classe	Valeur du Champ
Classe réservée	0
Connexion et Association entre PE (CA Messages)	1
Contrôle de Capacité (CAPCO Messages)	2
Maintenance d'états du PE (PESM Messages)	3
Maintenance du trafic du PE (PETM Messages)	4
Notification d'Evénement (EN Messages)	5
Spécifique aux Vendeurs (VS Messages)	6
Classes réservées par l'IETF	7 à 15

- Msg-Type (7 bits):

Ce champ distingue les différents types de messages au sien d'une classe. Chacun de ces types décrit le champ d'application des messages. Nous avons regroupé au Tableau 2.2 les principaux types de messages sur lesquelles reposent le protocole FACT.

Tableau 2.2 Types des messages du protocole FACT

Classe	Type	Valeur du Champ
Classe CA	Requête d'association (*Join Request*)	0
	Réponse d'association (*Join Response*)	1
	Requête de départ (*Leave Request*)	2
	Réponse de départ (*Leave Response*)	3
Classe CAPCO	Requête de capacité (*Capabilities Request*)	1
	Réponse de capacité (*Capabilities Response*)	2
	Configuration de Composants Logiques (*Configure Logic Components*)	3
	Réponse de Configuration de Composants (*Configure Logic Components Response*)	4
	Requête de commande (*Query Request*)	7
	Réponse de topologie (*Topology Response*)	8
Classe PESM	Message de contrôle de la connexion (*Heartbeat*)	9
	Réponse du contrôle de la connexion (*Heartbeat Ack*)	10

- Length (16 bits) :

Ce champ contient la taille totale du message en octets, entête incluse.

- CE-Tag (16 bits) :

  Ce champ se décompose en deux sous-champs, le CE-Set et le CE-Identifier, tous deux d'une longueur de 8 bits comme illustré à la Figure 2.5.

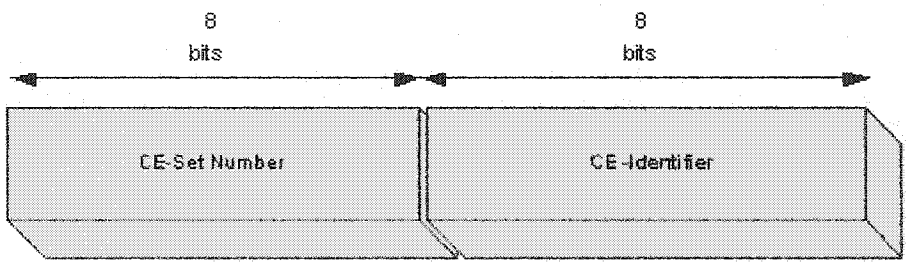


Figure 2.5 Décomposition du CE-Tag

En effet, les CEs peuvent être regroupés en ensemble ayant un identifiant CE-Set qui est unique dans un élément de réseau, et de la même manière le CE-Identifier est l'identifiant du CE en tant que tel.

- FE-Identifier (16 bits) :

  Ce champ correspond à l'identifiant unique du FE.

- Priority (1 bit) :

  Ce bit sert à coder la priorité du message. Si ce bit est à un, il sera traité avec une haute priorité.

- Transaction Sequence Number (32 bits) :

  Ce champ vise à attribuer à chaque transaction un identifiant unique. Si un CE envoie un message pour un changement de configuration, le FE correspondant renverra un message pour accuser l'état de la commande avec le même numéro de transaction.

La charge utile des messages se présente sous une forme *Type-Length-Value* (TLV) illustrée à la Figure 2.6. Les champs décrivant cette charge sont les suivants :

- Parameter Tag (16 bits) : Ce champ indique le type de charge utile encapsulé ;
- Parameter Length (16 bits) : Ce champ contient la taille du paramètre ;
- Parameter Value : Ce champ représente la charge utile des messages FACT relative à la classe et au type spécifiés.

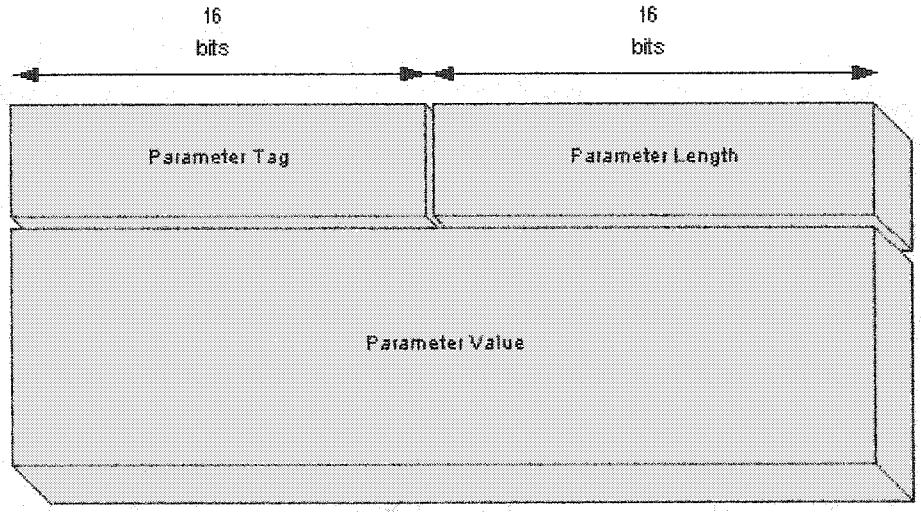


Figure 2.6 Charge utile des messages FACT

2.4.5 Limitations de l'architecture ForCES

Pour résumer, la séparation entre les parties *Acheminement* et *Contrôle*, dans le cadre de l'utilisation des processeurs réseau, peut donc apporter aux éléments de réseau tels que les routeurs une grande flexibilité associée à de bonnes performances. Tout cela permet de faire évoluer ces deux parties à une vitesse différente. Ainsi, nous pouvons mettre en évidence deux problématiques : la robustesse de l'architecture et la modélisation des données utilisées.

Tout d'abord, les requis présentés montrent la nécessité d'apporter une solution robuste qui soit pourvue de mécanismes de survivabilité pour assurer la séparation avec les différents types de consistance et le partage de charge. Le protocole FACT est un outil essentiel de ce point de vue. Pour améliorer la robustesse de cette architecture tout en restant cohérent avec les spécifications existantes, nous proposerons une solution par le biais d'un protocole sous-jacent qui encapsule les messages du protocole FACT.

D'autre part, l'architecture ForCES doit apporter une haute compatibilité entre CEs et FEs. Or, la modélisation de ces éléments ne possède pas de syntaxe définie. De plus, la représentation des données partagées par les CEs et FEs doit être suffisamment flexible pour permettre un usage réaliste dans un environnement physique hétérogène.

2.5 Technologies connexes à la séparation

Nous allons décrire dans cette partie des technologies connexes qui nous servirons de base pour nos propositions d'amélioration de l'architecture ForCES. Dans un premier temps, nous présenterons le langage *eXtensible Markup Language* (XML). Ensuite, nous décrirons le protocole TIPC. Comme tous les nouveaux concepts dans le domaine des réseaux, cette séparation ne peut prétendre à un véritable développement sans s'intéresser à la problématique de la sécurité. Dans cette optique, nous présenterons finalement le protocole IPSec dont nous nous servirons dans notre quatrième chapitre.

2.5.1 Le langage *eXtensible Markup Language*

Le langage XML est un langage proposé par le *World Wide Web Consortium* (W3C) [BRAY 2000] en 1996. Dans les faits, le XML est un sous-ensemble du langage *Standard Generalized Markup Language* (SGML) défini dans le standard ISO 8879. Ce dernier a été créé pour faciliter l'échange de documents structurés dans Internet. Comme son nom l'indique, le XML est un langage à balises mais qui, contrairement au HTML, possède la particularité que ces balises soient extensibles. En effet, l'utilisateur définit lui-même ses balises. Ainsi, pour décrire la syntaxe de ce langage, les documents XML s'accompagnent de fichiers modélisant de manière formelle les éléments XML. Ce type de fichier nommé *Document Type Definition* (DTD) permet donc à un utilisateur de vérifier la structure d'un document XML qui peut soit être valide (conforme à un DTD) ou être bien-formé (s'il ne comporte pas de DTD mais suit les règles de base du XML).

Pour mieux comprendre les grandes lignes du XML, nous pouvons ici reprendre les objectifs de conception tels qu'ils ont été définis par le groupe de travail XML du W3C :

- XML devrait pouvoir être utilisé sans difficulté sur Internet ;
- XML devrait soutenir une grande variété d'applications ;
- XML devra être compatible avec SGML ;
- Il devrait être facile d'écrire des programmes traitant les documents XML ;
- Le nombre d'options dans XML doit être réduit au minimum, idéalement à zéro ;

- Les documents XML devraient être raisonnablement clairs et lisibles par un humain ;
- La conception de XML devrait être préparée rapidement ;
- La conception de XML serait formelle et concise ;
- Il devrait être facile de créer des documents XML ;
- La concision dans le balisage de XML est de peu d'importance.

Comme nous venons de le voir, le XML est un moyen de séparer les données d'un document de leur présentation. Pour faire le lien entre les fichiers XML et la forme sous laquelle les données doivent être utilisées, il existe des entités logicielles appelées *parsers*. Les *parsers* sont de deux types : ceux basés sur une arborescence et ceux basés sur les événements.

Les *parsers* basés sur l'arborescence parcourt l'entièreté d'un document XML et crée une structure de données en mémoire (ou un objet suivant le type de langage de programmation adopté). Ces *parsers* suivent l'interface (API) '*Document Object Model*' (DOM [CHAN 2001a]) définie par le W3C. Par contre, les *parsers* basés sur les événements adoptent une logique différente. Au lieu de créer en mémoire une structure de données, un *parser* événementiel parcourt le document et appelle des fonctions de manière asynchrone à chaque fois qu'il rencontre une balise et consomme donc moins de mémoire. Ce type de *parser* possède l'API SAX pour '*Simple API for XML*'.

2.5.2 Le protocole Telecom Inter Process Communication

Telecom Inter Process Communication [TIPC 2003] est un service de communication rapide, fiable et orienté message. Ce système a été spécialement créé pour les communications à l'intérieur d'une grappe d'ordinateurs. La principale particularité de TIPC repose sur le concept de transparence d'adressage dans une grappe. En fait, il se base sur le concept d'adressage logique et non physique que l'on retrouve avec la pile TCP/IP, et cela en maintenant une table de correspondance d'adresse temps-réel. Dans les faits, TIPC assure un mode de communication sans connexion (comme pour les datagrammes UDP par exemple) mais aussi un mode léger orienté connexion,

particulièrement efficace pour un système où les messages sont courts. Dans ces deux modes de fonctionnement, TIPC fournit une Qualité de Service garantissant qu'il n'y a aucune perte de paquets, pas de duplication, et que l'ordre des séquences des messages ne soit pas interrompu. Une autre des grandes fonctionnalités de TIPC est l'extrême portabilité du système. En effet, il définit ses liens au réseau sous forme générique permettant une utilisation de TIPC avec des technologies telles que Ethernet, ATM ou *InfiniBand* et qui peuvent être utilisées simultanément.

Bien que *Transmission Control Protocol* (TCP) soit le protocole le plus utilisé pour les communications, nous pouvons remarquer qu'il possède deux grands désavantages. Tout d'abord, il y a la problématique de la transparence dans l'adressage. Il est nécessaire de connaître l'adresse IP d'un correspondant pour entamer une communication. Or, dans un environnement dynamique, ce type d'adressage peut être un frein à la fiabilité en cas de panne. Nous pouvons aussi noter qu'il existe des mécanismes d'adressage en complément comme *DNS* ou encore *CORBA Naming Service,* mais ils sont lourds en ressource et en complexité.

Dans le cadre des grappes d'ordinateurs, nous remarquons un autre désavantage, celui de la performance. En effet, TCP n'est pas optimisé pour les messages courts et les mécanismes d'initialisation de communications sont dispendieux (au moins neuf paquets échangés pour établir une transaction TCP).

Comme nous le disions précédemment, TIPC est adapté à l'environnement des grappes. Sa conception a été guidée par les remarques suivantes :

- La plupart des messages dans une grappe ne réalisent qu'un saut pour atteindre leurs destinations ;
- La majorité des messages passe par des connexions au sein d'un sous-réseau d'une grappe ;
- La perte des messages est faible et a fortiori les retransmissions sont rares ;
- Les ressources en bande passante et en mémoire sont conséquentes à l'intérieur d'une grappe, et donc une taille fixe pour les fenêtres glissantes suffit ;

- Tous les paquets sont vérifiés au niveau matériel par les média de communications (checksum), il n'est donc pas nécessaire de la faire au niveau logiciel.

La conception de TIPC introduit les concepts de zone, de sous-réseau et de processeur pour représenter la topologie de manière hiérarchique. Une grappe est décomposée en zones représentant l'ensemble des ressources gérées. Le sous-réseau est un sous-ensemble d'une zone avec la particularité que tous les sous-réseaux doivent être interconnectés les uns aux autres. Ensuite, un processeur est un élément d'un sous-réseau. De la même manière, tous les processeurs d'un sous-réseau nécessitent une interconnexion globale. Nous pouvons noter que l'implémentation actuelle permet 15 zones comprenant chacune 4097 sous-réseaux ayant 511 processeurs. Chaque entité est identifiée en interne par une adresse se présentant sous la forme zone.sous-réseau.processeur. La Figure 2.7 illustre un exemple de topologie de grappe avec TIPC.

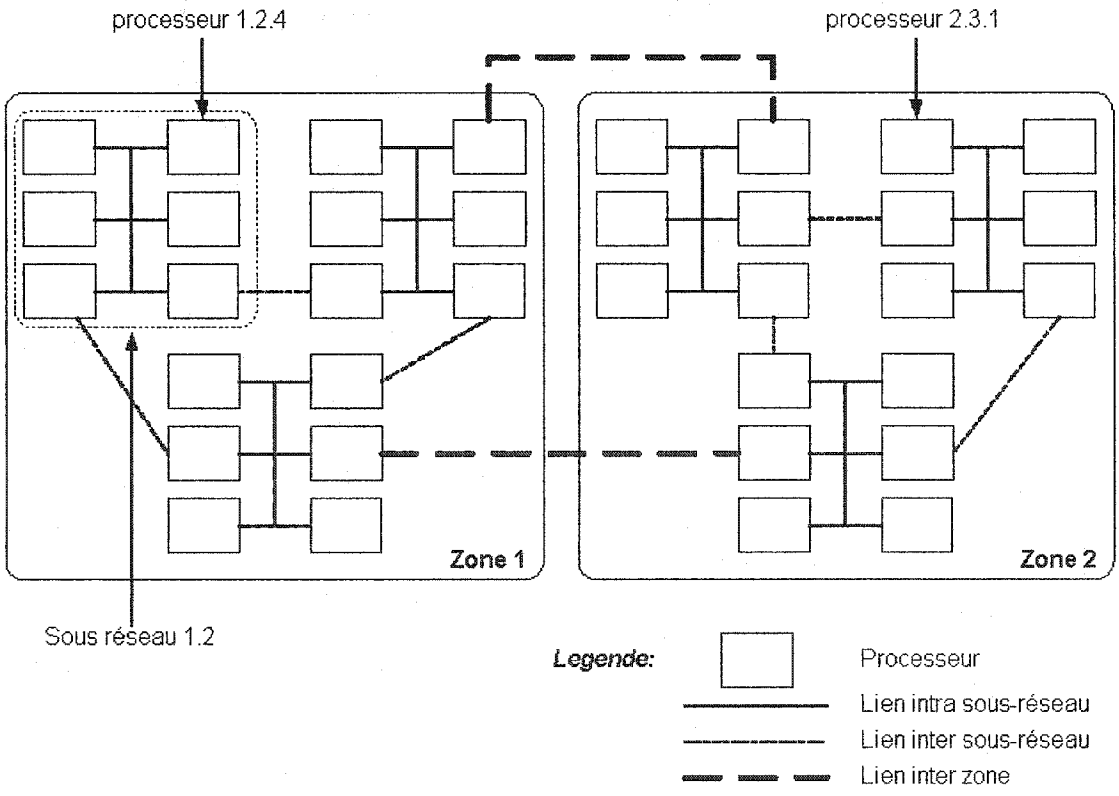


Figure 2.7 Exemple de représentation de grappe sous TIPC

Maintenant que nous avons vu la topologie, nous allons introduire le concept d'adressage logique avec TIPC. Il existe trois types accessibles d'adresse : le nom de Port, l'identifiant de Port et l'identifiant de zone.

Le nom de Port se présente sous la forme de deux entiers de 32 bits. C'est l'adresse persistante pour établir une communication, elle peut être comparée à un port sous TCP. Le premier entier, appelé type, identifie un type de service et le second, qui est l'instance, permet de représenter une instance particulière d'un service. Ce format fournit un moyen de partitionner les services apportés par des serveurs. Dans cette optique, TIPC introduit les séquences de nom de port qui donne un type, ainsi que des bornes inférieure et supérieure d'instance. Les séquences de nom sont utiles pour publier un service sur l'ensemble d'une grappe. La Figure 2.8 illustre ces concepts.

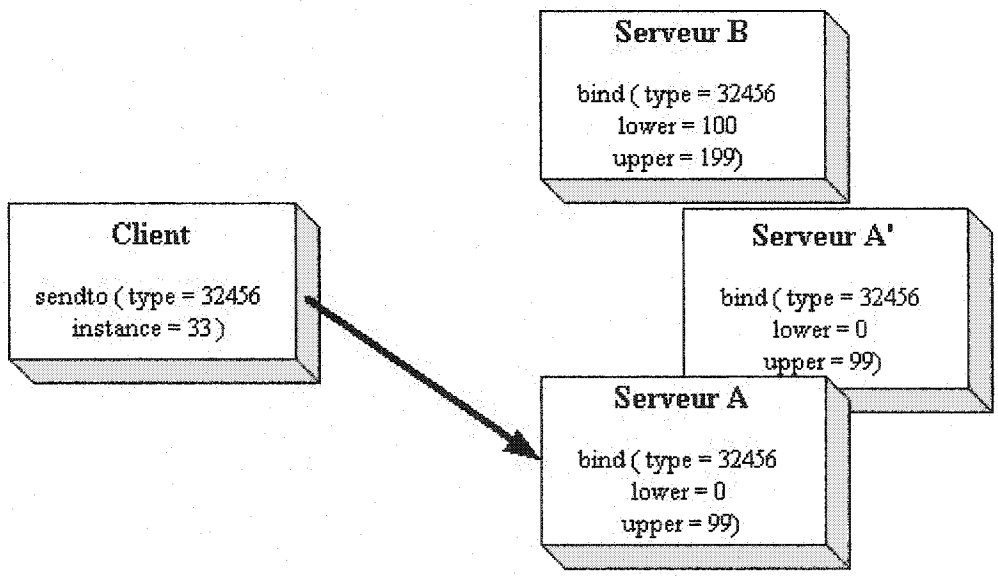


Figure 2.8 Exemple de communication Client/Serveur TIPC

Cet exemple simple d'une communication par le biais de *socket* TIPC permet d'illustrer deux concepts. Le premier est le partitionnement de services avec les serveurs *A* et *B* correspondant au même numéro de type 32456 mais avec des bornes différentes. Le second est la redondance de service. En effet, il est possible de démarrer un second serveur (le serveur *A'*) sur le même numéro de port. Si le Client perd la connexion avec

le serveur *A*, il peut automatiquement se reconnecter avec le serveur *A'* de manière transparente et ce, quelle que soit l'adresse IP de *A'*.

Pour en revenir au type d'adressage, l'identifiant de Port est une adresse volatile utilisée en interne par TIPC quand un port est créé. Elle est composée de deux entiers de 32 bits. Le premier est un nombre pseudo-aléatoire et le second est l'adresse du processeur sous la forme que nous avons vu plus haut.

Le dernier type d'adresse est l'identifiant de zone. En effet, la transparence des adresses est assurée uniquement au sein de la grappe, et dès qu'un message doit aller d'une zone à une autre, l'adresse de cette dernière doit être indiquée. Ainsi, l'identifiant de zone est un simple entier dont la valeur varie entre 1 et 15.

Pour synthétiser cette approche, nous pouvons voir que TIPC permet des communications inter-processus dans une grappe, comme s'ils se trouvaient sur le même ordinateur. Nous pouvons donc parler d'adressage transparent car TIPC n'utilise pas explicitement d'adresse IP, mais plutôt des adresses logiques pointant vers un service et non un processeur particulier.

Pour ce qui est des performances, une étude comparative de TIPC vis-à-vis de TCP/IP sur des plates-formes Linux (noyau 2.4) réalisée par Ericsson [TIPC 2003] montre que, pour des messages d'une taille inférieure à 16000 octets, TIPC donne de meilleurs résultats en temps d'exécution (envoi et accusé de réception) par message que les *sockets* TCP/IP.

2.5.3 Sécurisation du protocole IP par IPSec

Il existe différents mécanismes pour sécuriser les communications réseau au niveau de la couche applicative, transport, réseau ou encore physique. Nous avons choisi de nous intéresser à la sécurisation au niveau de la couche réseau ayant la particularité d'être transparent et indépendant des autres couches. L'ensemble de ces mécanismes appelés *IP Security Protocols* [KENT 1998a] noté IPSec œuvre au niveau d'IP. Cette technologie est obligatoire avec IPv6 mais est optionnelle pour IPv4. De manière générale, IPSec apporte deux grands services, le premier étant la confidentialité des

données (par chiffrage) et le second étant l'authenticité des données. Ces fonctionnalités sont apportées par deux protocoles AH et ESP.

Le protocole AH (*'Authentication Header'*) [KENT 1998b] vise à assurer l'authentification et l'intégrité des données. Nous pouvons noter que AH n'utilise pas de fonction de chiffrement afin d'être utilisable partout sans restriction légale. Pour ce faire, AH ajoute aux paquets IP une entête contenant les informations nécessaires à la vérification de l'authenticité des données à la réception. La Figure 2.9 illustre les champs de AH.

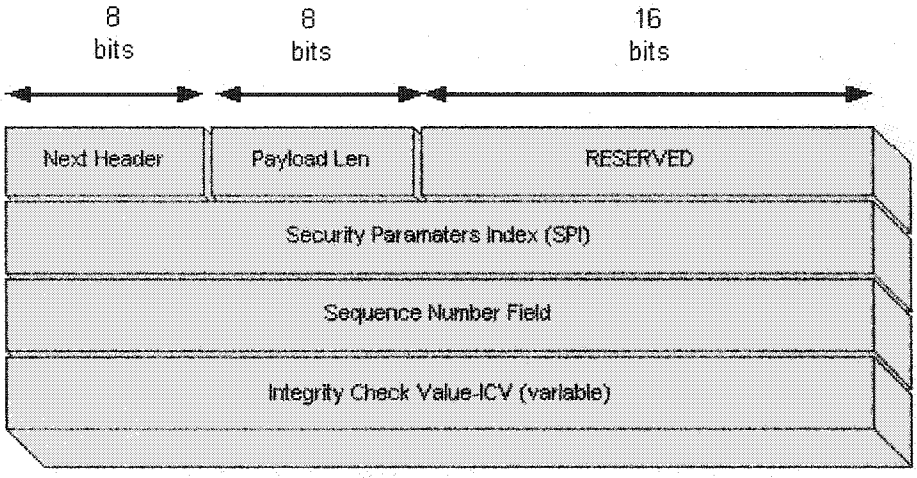


Figure 2.9 Format des entêtes AH

Le protocole *Encapsulating Security Payload* (ESP) [KENT 1998c] permet d'assurer la confidentialité des données mais aussi leur authenticité. Dans le cas de AH, le protocole utilise simplement une entête supplémentaire, alors qu'ici avec ESP, les données du datagramme sont encapsulés. Le format d'entête du protocole est illustré à la Figure 2.10.

Nous pouvons d'autre part remarquer que les protocoles AH et ESP n'imposent aucun algorithme cryptographique en particulier. AH peut user de MD5 ou encore SHA-1, et ESP peut se servir d'algorithme comme Triple DES. IPSec peut fonctionner soit en *mode Transport*, soit en *mode Tunnel*. Dans le mode Transport, seul le contenu des paquets IP est protégé. Ce mode s'utilise en général pour les communications entre deux hôtes. En effet, les données issues de la couche 4 (TCP ou bien UDP), sont d'abord

protégées par IPSec avant d'être transmises à la couche IP. Ce mode de communication possède le désavantage d'user de la couche IP après IPSec, et donc les adresses IP ne sont pas masquées, mais a l'avantage d'être facile à mettre en œuvre. Pour ce qui est du mode Tunnel, il s'agit en fait d'une encapsulation du paquet IP d'origine dans un nouveau paquet pour pallier les masquages d'adresses du *mode Transport*.

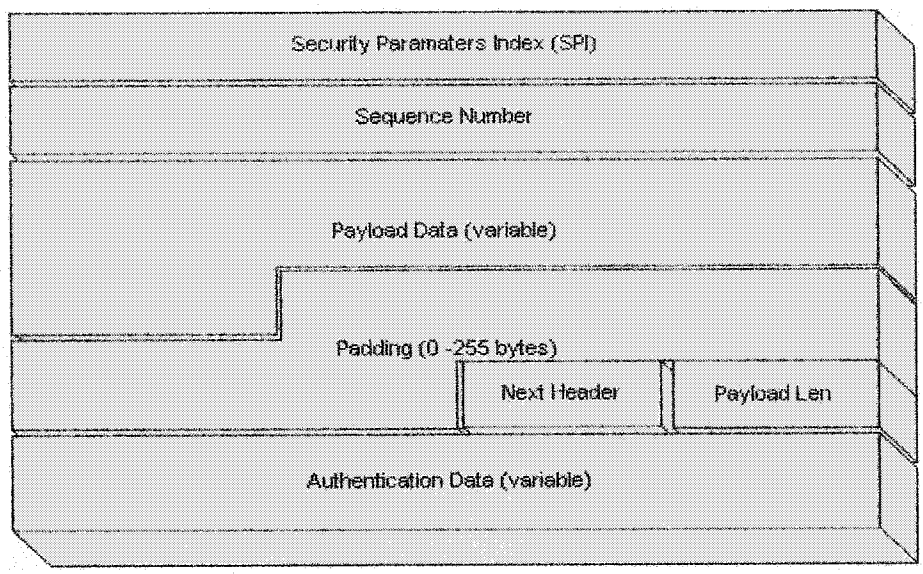


Figure 2.10 Format des entêtes ESP

IPSec se base donc sur un ensemble de fonctions cryptographiques possédant leurs propres paramètres telles que les clefs publiques/privées. En fait, soit l'administrateur prend le temps de paramétrer le système par lui-même, soit IPSec peut se baser sur un protocole de gestion dynamique des paramètres, tel que IKE ('Internet Key Exchange'), pour l'échange des clefs.

CHAPITRE III

MODÉLISATION DE LA PARTIE ACHEMINEMENT ET PROTOCOLE FACT

L'ensemble des requis relatifs à la séparation des parties *Contrôle* et *Acheminement* permet d'établir une architecture fonctionnelle répondant au souci de gestion de l'hétérogénéité du matériel et de la capacité de faire évoluer ces deux composantes à des rythmes différents. Dans ce chapitre, nous étudierons en détail deux problématiques : celle de la robustesse de cette architecture et celle de la modélisation de la structure des FEs, des fonctionnalités des CEs, ainsi que de leurs interactions. En effet, l'objectif de notre approche est d'établir une critique constructive des propositions du groupe ForCES de l'IETF afin de proposer une architecture évolutive et automatisée permettant l'émergence de ce concept.

3.1 Principes de la modification du protocole ForCES

Le point de départ de la séparation entre la partie *Contrôle* et la partie *Acheminement*, se trouve être l'idée de diviser la pile des protocoles en ces deux parties. Ainsi, chacune des deux pourra être indépendante des évolutions de l'autre tout en optimisant leurs performances à un coût restreint. L'architecture proposée par le groupe de l'IETF s'intéresse, comme nous l'avons vu dans le précédent chapitre, aux points importants de ce concept : la phase de pré-association gérée par les CE et FE Managers, la gestion des FEs par les CEs par le biais du protocole FACT, ainsi que les mécanismes de haute disponibilité. Étant donné que cette architecture en est à ses débuts, certains points d'ombre subsistent, notamment sur la place du protocole FACT dans

l'architecture et, d'autre part, sur le fait qu'aucun mécanisme n'est spécifié pour décrire la phase de pré-association. Nous pouvons noter de même que la modélisation des FEs est laissée entièrement libre, pourtant ses interactions avec le protocole FACT sont primordiales. De même, la modélisation des CEs et des FEs constitue aussi un domaine d'étude à approfondir pour permettre un véritable essor des technologies des processeurs réseau. En effet, l'interopérabilité est un des grands critères qui conditionnent le succès d'une spécification dans le monde très concurrentiel des réseaux. Ce constat nous amène à essayer d'apporter un point de vue global et synthétique sur les différentes propositions de l'IETF.

Dans ce mémoire, nous allons nous focaliser sur deux grandes problématiques : celle de la robustesse de l'architecture ForCES et a fortiori de FACT, et aussi celle de la modélisation des fonctionnalités et des données des parties *Contrôle* et *Acheminement.*

Notre approche pour la problématique de la robustesse passera par l'usage d'un protocole sous-jacent à l'architecture ForCES. Pour ce qui est de la problématique de la modélisation, nous avons décidé de scinder notre approche en deux points, l'un lié à la phase de pré-association et l'autre à la phase de post-association. En effet, la modélisation des données de ces deux phases possède chacune leurs spécificités. Pour la phase de pré-association, nous devons modéliser les entités logiques pour pouvoir les faire interagir de manière optimale. D'autre part, pour la post-association, il est nécessaire de modéliser les structures de données échangées par les CEs et les FEs. Mais en fait, ces deux types de modélisation visent à une meilleure flexibilité de l'architecture.

Nous avons choisi le langage XML présenté dans notre second chapitre, comme support commun pour ces modélisations. En effet, ce langage s'avère être un outil formel flexible pour la modélisation des différents types de données échangés. Il est important de voir qu'au-delà d'un modèle théorique, la séparation de la partie *Contrôle* et *Acheminement* se traduit par des éléments matériels qui sont très hétérogènes. Il nous faut donc pourvoir être capable d'implémenter un routeur à partir de composants de différents constructeurs (que ce soit de processeurs réseau, ou bien de fonctions de la

partie *Contrôle*). Or, XML par sa syntaxe évolutive nous apporte une solution crédible à la problématique de la modélisation, et ipso facto, à celle de l'interopérabilité. Mais, il faut toutefois être conscient que la nécessité d'utiliser un *parser* peut altérer les performances du système, comparées à celles d'une modélisation sous forme de structures de données binaires. Le traitement de la problématique de la flexibilité nous amène donc à faire des concessions sur les performances sur lesquelles nous reviendrons dans le quatrième chapitre.

3.2 Amélioration de la robustesse du protocole

Lors de notre présentation des requis de l'architecture ForCES, nous avons vu que ceux relatifs à la robustesse étaient intimement liés à la gestion de la redondance des CEs et de la gestion dynamique des PEs. D'autre part, nous avons aussi introduit le protocole TIPC.

3.2.1 Hypothèse initiale et choix de TIPC

Comme nous l'avons vu dans le chapitre précédent, l'avènement d'une technologie comme les processeurs réseau, permet de concrétiser la séparation entre la partie *Contrôle* et la partie *Acheminement.* Dans les faits, les FEs sont une représentation logique des fonctionnalités de ces processeurs. Or, les routeurs basés sur cette technologie sont en fait des ordinateurs traditionnels connectés via un *Backplane*, ou bien un réseau local à des cartes munies de ces processeurs. Cet aspect nous permet de nous placer dans le cadre d'une grappe d'ordinateurs pour l'interconnexion entre CEs et FEs.

En adoptant cette hypothèse, nous pouvons remarquer la corrélation entre les requis de robustesse de l'architecture ForCES et l'usage de TIPC comme protocole de communication. Le support de différentes technologies de communication (UDP, ATM, Ethernet, etc.) et surtout l'adressage logique au sein d'une grappe font du protocole TIPC une solution intéressante pour la robustesse de l'architecture. Dans ce mémoire, nous utiliserons ce protocole de communication. Ce choix nous amène à modifier le protocole FACT.

3.2.2 Impact sur le protocole FACT

D'un point de vue conceptuel, nous utilisons TIPC comme protocole sous-jacent au protocole FACT. Ainsi, chaque CE se comporte donc comme un serveur TIPC dont l'adressage sous forme de nom de port est souscrit sur l'ensemble de la grappe et chaque FE se comportera symétriquement comme un client.

L'adressage sous forme de nom de port nous assure une gestion très simplifiée de la redondance du point de vue du FE. On associe chaque CE à un type donné au sens d'un nom de port, ainsi qu'une borne inférieure et une supérieure fixes pour tous les noms de port publiés au sein de la grappe. Ce choix est motivé par la transparence d'adressage qui en découle. Pour cela, prenons l'exemple simple d'un CE ayant un nom de port dont le type est 17777 et que nous ayons un FE qui devient actif et qui établit une connexion TIPC avec ce type. Ensuite, nous démarrons un autre CE avec le même nom de port. Si le premier CE connaît une panne quelconque, le FE par le biais du protocole TIPC remarquera la perte de connexion (en usant de TIPC avec Ethernet par exemple, les pertes sont détectées en 1,5 secondes) et il pourra tenter de se reconnecter au même type 17777. Dans ce cas, il se verra automatiquement en communication avec le second CE.

L'usage de TIPC comme médium permet donc une certaine simplification à l'architecture car la gestion de la redondance et de la détection de perte de connexion se trouve donc sous-jacente à l'architecture ForCES. D'autant plus que TIPC est un protocole robuste, largement utilisé dans l'industrie depuis une dizaine d'années.

En effet, le protocole FACT assure la gestion des pertes de communications de manière explicite par le biais de minuteries. Comme nous venons de le voir, ce type de mécanisme n'est plus nécessaire en utilisant TIPC. Les messages de type *Heartbeat* de FACT ne seront plus utilisés et n'auront pas à être implémentés.

3.2.3 Impact sur la phase de pré-association

Nous venons de voir qu'elle est l'apport de TIPC vis-à-vis du protocole FACT et de la phase de post-association. Dans la même optique d'amélioration de la robustesse

de l'architecture ForCES nous avons aussi décidé d'utiliser TIPC pour la phase de pré-association. Ce choix a été motivé par deux points : l'adressage même des Managers et la configuration des adresses des PEs.

Pour gérer l'adressage des Managers, nous assignons aux CE Managers et aux FE Managers deux adresses génériques connues de tous les PEs. Cela permet aux PEs qui deviennent actifs de pouvoir directement se connecter à leur Manager respectif. En fait, les Managers seront simplement des serveurs TIPC et tous les PEs des clients ce qui simplifie la phase de pré-association et leur configuration. D'ailleurs, ce choix nous amène à utiliser le même procédé de redondance vu pour les CEs pour les Managers afin d'assurer la robustesse de la pré-association. Pour illustrer notre point de vue, il suffit d'imaginer qu'un premier CE Manager devienne actif et donc publie le nom de port générique des CE Managers. Ensuite, un second complète la même étape. Le premier aura la responsabilité de gérer toutes les pré-associations des PEs. Mais si ce dernier connaît une panne, le second qui a publié le même nom de port, pourra ainsi assurer le rôle de Manager de manière transparente et automatique pour tous les PEs déjà en connexion et ceux à venir.

Bien que TIPC simplifie l'adressage, il subsiste une interrogation sur la manière de configurer les appariements CE – FE du protocole FACT. Tel que vu par le groupe de travail ForCES, les configurations devraient être manuelles avant l'entrée en activité d'un PE. De plus, pour reprendre les requis de cette architecture, un élément de réseau doit pouvoir gérer des centaines de FEs et donc leur configuration deviendrait vite problématique. Or, TIPC nous apporte une solution à la problématique de la configuration des éléments de réseau. En effet, nous allons affecter aux CE Managers une nouvelle responsabilité, celle de l'affectation des noms de ports à tous les CEs. Lors de la phase de pré-association, les CE Managers auront donc à donner à tout nouveau CE un nom de port pour assurer une cohérence maximale à nos choix pour la phase de post-association.

3.3 Modélisation de la phase de pré-association

Le traitement de la problématique de la modélisation de la phase de pré-association passe avant tous par une étude explicite des besoins de cette phase. Pour cela, il nous faut revenir sur la fonction première de la pré-association : l'appariement des CEs et des FEs. En effet, la mise en relation CE – FE nécessite que ces Managers connaissent leurs structures fonctionnelles. Dans cette section, nous verrons notre proposition pour formaliser les liens CE – CE Manager, FE – FE Manager et CE Manager – FE Manager par le biais d'une représentation en XML des particularités de chacun.

3.3.1 Interactions CE – CE Manager

Un CE Manager se trouve être, au sens du document ietf-forces-requirement [KHOS 2003], responsable de déterminer quelles FEs sont susceptible d'être contrôlés par un certain CE.

Pour mieux comprendre l'importance de cette entité, nous pouvons prendre l'exemple suivant où un CE quelconque devient opérationnel au sein d'un élément de réseau. En effet, ce dernier possède un certain nombre de fonctionnalités pour lesquelles il est compétent, comme par exemple le support du conditionnement *DiffServ* dans un routeur à la frontière d'un domaine ou d'une région *DiffServ*. Pour être entièrement opérationnel, il se doit dans un premier temps de s'enregistrer au sein de l'élément de réseau. Mais, aussi d'être mis en contact avec des FEs susceptibles de lui être complémentaires. Pour cela, son interaction avec le CE Manager suit le diagramme temporel de la Figure 3.1 repris du document ietf-forces-framework [YANG 2003].

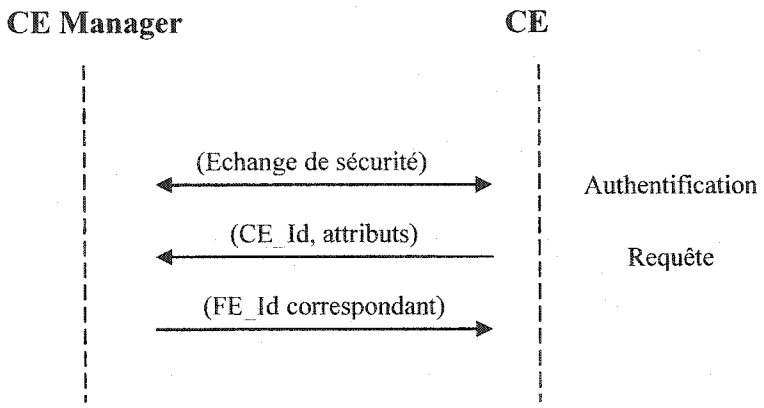


Figure 3.1 Interaction CE – CE Manager

Selon cette figure, l'état des spécifications se trouve être incomplet pour le moment, nous allons essayer de compléter cette approche, en spécifiant de manière plus précise chacun de ces échanges étape par étape :

1. Dans un premier temps, le CE prévient le CE Manager de son existence. Le CE Manager l'authentifie et accepte ou non de traiter sa demande.
2. Ensuite, le CE Manager lui fournit un identifiant, le CE-Id. En effet, nous pouvons noter que son attribution n'est pas explicitée dans le cadre de l'architecture proposée par le groupe ForCES.
3. Ensuite, le CE aura a déclaré ses besoins et ses restrictions (de capacité de traitement par exemple) pour voir si ces services sont disponibles parmi les FEs appartenant à l'élément de réseau et recevra ensuite l'aval du CE Manager. On notera surtout dans ce point que la représentation de ses besoins est, en fait, une problématique en soit. Pour apporter une solution à ce problème, nous nous sommes basés sur les requis présentés par le document ietf-forces-requirement [KHOS 2003]. Les types de fonctions logiques implémentés au sein d'un FE nous guident vers une déclaration simple des besoins des CEs sous cette même forme. Pour cela, nous avons défini un DTD combinant les besoins et ceci, dans un sens large (c'est au CE de décider s'il accepte de travailler avec le FE proposé

afin de ne pas être redondant vis-à-vis de la phase de post-association), mais aussi ses limitations en nombre de FEs que le CE peut gérer. Ce modèle sera dénommé *CE Specification.*

Dans ce DTD se trouvant en Annexe A, nous avons discerné six grands types de besoins fonctionnels pouvant être exprimés par un CE : le type de port devant être présents par exemple une interface Ethernet, le type d'acheminement réseau pouvant être IP version 4 ou 6, le type de Qualité de Service si le CE œuvre suivant une architecture *IntServ* ou encore *DiffServ*, le type d'encapsulation, le type de sécurité (dans un routeur d'accès, les fonctions IPSec se trouvent être nécessaires) et enfin le type de fonctionnalités propres aux constructeurs (seulement dans le cas où les FEs les reconnaissent).

Pour illustrer cette approche, nous pouvons revenir sur l'exemple proposé au début de cette section. Pour formaliser les besoins de ce CE, nous utilisons le document XML représenté à la Figure 3.2.

```
<CESpecification>
    <maxFE>3</maxFE>
    <EncapType>XML</ EncapType >
    <FailOver>Enable</ FailOver >
    <FunctionList size="2">
        <Function>
            <Port>
                <PortNb>2</PortNb>
                <PortTypeList size="1">
                        <PortType>Ethernet</PortType>
                </PortTypeList>
            </Port>
        </Function>
        <Function>
            <QoS>
                <QoSTypeList size="1">
                    <QoSType>DiffServ</QoSType>
                </QoSTypeList>
            </QoS>
        </Function>
    </FunctionList>
</CESpecification>
```

Figure 3.2 Exemple de *CE Specification*

Nous pouvons noter que ce CE ne peut traiter que trois FEs simultanément, qu'il nécessite deux interfaces Ethernet et que, d'autre part, il utilise l'architecture *DiffServ* à la frontière d'un domaine. Cela permet au CE Manager de le mettre en relation avec les FEs pouvant remplir ces requis.

4. Nous rentrons ensuite dans une phase événementielle. En effet, une fois les trois étapes précédentes complétées, le CE Manager se doit de savoir s'il le CE concerné se trouve dans un état actif, inactif ou bien en attente. Une telle phase passe donc par des messages qualifiables d'événements, afin que le CE Manager soit mis à jour. Nous nous trouvons face à deux cas de figure. Le premier correspond simplement à une inactivation d'un CE pour une raison quelconque mais due à un fonctionnement normal. Dans ce cas, l'envoi d'un message asynchrone est suffisant pour assurer le suivi du CE Manager. Le second cas de figure réfère à une panne d'un CE ou bien d'une perte de connexion. La gestion de ce genre d'événement peut passer naturellement par l'usage de messages périodiques à une fréquence définie, mais imposera au système une latence. Nous proposerons comme nous l'avons vu dans la section traitant de la robustesse du protocole ForCES, d'user du protocole TIPC pour détecter ce genre d'événement.

Cette décomposition du comportement potentiel des relations CE – CE Manager nous servira de base dans la suite de notre modélisation de la phase de pré-association.

3.3.2 Interactions FE – FE Manager

Nous pouvons maintenant nous intéresser aux interactions entre un FE et un FE Manager. Pour cela, nous allons dans un premier temps présenter sous forme de diagramme les propositions du groupe de travail ForCES. Une fois encore, il faut noter à la Figure 3.3 tirée du document ietf-forces-framework, un point de départ pour une amélioration de ces processus de découverte et d'initialisation.

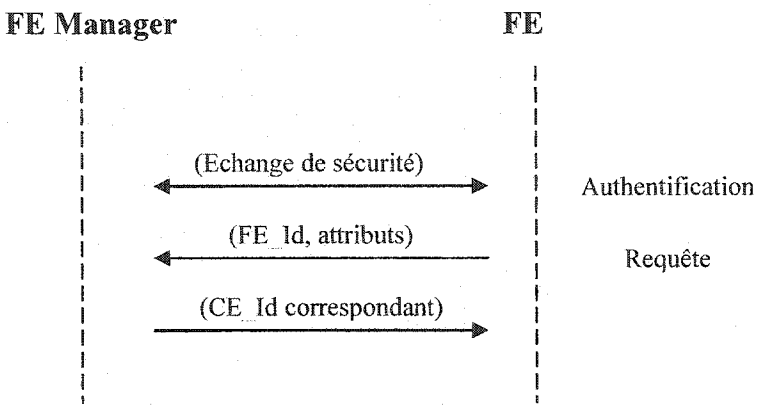


Figure 3.3 Interaction FE – FE Manager

La similitude avec les communications CE – CE Manager nous permet aisément d'adopter la même approche afin de compléter cette proposition. Nous décomposerons donc ces interactions suivant les mêmes grandes étapes similaires à celle des CEs. Pourtant, la problématique de la modélisation se trouve être plus complexe que pour les CEs. En effet, nous avons vu dans le second chapitre que le groupe ForCES à énumérer un certain nombre de spécifications relativement à ce qu'il dénomme le *FE Model*. Les FEs étant une représentation logique d'un élément physique, ils sont donc soumis à la contrainte de l'hétérogénéité et surtout de la flexibilité. Il est évident que la vitesse d'évolution du matériel de la partie *Acheminement* est bien plus rapide (la loi de Moore étant encore vérifiée) que celle des protocoles représentés dans la partie *Contrôle* nécessitant l'aval des consortiums de normalisation.

Nous avons pu voir que la modélisation des FEs passe par une dissociation entre le modèle des capacités des FEs sous la forme de blocs logiques, et le modèle d'état représentant l'état au niveau des connexions de ces blocs. Pour garantir la flexibilité de notre proposition de modèle de FEs, nous nous orientons vers une formalisation en XML. Dans cette optique, nous avons établi un DTD permettant de modéliser les FEs contenant des fonctionnalités de Port, de routage IP, mais aussi correspondant aux blocs logiques de l'architecture de Qualité de Service *DiffServ*. L'ensemble de cette spécification est disponible en Annexe A.

Nous retrouvons au sein de ce DTD les requis avancés par le groupe ForCES pour ce qui est des différents types de fonctions logiques (balise « Type »). Un point pourtant reste à développer, il s'agit de la représentation formelle du type de statistique que peut fournir un FE (la balise « StatType » étant optionnelle). En effet, l'état des implémentations des FEs sur les processeurs n'est pas encore suffisamment avancé pour synthétiser toutes les formes de statistiques qui pourront être fournies.

Pour illustrer ce DTD relativement abstraite, nous allons revenir sur l'exemple présenté au chapitre 2 (Figure 2.3). Il s'agit ici d'un nœud *DiffServ* à la frontière d'un domaine dont nous avons représenté la modélisation complète en Annexe C. D'une manière générale, nous représentons dans deux balises les modèles de capacités et d'états, comme nous pouvons voir dans l'exemple illustré par la Figure 3.4 tiré de l'Annexe B.

```
<FEModel>
    <CapabilityModel>
        <LBList size="9" >
            <LB>...</LB>
            </LBList>
    </CapabilityModel>
    <StateModel>
        <LinkList size="9">
            <Link>...</Link>
        </LinkList>
    </StateModel>
</FEModel>
```

Figure 3.4 Exemple de *FE Model*

Pour ce qui est des blocs logiques en eux-mêmes, nous y retrouvons un identifiant (balise « Handle ») et un type de fonctionnalité (balise « Type ») qui dans une autre sous-section de l'exemple de l'Annexe B, est un *Meter* DiffServ représenté à la Figure 3.5. Dans cet exemple, nous décrivons les particularités de ce *Meter* qui est de type *Token Bucket* [BAKE 2002].

```
<LB>
    <Handle>4</Handle>
    <Type>
        <DS_Meter>
            <MeterType>
                <Dir>In</Dir>
                <MeterCap>simpleTB</MeterCap>
                <HandleFail>5</HandleFail>
            </MeterType>
        </DS_Meter>
    </Type>
</LB>
```

Figure 3.5 Exemple de balise LB

Pour ce qui est de la représentation du modèle d'état, nous avons opté pour une liste de balises qui contiennent les différents identifiants des blocs logiques reliés en amont et en aval les uns des autres. La Figure 3.6 nous montre un exemple simple, mais les balises Link peuvent contenir plus d'une balise *DownLBhandle* ou *UpLBHandle*, comme pour un *Scheduler* DiffServ qui peut avoir plus d'une queue en amont.

```
<Link>
    <DownLBHandle>3</DownLBHandle>
    <UpLBHandle>8</UpLBHandle>
</Link>
```

Figure 3.6 Exemple de balise Link

3.3.3 Interactions CE Manager – FE Manager

Pour ce type d'interaction, notre approche sera semblable aux échanges précédents. Dans un premier temps, nous allons illustrer les propositions du groupe de travail ForCES par la Figure 3.7.

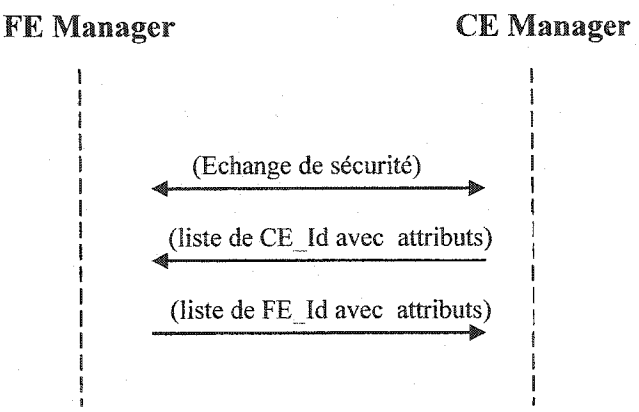


Figure 3.7 Interaction CE Manager – FE Manager

Pour compléter notre approche entre les Managers et les entités logiques, nous allons user de l'approche de ce schéma. Pour ce qui est de l'échange des listes d'identifiants et des attributs, nous avançons le DTD suivant qui complète les deux DTDs présentés dans les deux points précédents :

```
<!ELEMENT FEModelList (FEModel+)>
<!ATTLIST FEModelList size CDATA #REQUIRED>

<!ELEMENT CESpecList (CESpec+)>
<!ATTLIST CESpecList size CDATA #REQUIRED>
```

Figure 3.8 Exemple de balise Link

Elles possèdent néanmoins la particularité de devoir indiquer dans le *CESPec* et le *FEModel* les balises CEId et FEId qui sont optionnelles dans les DTD en Annexe.

3.3.4 Synthèse des échanges pré-association

En considérant nos propositions d'évolution vis-à-vis des interactions FE - FE Manager et CE – CE Manager, il est important de noter une grande similitude dans leurs relations respectives. Les diagrammes temporels et les types d'information échangés nous amènent à proposer une approche basée sur la même syntaxe de message.

Pour offrir, un protocole de communication qui soit le mieux adapté possible, il faut d'abord synthétiser le genre d'informations utilisé. Tout d'abord, dans les deux

diagrammes proposés, chaque échange nécessite un identifiant pour les CEs et les FEs. Ensuite, il est important d'indiquer quel type d'élément se trouve être en communication (CE ou FE). Il est aussi important d'apporter un moyen d'adresser, de manière absolue, le CE Manager et le FE Manager. D'autre part, il faut que le protocole puisse fournir les informations événementielles sur l'état d'activité de l'élément et ce, même dans le cas d'une perte d'un CE ou d'un FE à cause d'une panne. Le protocole se devrait de pouvoir apporter une description des données de modélisation des FEs, et des besoins des CEs. En effet, le choix de XML que nous proposons se doit d'être en accord avec l'idée de l'architecture du groupe ForCES qui stipule que l'on doit se permettre une certaine latitude dans le choix de modélisation en XML ou autre. D'autre part, il faut rappeler que l'objectif de ce mémoire est d'apporter une base architecturale fonctionnelle et homogène aux interactions des entités logiques. Pour cela, le choix de TIPC comme médium de communication semble être un choix justifié, d'autant plus que ses particularités d'adressage logique vont nous permettre une grande latitude d'action.

Dans un souci de synthèse, nous allons ici décrire les différents cas d'utilisation de ce protocole commun à toutes les entités concernées par la phase de pré-association. En premier lieu, la Figure 3.9 présente les interactions CE – CE Manager et FE – FE Manager sous la forme de messages échangés. Ici, les interactions sont décomposées en trois phases temporelles représentant la phase de pré-association, la notification d'événements et la fin des échanges. Au-dessus des flèches représentant les messages, nous indiquons le type de messages.

La première phase se décompose en deux étapes. Tout d'abord, nous avons un échange de messages d'initialisation pour l'authentification et le fait que le Manager peut accepter ou rejeter la demande de l'entité pour faire partie de l'élément de réseau qu'il représente.

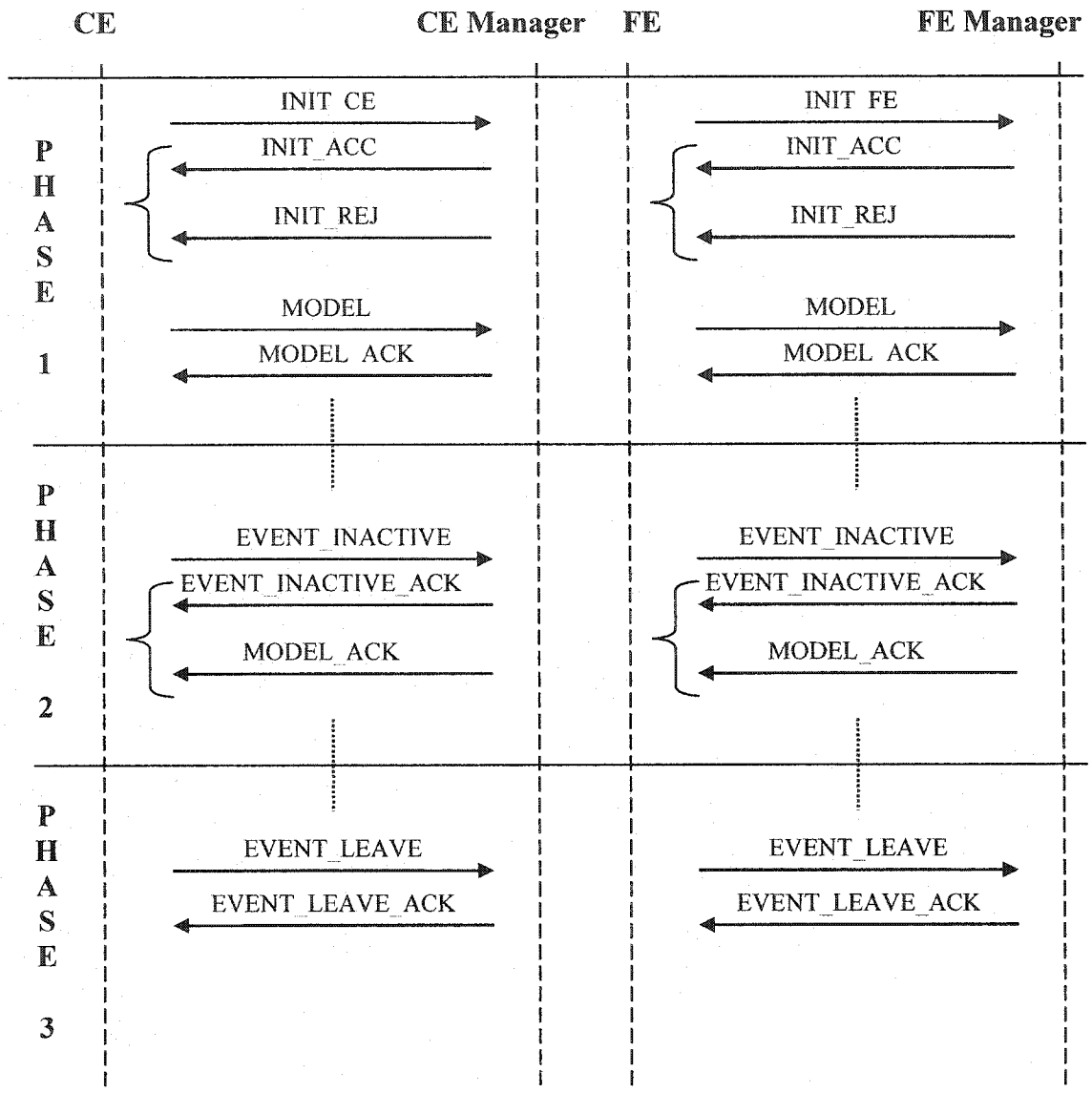


Figure 3.9 Interactions Entités –Managers complétées

D'autre part, nous pouvons imaginer le cas de figure où un CE soit déjà rentré en communication avec un CE Manager mais que ce dernier soit tombé en panne après la phase de pré-association. Dans ce cas, le CE possède déjà un identifiant et une adresse publiée et donc, s'il se reconnecte à un autre Manager, il doit lui envoyer son identifiant et son adresse. Ainsi, il parait donc nécessaire que les CEs ou les FEs encapsulent dans leurs messages INIT_CE et INIT_FE leurs identifiants ainsi que leurs adresses. D'autre

part, afin d'offrir un large choix d'authentification, nous n'imposons pas de manière standard de présenter ces données. De plus, dans le cadre de ce mémoire, nous nous plaçons dans le cadre d'une grappe d'ordinateurs qui est un système fermé du point de vue des communications CE/FE avec les Managers que nous pouvons supposer de confiance.

Pour ce qui est des messages INIT_ACC et INIT_REJ, ils doivent aussi encapsuler, pour le INIT_ACC, l'identifiant de l'élément de réseau auquel appartienne le CE ou le FE, et pour le INIT_REJ, la raison du refus. Alors, nous nous retrouvons dans la seconde étape de cette phase qui correspond à l'envoi des modèles. Dans le cas du CE, ce dernier aura à envoyer un message de type MODEL encapsulant sa modélisation XML. Le CE Manager lui renverra ensuite un MODEL_ACK contenant, d'une part, une adresse TIPC pour qu'il puisse démarrer son serveur répondant aux demandes de connexions FACT et, d'autre part, une liste de FEs qui potentiellement peuvent lui être assigner. Pour ce qui est du FE, le schéma suit la même logique mais le message de type MODEL encapsule son FE Model. À la fin de cette phase, les Managers respectifs considèrent leurs entités comme actives ou en voie de l'être.

Nous arrivons maintenant au cœur de la phase 2 qui représente la gestion des événements relatifs à l'état des CEs et FEs. Il est possible qu'un FE n'ait pas réussi à se connecter à un CE, et demande donc une autre adresse à laquelle se connecter ou encore qu'un CE se trouve à attendre des connexions qui n'arrivent pas. Dans ces différents cas, il est important que les Managers respectifs soient au courant de leurs états, d'où ces messages événementiels. Ce type de message EVENT_INACTIVE doit d'ailleurs encapsulé la raison de cette inactivité. Les réponses à ces messages peuvent être de deux types : soit le Manager indique à son entité logique qu'elle peut rester en attente (et après un laps de temps donné réessaye de communiquer avec le Manager) c'est le type EVENT_INACTIVE_ACK, soit le Manager renvoie un message de type MODEL_ACK vu dans la seconde étape de la phase 1 mais avec d'autres propositions d'affectation encapsulées.

Finalement, la troisième phase correspond simplement à la notification auprès d'un Manager du fait qu'un FE ou un CE a décidé de quitter l'élément de réseau, ce sont les messages de type EVENT_LEAVE (qui encapsulent la raison de cette demande). Et les messages de types EVENT_LEAVE_ACK sont un simple accusé de réception de cette demande. Là encore, il aurait été envisageable d'utiliser des messages périodiques pour vérifier l'activité d'un élément, mais dans notre cas, avec l'usage de TIPC, ce type de mécanisme devient inutile.

Maintenant, nous allons continuer notre synthèse en nous focalisant cette fois sur les interactions entre les Managers, illustrées à la Figure 3.10. Nous y représentons les échanges sous la forme de trois phases. Les types de messages relatifs aux phases 1 et 3 se trouvent être dans la même approche que pour les interactions CE – CE Manager (ou FE – FE Manager), à la seule différence que les flèches ici sont bidirectionnelles. En effet, le message de type INIT_MANAGER peut être initialisé soit par le FE Manager, soit par le CE Manager. Si l'un des deux commence à initier la communication, cela signifie qu'il est prêt à accepter une collaboration avec le FE Manager correspondant. Ainsi, l'autre Manager indique s'il accepte cette collaboration. Pour ce qui est de la phase 2, nous retrouvons l'échange des modélisations des FEs et CEs des Managers afin de compléter la phase 1 des relations CE – CE Manager et FE – FE Manager. Ce sont les messages de type LIST (et LIST_ACK qui est un accusé de réception du type précédent) qui encapsule une liste de modélisation de CEs ou de FEs ainsi que leurs adresses respectives.

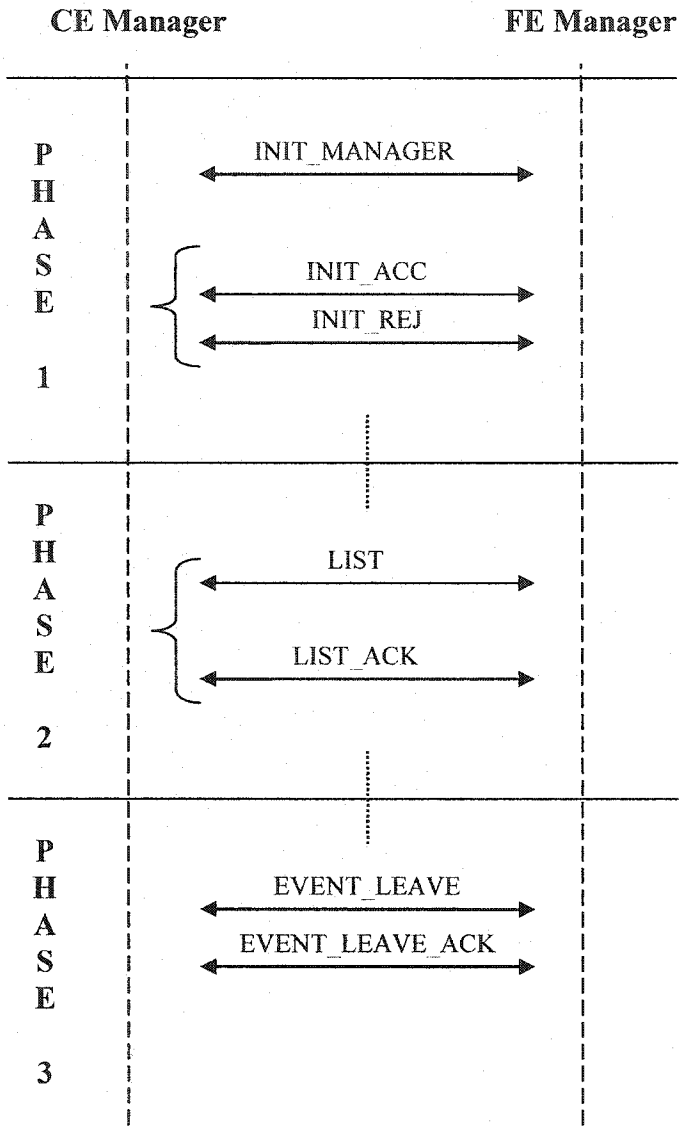


Figure 3.10 Interactions inter Managers complétées

Cette synthèse nous amène donc à proposer des messages dont les entêtes comprendraient les champs suivants que nous illustrons par la Figure 3.11 :

- Version : Champ de 5 bits contenant la version de ce protocole ;
- MsgType : Champ de 11 bits contenant le type du message (ce qui est amplement suffisant pour les types de messages déjà implantés et laisse une marge d'action dans le cas où d'autres seraient amenés à être définis) ;

- Length : Champ de 16 bits contenant la taille du message en octets permettant ainsi des charges utiles de 65 kilo-octets ;
- Identifier : Ce champ de 16 bits identifie un message. Les messages peuvent être longs (une liste de modélisation XML par exemple) et dépasser les 65 kilo-octets nominales d'un message. En effet, il faut noter que l'évolution des processeurs semble promettre dans un avenir proche, qu'un unique FE puisse être capable de traiter un large champ de fonctionnalités et donc avoir une modélisation XML d'une taille substantielle. Ainsi, les données peuvent donc nécessiter d'être fragmentées au sein de plusieurs messages. Et dans ce cas, les différents messages doivent garder le même identifiant ;
- Le bit F : un bit de fragmentation indiquant si le message est fragmenté. Si ce bit est mis à 0, il n'y a aucun autre fragment de message qui suivra, sinon, il indique que d'autres fragments suivront ;
- OffSet : Ce champ de 15 bits indique la position de ce fragment dans la globalité des données fragmentées.

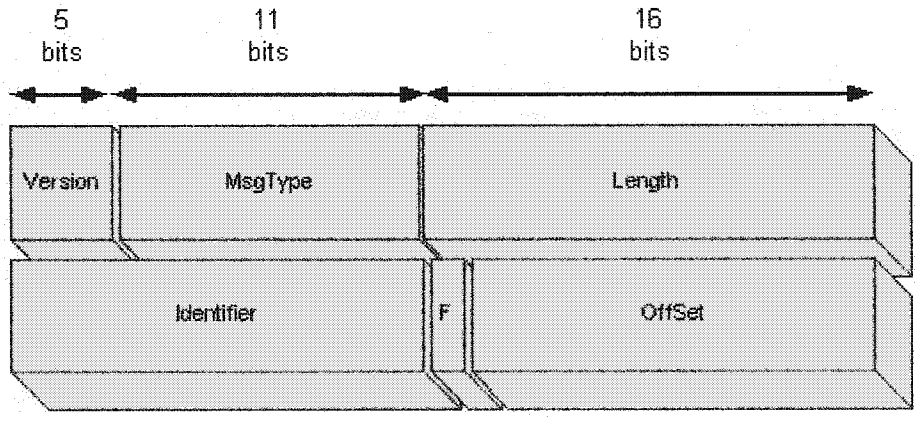


Figure 3.11 Entête du protocole de pré-association

Pour compléter notre approche, nous allons ici proposer, pour chaque regroupement de type de messages, une syntaxe particulière pour transporter la charge utile des messages. Cette charge aura naturellement besoin de présenter le type d'encapsulation pour les données. Nous utiliserons en fait une approche similaire à celle

du protocole FACT, c'est-à-dire une encapsulation TLV avec les trois champs suivants illustrés à la Figure 3.12 :

- EncapType (16 bits): contenant le type des données (XML, binaire, etc) :
- Param Length(16 bits): Taille de la charge utile :
- Value : Charge utile en tant que telle.

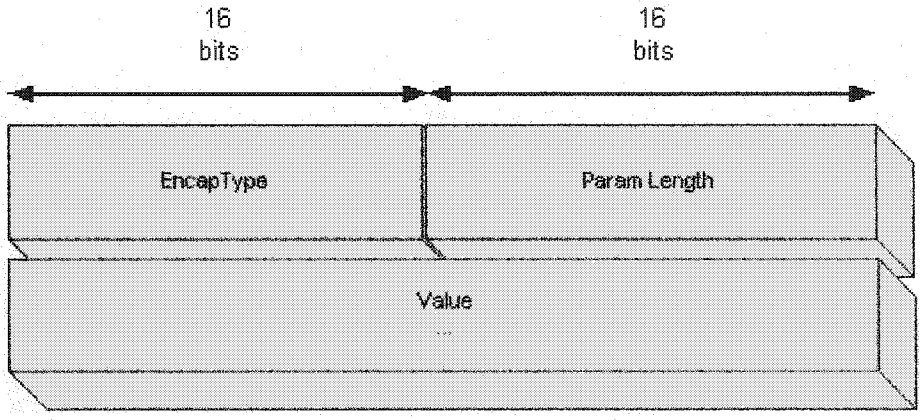


Figure 3.12 Charge des messages du protocole de pré-association

Étant donné que l'encapsulation des données est propre à chaque type de messages, nous ne présentons qu'en Annexe C la manière dont sont encapsulées les données de ce protocole pour chaque type de messages.

3.4 Modélisation de la phase de post-association

Une fois la phase de pré-association effectuée, les CEs et les FEs rentrent dans une seconde étape opérationnelle. Cette phase de post-association utilise, comme indiquée dans le chapitre 2, le protocole FACT. Ce dernier, en cours de spécification, permet aux éléments logiques de communiquer dans une relation de maître à esclave (CE à FE). Nous avons noté précédemment que les requis de l'architecture ForCES ont fait le choix de la polyvalence en optant pour différents moyens pour encapsuler les données. Nous allons exposer dans cette section nos choix de représentation des données.

3.4.1 Présentation des données

En fait, la problématique de l'hétérogénéité du matériel soulevée durant la phase de pré-association demeure la même. Afin de présenter les données sous forme de structure binaire, un API est en cours de développement au sein du NP Forum [NPF 2003], mais impose le choix des langages C/C++ pour l'implémentation. D'autre part, nous ne sommes pas sûr que l'ensemble des protagonistes industriels des processeurs réseau vont suivre ce modèle dans l'avenir. À la lumière de ces observations, nous avons adopté une approche basée sur XML pour représenter les messages de configuration du protocole FACT.

Lors de notre présentation de XML, nous avons noté la problématique du temps de traitement des documents XML par les *parsers*. Ce compromis entre la performance et la flexibilité vis-à-vis de l'implémentation se trouve plus complexe dans la phase de pré-association. En effet, la phase de pré-association ne se déroule qu'une fois au moment de l'initialisation, nous pouvons donc nous permettre un plus large compromis sur la performance. Par contre, dans la phase de post-association, nous pouvons avoir des messages de configuration lourds en XML qui seraient de fréquences élevées.

Pour avoir un ordre de grandeur du poids du choix de XML dans ce cas, nous avons essayé de prévoir le pire cas de figure. Après avoir identifié les différents types de messages FACT et leurs utilisations, il semble que ce soit les messages d'ajout d'entrées dans une table de routage (ou autre) qui peuvent s'avérer problématique. En effet, il suffit d'imaginer un FE implémentant un routeur virtuel classique. Si la partie *Contrôle* décide de créer une nouvelle table de routage, elle sera amenée à peupler cette table avec 100 000 entrées. Cela peut donc impliquer une indisponibilité du FE plus longue en comparaison d'une encapsulation des routes sous forme binaire. Nous pouvons manifester une réserve à cette option en remarquant que, dans un environnement multi-tâches si nous adoptons un *parser* de type événementiel, il est possible de cacher la latence du traitement XML. En effet, nous pouvons décomposer la liste des entrées en groupe de taille restreinte, par exemple 50 entrées, lors du parcours du document XML. Et une fois que nous avons rempli ses sous-groupes, nous pouvons appeler la fonction

les incluant dans la table de routage du FE. Or, l'inclusion d'entrée est un processus complexe (vérifications des entrées, etc.), et ce temps de traitement cachera le temps de parcours du document XML des 50 prochaines entrées.

Cette synthèse des avantages et inconvénients d'XML pour une partie des messages du protocole FACT nous amène à finalement à choisir entre la flexibilité et la performance de l'architecture. Étant donné que la flexibilité et la capacité de gérer du matériel hétérogène sont des aspects peu traités dans les propositions d'implémentations au sein du groupe de l'IETF, nous avons décidé d'apporter une vision complémentaire en usant d'XML. Ce choix se fonde aussi sur le fait que l'évolution des processeurs réseau est rapide, ce qui permettra dans l'avenir de meilleures performances des *parsers* (qui ne sont pas optimisés pour le moment pour ce type d'architecture). D'autre part, la forme humainement compréhensible du XML offre aussi une certaine facilité au niveau du développement informatique pur et du déboguage.

Il faut néanmoins avoir en tête que l'élaboration de ce type de messages de configuration se trouve être un domaine particulièrement vaste. Il est difficile pour le NP Forum d'introduire des structures de données assez larges pour apporter une vision exhaustive de cette problématique. En effet, dans notre approche, nous nous sommes basés sur le travail de synthèse du NP Forum et sur les différents standards sur les *Management Information Bases* (MIB) pour proposer un ensemble de structures de donnée en XML relatif au domaine suivant :

- Interface
- Routage IP avec dissociation de la table de routage en une table de préfixes et une table de prochains sauts pour les versions 4 et 6
- Lien entre le niveau 2 et 3 du système OSI
- Architecture DiffServ.

3.4.2 Ajout au protocole FACT

Afin de mieux comprendre l'impact de cette proposition, il nous faut revenir sur les messages du protocole FACT. Lors de notre introduction de ce protocole dans le

chapitre précédent, nous avons présenté la classe de message *Capabilities Control* (CAPCO) et les types qui lui sont associés. Une étude du document ietf-gopal-forces-fact [AUDU 2003] nous montre qu'il existe six types de message nécessitant une représentation des données spécifiques : *Capabilities Request, Capabilities Response, Configure Logic Components, Configure Logic Components Response, Query Request Query Response.*

Dans les faits, nous nous sommes limités aux quatre derniers d'en eux. Étant donné le grand nombre de structures de données présentées sous forme XML, nous allons illustrer notre approche en nous intéressant à un exemple simple et représentatif, celui de l'ajout de routes dans une table de routage. Dans le cadre d'une communication entre un CE et un FE qui implémente une table de routage, le CE se doit d'y insérer des routes. Ce mécanisme se retrouve en fait dans les messages *Configure Logic Components* du protocole FACT. Notre approche est donc de représenter les données sous forme XML pour assurer une compatibilité maximale entre CE et FE.

La Figure 3.13 illustre cette représentation. Nous retrouvons dans cet exemple les informations classiques relatives aux deux routes et un ensemble de prochain saut (*Next Hop* représenté ici par la balise NHRoute).

En plus de pouvoir peupler les différentes tables présentes dans le FE, notre modélisation apporte aussi la possibilité de créer des structures logiques à l'intérieur d'un FE. En effet, si nous reprenons l'exemple d'un routeur implémentant les fonctions de conditionnement *DiffServ* à la frontière d'un domaine, il est important de pouvoir créer et modifier des blocs fonctionnels comme une Queue.

La Figure 3.14 montre comment un bloc de ce type peut facilement être créé. Nous y retrouvons, d'une part, les informations issues des tables *DiffServ* comme spécifié dans la MIB [BAKE 2002] que nous indiquons dans la balise DS_Id et, d'autre part, les données paramétrant la Queue. Ici, les informations dans la balise QueueEntry indique la priorité de la queue ainsi que le débit minimal qu'elle peut traiter.

```
<RouteList vers="IPv4" size="2">
    <Route>
        <IPAddr>132.200.111.0</IPAddr>
        <Mask>255.255.255.0</Mask>
        <NHRouteList size="1">
            <NHRoute>
                <IPAddr>132.200.111.143</IPAddr>
                <EgrInf>2</EgrInf>
                <Metric>2</Metric>
            </NHRoute>
    </Route>
    <Route>
        <IPAddr>132.200.112.0</IPAddr>
        <Mask>255.255.255.0</Mask>
        <NHRouteList size="1">
            <NHRoute>
                <IPAddr>132.200.112.101</IPAddr>
                <EgrInf>1</EgrInf>
                <Metric>3</Metric>
            </NHRoute>
    </Route>
</RouteList>
```

Figure 3.13 Exemple de route en XML

```
<DS_Queue>
    <DS_Id>
        <PortId>1</PortId>
        <Dir>Out</Dir>
        <TableId>901</TableId>
        <InstId>1</InstId>
        <NextTableId>910</NextTableId>
        <NextInstId>2</NextInstId>
    </DS_Id>
    <QueueEntry MRCount="1">
        <MinRate>
            <Priority>2</Priority>
            <Absolute>1500</Absolute>
            <Relative>1000</Relative>
        </MinRate>
    </QueueEntry>
</DS_Queue>
```

Figure 3.14 Exemple de création d'une Queue DiffServ en XML

Nous nous sommes intéressés, lors de notre modélisation, aux structures utilisables pendant la phase de post-association aux fonctionnalités essentielles qu'un routeur se devrait d'implémenter. Dans un souci de lisibilité, nous ne détaillerons pas ici tous ces

types de messages mais ils peuvent être trouvés sous forme de DTDs en Annexe D. Nous avons indiqué au début de cette section que nous n'avons pas traité de manière exhaustive le cas des messages FACT des types *Query Request* et *Query Response* car ceux-ci s'orientent vers la représentation des statistiques, alors que notre travail s'est surtout porté sur les données nécessaires pour configurer un FE.

Notre analyse de la modélisation pour la phase de post-association nous a amené à modifier un point du protocole FACT. En effet, ce protocole se veut indépendant du type de représentation adopté pour les données qu'il véhicule. Or, si nous étudions les messages du type *Capabilities Response*, nous remarquons alors que la représentation des données de capacités est obligatoirement en binaire. Cela est en contradiction avec la liberté du choix de la modélisation. Nous proposons donc de pouvoir laisser le choix à l'utilisateur de remplacer les données encapsulées dans ce message par le modèle de Capacité du *FE Model* tel que présenté dans notre modélisation de la phase de pré-association.

CHAPITRE IV
IMPLÉMENTATION ET RÉSULTATS

Dans de ce chapitre, nous allons proposer une implémentation des concepts présentés le long de ce mémoire. L'objectif, ici, est d'émuler la séparation entre les parties *Contrôle* et la partie *Acheminement* pour prouver la viabilité de nos propositions grâce à un prototype fonctionnel. Dans un premier temps, nous allons présenter les modifications que nous avons apportées à notre proposition initiale. Ensuite, nous présenterons l'environnement d'implémentation, puis nous nous focaliserons sur la mise en œuvre de l'architecture. Et finalement, nous conclurons par une évaluation de performance de notre proposition.

4.1 Adaptation du modèle à des fins d'implémentation

Notre proposition vise à concevoir une architecture générique qui soit orientée vers l'interopérabilité et la survivabilité. Sa mise en exergue passe par une adaptation en vue de pouvoir apporter un prototype simple et représentatif de la réalité de la séparation entre les parties *Contrôle* et *Acheminement.* Pour cela, nous allons adapter et simplifier notre architecture pour l'implémenter. Nous décrirons les quatre points importants de modification : l'adaptation structurelle, l'adaptation de notre modélisation XML, l'adaptation vis-à-vis du protocole de communication et enfin l'adaptation en vue de sécuriser notre plate-forme.

4.1.1 Adaptation structurelle

La séparation logique entre le CE Manager et le FE Manager amenée par le groupe de travail ForCES correspond à une vision particulière de la gestion des CEs et des FEs. Dans un souci de simplification, notre architecture fusionne ces deux entités très semblables en un unique Manager. En effet, le principal impact de ce choix est le fait que les messages de type INIT_MANAGER, LIST et LIST_ACK n'auront pas à être implémentés.

4.1.2 Adaptation de la modélisation

L'architecture ForCES étant en plein développement, l'usage du *FE Model* n'y est pas entièrement défini. Pourtant, nous pouvons y voir un outil large utilisable autant pour la phase de pré-association que pour celle de la post-association. Il est important de noter qu'il faut que les mécanismes des phases de pré-association et de post-association puissent au maximum éviter la redondance.

Dans le cadre de notre implémentation nous userons du *FE Model* uniquement lors de la phase de pré-association et, pour être plus précis, seul le modèle de capacités (*Capability Model*) sera encapsulé dans les messages MODEL. En effet, le modèle d'états est plus spécifique à la phase de post-association. Nous pouvons noter d'ailleurs que, grâce à l'usage de *parsers* XML pour traiter les données, il est très aisé de ne pas tenir compte des données du *FE Model* dont nous ne nous servons pas lors de notre implémentation.

4.1.3 Adaptation du protocole de communication

Le choix de TIPC comme support de communication entre les différents éléments de l'architecture nous amène à modifier l'usage du protocole de pré-association. En effet, même si nous avions originellement l'idée d'utiliser TIPC, la conception de ce protocole a suivi les requis du groupe de travail ForCES stipulant que nous pouvons aussi utiliser n'importe quel protocole comme support.

De manière plus précise, les grandes adaptations que nous apportons se trouvent être liées à la structure des communications CE – FE. Les CEs, comme nous le

préciserons plus loin, peuvent être vus comme des serveurs possédant une certaine adresse et les FEs simplement comme des clients. Donc, pour leur fournir une base de communications, il suffit que le Manager attribue les adresses TIPC judicieuses aux CEs (dans le sens de la redondance mais nous y reviendrons plus tard). Ainsi, pour simplifier l'approche, le Manager n'aura pas à envoyer une liste de FEs avec laquelle le CE peut communiquer. Symétriquement, le Manager n'aura pas à assigner d'adresses spécifiques aux FEs. Autre point de simplification par rapport au modèle originel, le Manager n'aura qu'à fournir aux FEs qui le sollicite, une seule adresse de CE pour établir une communication.

4.1.4 Adaptation liée à la sécurité

Pour revenir à l'objectif de notre implémentation, nous cherchons à montrer la viabilité de nos concepts. Or, nous avons fait comme hypothèse que nous œuvrons dans une grappe d'ordinateurs que nous considérerons de confiance. La problématique de la sécurité se voit quelque peu éludée. Mais dans notre implémentation, nous avons voulu prouver que l'adjonction de mécanismes de sécurité s'avère simple à mettre en œuvre, si par exemple notre grappe d'ordinateurs devient plus large et nécessite plus d'un saut entre machines.

Nous nous sommes orientés vers la technologie IPSec pour sa simplicité d'utilisation étant donné qu'elle agit au niveau trois de la couche OSI et est transparente pour les couches supérieures. Bien que IPSec souffre de sa complexité intrinsèque (usage de IKE), nous avons opté pour le mécanisme ESP en mode Transport pour notre implémentation car il nous permet d'apporter un mécanisme de chiffrement qui soit flexible vis-à-vis du choix de l'algorithme mais aussi de sa transparence par rapport au reste de notre plate-forme.

4.2 Détails et environnement d'implémentation

Le développement d'une telle plate-forme nécessite de prendre en compte un grand nombre de mécanismes, autant du point de vue de la partie *Contrôle* que de la partie *Acheminement.* Comme support de développement, nous avons opté pour la plate-

forme complémentant le processeur réseau IXP 2400 d'Intel© présenté dans le second chapitre. Ce choix est basé sur le fait que Intel© propose avec son processeur une plate-forme logicielle gérant la dissociation des parties *Contrôle* et *Acheminement* dénommée *Control Plane Platform Development Kit* (noté CP PDK™ et qui est une marque de commerce déposée par Intel©). Dans la suite de notre exposé, nous allons présenter sommairement le fonctionnement de cette plate-forme de développement. Ensuite, nous présenterons le *parser* XML Expat, puis le logiciel FreeS/WAN qui est une solution de sécurisation implémentant IPSec. Finalement, nous décrirons les détails de l'usage de TIPC comme médium de communication.

4.2.1 Présentation du CP PDK™

Comme nous venons de le voir, le CP PDK™ d'Intel© nous fournit une plate-forme de base pour implémenter notre proposition d'architecture. Avant de décrire son fonctionnement, nous avons choisi de laisser les termes originaux en langue anglaise, car ce logiciel est une marque déposée. Tout d'abord, la structure du CP PDK™ définit trois principaux composants : le *Control Plane Module*, le *Transport Plugins* et le *Forwarding Plane*. Nous y retrouvons simplement les concepts que nous étudions.

Les principales fonctionnalités du *Control Plane Module* sont les suivantes :

- Il gère un ensemble de *Forwarding Planes*, et est en mesure de répliquer les données sur chacun d'entre eux ;
- Il gère leurs topologies ;
- Il possède des mécanismes de découverte et d'initialisation ;
- Il implémente une partie des APIs du NP Forum comme le *IPv4 Unicast Forwarding* API, ainsi que d'autres API d'acheminement tel que *DiffServ* et MPLS :
- Il utilise une couche d'abstraction pour assurer l'indépendance vis-à-vis du matériel et du système d'exploitation.

Pour ce qui est du *Forwarding Plane Module*, il s'agit en fait d'un API utilisé comme une interface de programmation pour la partie *Acheminement.* Cette entité reçoit,

par le biais du *Transport Plugins*, les invocations du NPF API et les transcrit en des invocations propres à son environnement, le processeur IXP 2400. Ce module se compose d'entités qui sont en fait des gestionnaires ayant diverses responsabilités. Les plus importantes d'entre elles sont : le gestionnaire de découverte et d'association, le gestionnaire d'IPv4 qui implémente le NPF API, ou encore les gestionnaires de Qualité de Service et de MPLS.

D'autre part, le *Transport Plugins* réalise l'interface logicielle entre ces deux modules. Ce module est au centre de nos modifications en vue de l'implémentation de notre proposition. Son architecture se décompose en quatre entités :

- *FP Plugin API* : Ce module est un API intégré au *Control Plane Module* afin de rendre transparents les appels en direction du *Forwarding Plane Module* ;
- *Backend API* : Réciproquement, cet API est intégré au *Forwarding Plane Module* pour l'interfaçage transparent du *Control Plane Module* ;
- *Transport Protocol* : Cette entité correspond aux deux démons de communication sous-jacents aux deux APIs énoncés précédemment ;
- *Interconnect Abstraction Layer* : Cette couche logicielle vise simplement à cacher au *Transport Protocol* les détails de l'interconnexion.

L'implémentation du CP PDK™ 1.2 qui nous a été fournie, a été réalisée en langage C sous le système d'exploitation Linux. D'ailleurs, ce logiciel sur laquelle nous avons travaillé est une version de développement non encore finalisée.

4.2.2 Présentation du *parser* XML Expat

L'usage de XML pour structurer et présenter les données de notre architecture nous amène à nous intéresser aux outils disponibles pour traiter les documents XML. Étant donné que notre plate-forme s'exécute sur le système d'exploitation Linux et que l'ensemble de notre implémentation se base sur le langage C, nous avons effectué une étude des principaux *parsers* existants. D'autre part, nous avons opté pour un *parser* de type SAX (événementiel) plutôt que de type DOM afin d'obtenir de meilleures performances et un moindre usage de la mémoire comme vu au second chapitre. Cette

étude, réalisée par deux étudiants du LARIM (Pascal Bonheur et Nicolas Verducou) en projet de fin d'étude, s'est focalisée sur trois *parsers* vérifiant les requis précédents : LibXML2, Expat et RXP.

La comparaison s'est axée sur deux points : le temps de traitement de document XML qui était ici des documents d'ajout de route comme vu à la section 3.4.2 du chapitre précédent, et l'utilisation de la mémoire en fonction de la taille du document XML à traiter. L'analyse des performances nous indique que, pour le temps de traitement comme pour l'usage de la mémoire, le *parser* Expat se trouve être le plus performant, bien que LibXML2 le talonne.

Fort de cette étude comparative, nous nous sommes basés sur le *parser* Expat dans sa version 1.95 pour la conversion des documents XML représentant les modèles en structures gérables par le Manager et les données du protocole FACT en structures conformes aux APIs du NP Forum utilisés par le CP PDK™.

4.2.3 Implémentation de TIPC

TIPC se présente sous la forme d'un module chargeable dans le noyau de Linux. Nous avons vu précédemment que TIPC permet d'utiliser un grand nombre de types d'interface appelé *Bearer* dans sa terminologie, comme UDP, Ethernet ou encore ATM. Son implémentation, initialement codée en C++, satisfait aux exigences du code C ANSI et est basée sur différentes couches d'adaptation rendant plus abordable son portage d'une architecture à une autre.

Le fonctionnement de TIPC est basé sur un protocole interne servant pour la découverte et le routage au sein de la grappe d'ordinateurs. Ces fonctionnalités dépendent du type de *Bearer* utilisé. Pour l'adaptation Ethernet, la découverte utilise l'adresse *broadcast* MAC alors que pour UDP ce sont des adresses IP de type *multicast*.

Dans le cadre de notre implémentation, nous nous sommes focalisés sur Ethernet et UDP. TIPC possède la particularité d'être simple d'utilisation car son interface de programmation est très proche de celui des sockets BSD communément utilisés. La

Figure 4.1 illustre la structure interne de TIPC. Dans notre implémentation, nous avons utilisé la version 0.97 de ce logiciel.

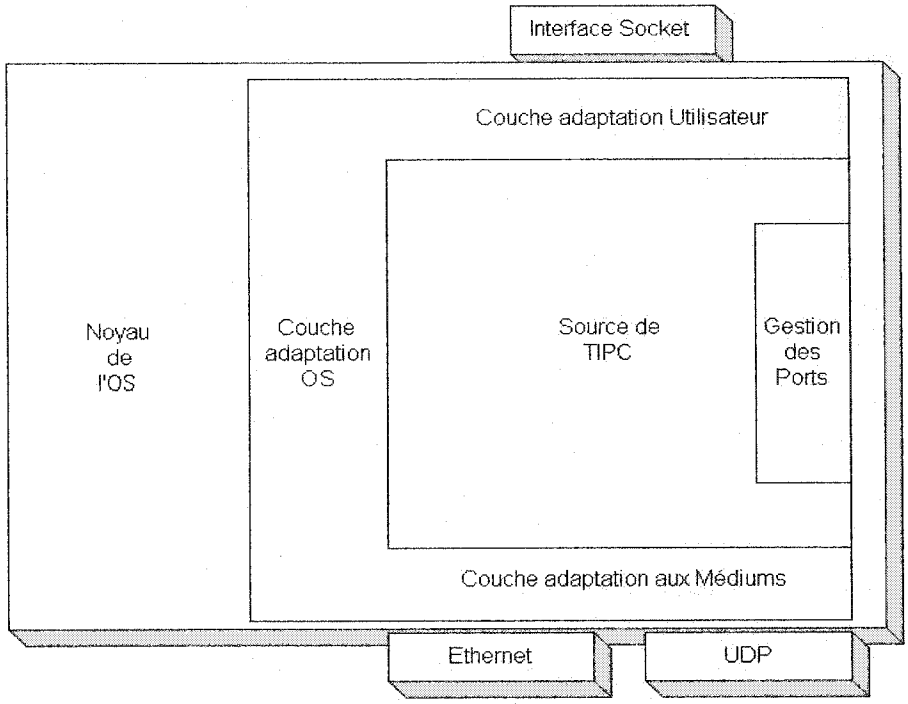


Figure 4.1 Structure du code de TIPC

4.2.4 Présentation de FreeS/WAN

FreeS/Wan [FREE 2003] est une implémentation de IPSec sous Linux, qui se présente là aussi sous la forme d'un module chargeable dans le noyau. Cette implémentation se compose de trois composants. Le premier est le programme KLIPS qui implémente les protocoles AH et ESP, présentés au chapitre 2, ainsi qu'un gestionnaire d'interception de paquets au niveau du noyau de Linux. Le second composant est le démon Pluto qui incarne le protocole IKE pour la négociation de connexions entre machines. Le dernier composant est en fait un ensemble de scripts d'administration pour IPSec. FreeS/Wan complète alors la pile IPv4 de Linux pour assurer le chiffrage. D'autre part, étant donné que IPSec est obligatoire avec IPv6, les concepteurs de ce projet travaillent à intégrer FreeS/Wan à l'intérieur de la pile IPv6 de Linux.

FreeS/Wan supporte la gestion des clefs privées/publics (RSA) ainsi que les clefs partagées. La gestion des certificats X 509 nécessite un patch logiciel. Pour ce qui des méthodes cryptographiques, FreeS/Wan utilise MD5 pour l'authentification et Triple DES pour le chiffrement.

Dans notre implémentation, nous avons utilisé cette plate-forme pour chiffrer les communications à l'aide du protocole ESP en mode Transport. La clef que nous avons utilisée est une clef partagée par souci de facilité. Par contre, l'usage de clefs asymétriques est fortement recommandé pour une véritable utilisation en milieu non protégé. Nous pouvons noter que la gestion du *multicast* par IPSec est à l'état expérimental et par conséquent FreeS/Wan ne l'implémente pas. Pourtant, TIPC utilise le mécanisme *multicast* de IP lorsque l'on utilise l'adaptation UDP afin de découvrir les machines de la grappe. Pour pallier ce problème, nous avons conçu un module qui fournit simplement à TIPC, sur une machine quelconque, l'adresse IP d'une nouvelle machine disponible dans la grappe.

Ce mécanisme diminue les fonctionnalités de TIPC en nous privant du mécanisme de découverte, mais il se trouve que c'est une solution pour prouver dans un premier temps la faisabilité de la sécurisation de notre architecture. De plus, l'implémentation du *multicast* est une étape à venir dans l'évolution de FreeS/Wan. Il existe un prototype d'implantation du *multicast* avec IPSec développé par les laboratoires de IBM© (l'implantation de IPSec sur notre architecture a été réalisée en collaboration avec Damien Bayle, un étudiant du LARIM en projet de fin d'étude).

4.3 Mise en œuvre de l'architecture

Maintenant que nous avons vu l'ensemble de l'environnement d'implémentation, nous pouvons voir plus dans le détail la structure de notre implémentation. Tout d'abord, nous décrirons en quoi nous avons modifié le CP PDK™, ensuite nous nous focaliserons sur la manière dont nous avons implémenté le Manager et finalement nous étudierons notre implémentation des CEs et des FEs.

4.3.1 Modification du CP PDK™

La mise en œuvre de notre architecture passe par la modification du CP PDK™. La partie sur laquelle nous allons nous focaliser est le *Transport Plugins* autant du côté du *Control Plane Module* que du *Forwarding Plane Module*. Au point de départ, ce module utilisait COPS-PR [CHAN 2001b] pour assurer les communications en les présentant sous forme de politiques. Puis, ce module a récemment changé de médium pour adopter le protocole FLEX qui est une version simplifiée du protocole FACT au-dessus de TCP/IP, qui de plus est spécifique au CP PDK™.

Lors de notre première implémentation nous avons implémenté les spécifications de FACT au-dessus de TIPC. Mais quand Intel© a décidé de faire évoluer sa plate-forme vers FLEX, nous avons continué à développer avec FACT, car l'objectif était de montrer la viabilité de TIPC comme choix de protocole de communication de FACT.

D'un point de vue plus technique, notre travail a surtout consisté à remplacer le module *Transport Protocol* présenté plus haut et aussi d'adapter le *FP Plugin API* et le *Backend Plugin API*. D'autre part, ces modifications ont aussi été guidées par le besoin de les rendre transparentes vis-à-vis des modules en amont et en aval du *Transport Plugin*.

4.3.2 Structure du Manager

Selon les spécifications du groupe de travail ForCES, le Manager a la responsabilité de gérer l'adressage au sein d'un élément de réseau. Cette fonctionnalité se trouve réalisée suivant le protocole de pré-association. Mais, la manière d'assurer l'adressage permet aussi de gérer la survivabilité et la performance de l'élément de réseau. En effet, nous pouvons décomposer le comportement du Manager en deux points : l'affectation 'intelligente' des adresses TIPC aux CEs et l'appariement de CEs aux FEs de manière optimale. Cette architecture est illustrée à la Figure 4.2.

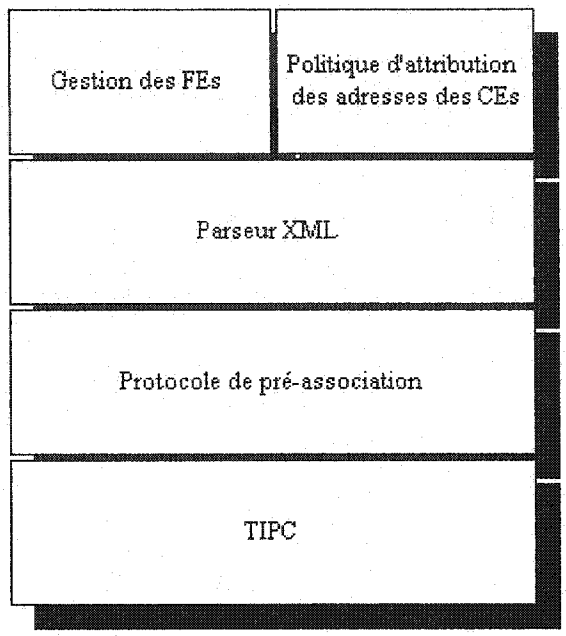


Figure 4.2 Structure du Manager

- L'affectation 'intelligente' des adresses TIPC aux CEs :

Lorsqu'un CE se connecte au Manager, ce dernier lui fournit sa spécification XML. Ainsi, le Manager peut savoir les fonctionnalités qui lui sont nécessaires, et aussi si ce dernier supporte la redondance de CEs. Dans ce cas de figure, le Manager peut décider s'il veut faire de ce CE un CE de sauvegarde et, pour cela, il aura à affecter à ce CE une adresse TIPC qui est la même que celle d'un autre CE qui possède les mêmes besoins. Pour illustrer ce concept de politique d'affectation d'adresse, le Manager vérifie systématiquement, dans notre implémentation, si le CE peut être redondant et alors vérifie dans sa liste de CEs déjà enregistrés s'il en existe un qui est compatible et ne dispose pas déjà d'un CE de sauvegarde.

- L'appariement de CEs aux FEs de manière optimale :

Étant donné qu'il relève uniquement du CE s'il désire gérer un FE, le choix de l'appariement se trouve entièrement dans la phase de post-association. Pourtant, si nous avons un grand nombre de FEs, le nombre de connexions à un CE qui se solde par un échec pour le FE peut s'avérer élevé. Ainsi, l'idée que nous avons proposée est que le Manager, après avoir récupéré la modélisation d'un FE, cherche le CE qui

lui est le plus complémentaire dans sa liste de CEs. Et, le Manager vérifie que le nombre de FE qui ont essayé de s'apparier avec ce dernier est inférieur au nombre de FEs qu'il peut administrer (ce compteur est décrémenté si un FE que l'on pensait connecté à un certain CE, envoie un message de type EVENT_INACTIVE). Tous ces mécanismes visent à minimiser le nombre de communications entre les FEs et les CEs, car le Manager est la seule entité à avoir une vision globale de l'élément de réseau. D'ailleurs, les deux blocs logiques du bas de la Figure 4.2 sont en fait l'implémentation en tant que telle du protocole de pré-association.

La couche du protocole est basée sur un serveur multi-tâches TIPC qui souscrit le nom de port possédant le type 17770 et gérant les instances dont les bornes inférieure et supérieure sont 0 et 99. Son architecture logique peut être simplement représentée comme à la Figure 4.3. Nous avons utilisé la librairie pthread (POSIX) pour la création des *threads*.

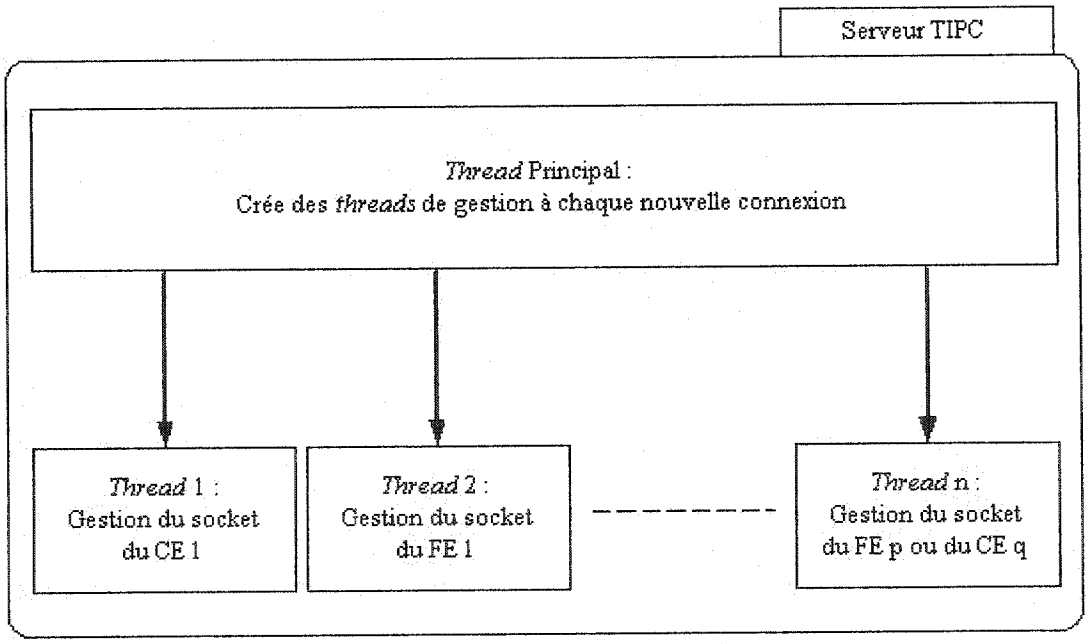


Figure 4.3 Serveur multi-tâches du Manager

4.3.3 Structure des CEs et des FEs

La structure des CEs et des FEs se divise en deux concepts : la gestion de la phase de pré-association et la gestion des communications de CEs à FEs. Dans notre implémentation, nous avons décidé de séparer au mieux le développement des modules prenant en charge ces deux concepts.

Pour ce faire, nous avons implémenté les structures des messages du protocole de pré-association au-dessus de TIPC dans un module qui soit séparé de l'implémentation du client qui initie les communications vers le Manager. Cette approche a été pratiquée autant pour les CEs que pour les FEs.

Pour ce qui est de l'implantation du protocole FACT, il faut considérer de manière séparée le cas des FEs et celui des CEs :

- Implémentation de FACT pour les CEs :

  Comme nous l'avons vue lors de notre présentation de l'architecture ForCES, les CEs sont en fait des serveurs attendant les demandes de connexions des FEs. Ainsi, la structure du protocole FACT pour les CEs est basée sur un serveur TIPC comme illustré à la Figure 4.3 pour le Manager. Mais pour ce qui est de l'encapsulation des données, nous avons opté comme pour la phase de pré-association pour séparer cette entité logicielle dans un module à part (qui est en réalité commun aux FEs). Ce module possède la particularité de permettre au code l'utilisant qui ici est le serveur TIPC représentant le CE, de lui laisser le choix de la manière de traiter la charge utile des messages. Cela signifie que notre implémentation du CE possède ses propres fonctionnalités pour traiter les données en XML qui ne sont pas communes aux FEs, chacun traitant des données différentes. La Figure 4.4 a pour but d'illustrer notre approche modulaire de l'architecture des CEs.

- Implémentation de FACT pour les FEs

  Il est évident que ces deux types d'entités CEs et FEs possèdent beaucoup de similarité du point de vue de l'implémentation. Comme nous l'avons expliqué dans le chapitre précédent, les FEs sont en fait des clients TIPC. La grande différence d'implémentation entre le CE et le FE réside dans les fonctions de traitement des

données réaliser par le *Parser*. En effet, comme nous le montre les DTDs de l'Annexe D, le FE doit convertir toutes les entrées représentées en XML en structure NPF API qui sont plus nombreuses au niveau du FE. Pour conclure sur l'implémentation de FACT pour les FEs, nous pouvons représenter la structure modulaire des FEs comme à la Figure 4.4.

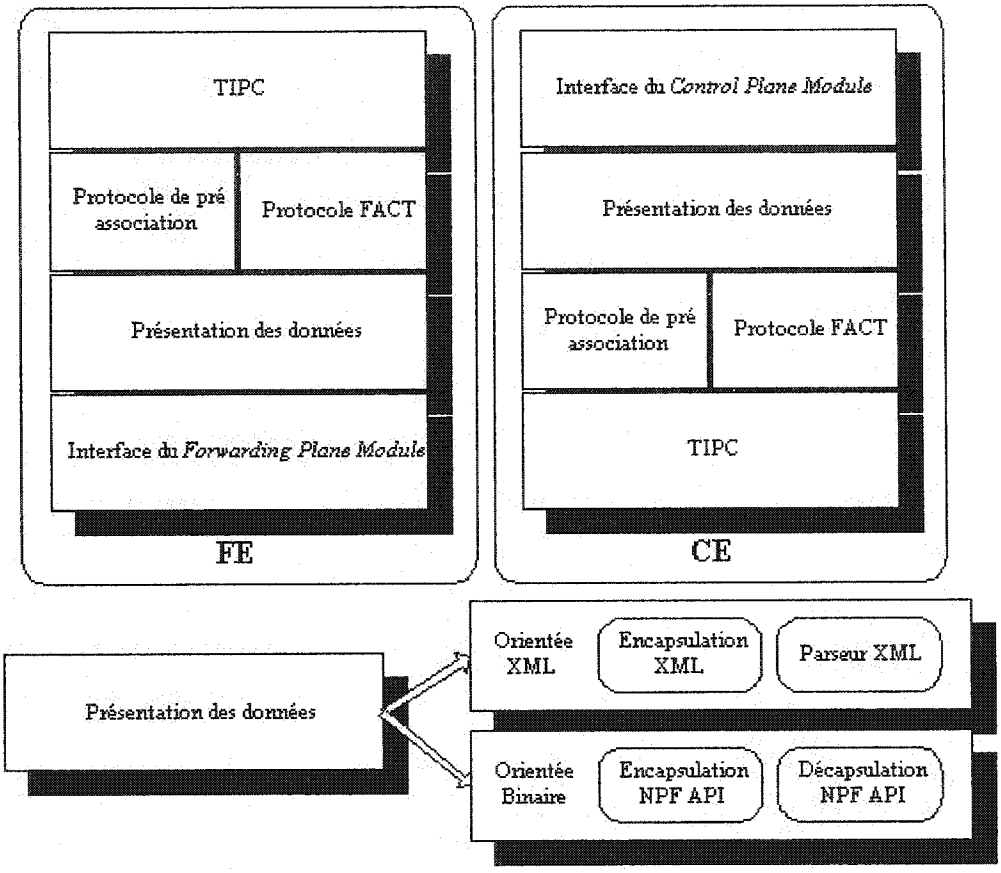


Figure 4.4 Architecture modulaire d'un FE

4.4 Plan d'expérience

Afin de tester notre proposition, nous avons opté pour une architecture basée sur deux machines utilisant le système d'exploitation Linux avec le noyau 2.4.18-3. Pour simuler les parties *Contrôle* et *Acheminement*, nous utilisons le CP PDK™ 1.2 présenté précédemment. Les deux machines sont interconnectées par un lien Ethernet 100 Mb/s comme illustré à la Figure 4.5 présentant notre architecture matérielle.

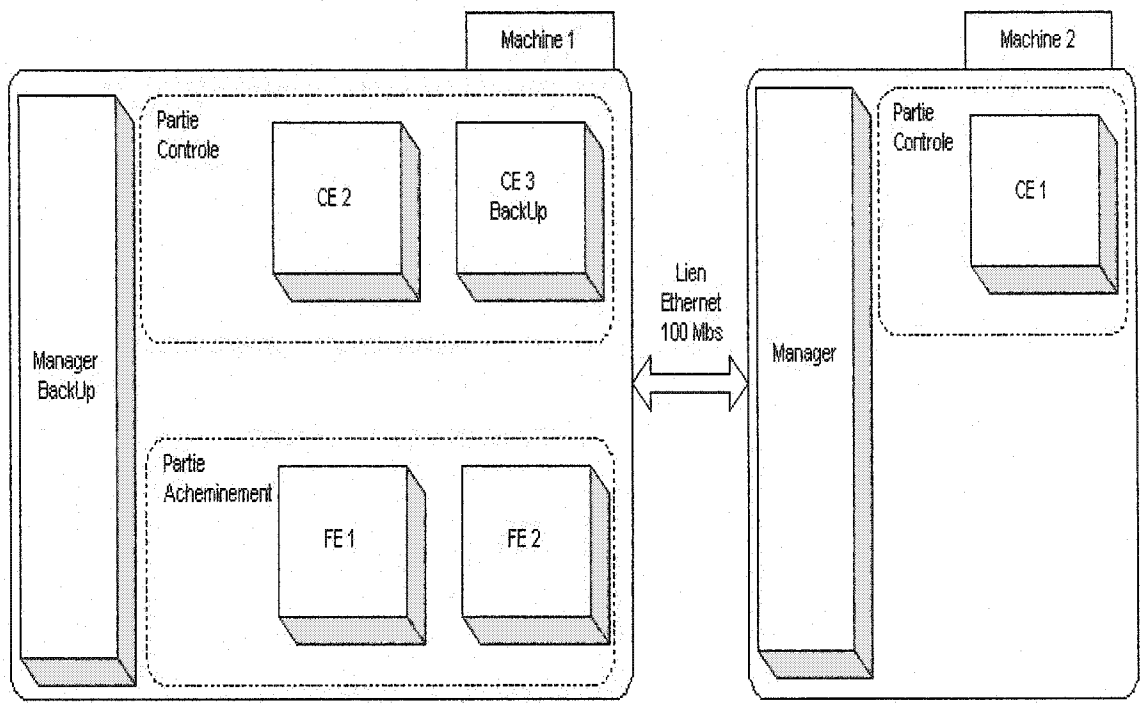


Figure 4.5 Architecture de test

Afin de mieux préciser l'architecture, il faut noter que les deux machines utilisées sont des ordinateurs portables. Le processeur de la Machine 1 est un Celeron cadencé à 733 MHz et, pour la seconde machine, il s'agit d'un Pentium III à 500 MHz.

Dans les faits, nous avons exécuté deux types de tests. Le premier consiste en un test fonctionnel qui vise à vérifier si l'architecture que nous avons implémentée réalise bien nos requis de robustesse et de flexibilité. La seconde série est un test de performance de notre architecture qui se compose d'un ensemble de trois cas de figures notés Cas 1, Cas 2 et Cas 3.

Notre test fonctionnel a pour but d'assurer que notre architecture supporte bien les fonctionnalités de redondance et survivabilité énoncées. De plus, il nous faut vérifier que la gestion de l'adressage des CEs reste cohérente avec notre approche. Pour cela, nous avons choisi de le subdiviser en deux sous cas. Le premier est une étude de la survivabilité de nos concepts :

Sous-cas 1 :

- Étape 1 : Sur la machine 2, nous lançons un premier Manager. Ensuite, nous démarrons sur la même machine un CE dont la modélisation XML indique qu'il supporte la redondance et nécessite des fonctionnalités de Qualité de Service DiffServ.
- Étape 2 : Sur la machine 1, nous démarrons deux CEs. Le premier est un CE gérant la redondance et nécessitant les mêmes fonctionnalités que le CE de l'étape 1. Le second est un CE qui requière des fonctionnalités IPv4. Puis, de la même manière, nous démarrons deux FEs, dont le premier fournit le support des fonctions de conditionnement *DiffServ*, alors que le second implémente un *IPv4 forwarder*.
- Étape 3 : Nous démarrons un second Manager sur la machine 1.
- Étape 4 : Chaque FE envoie un message FACT de type *Join Request* pour se connecter à son CE.
- Étape 5 : Nous coupons le lien Ethernet entre les deux machines.

Le second sous-cas vise simplement à vérifier le comportement de notre plate-forme si un FE ne réussit pas à trouver un CE avec lequel rentrer en connexion après la phase de pré-association. En fait, c'est une vérification du fonctionnement de l'étape 2 de notre protocole de pré-association. Pour cela, nous nous placerons uniquement sur la machine 2 et nous réaliserons les étapes suivantes :

Sous-cas 2 :

- Étape 1 : Nous démarrons sur la Machine 2 un Manager.
- Étape 2 : Nous lançons le FE implémentant *DiffServ* que nous avons présenté à l'étape 2 du sous cas précédent.
- Étape 3 : Nous forçons le FE à essayer d'envoyer un message FACT de type *Join Request*.
- Étape 4 : Nous démarrons sur la Machine 2 un CE dont la modélisation XML nous indique qu'il nécessite des fonctionnalités de Qualité de Service *DiffServ*.

Pour ce qui est du premier cas de la série de test de performance, il s'agit là de quantifier le temps mis pour assurer la reconnexion automatique des CEs et FEs en cas de perte du lien physique entre les machines 1 et 2.

Le second cas correspond à une mesure des performances de notre architecture sous différentes configurations de TIPC. Mais, le point le plus important de ce test est de mesurer l'impact de l'usage de IPSec pour sécuriser l'architecture.

Le dernier cas vise à mesurer la différence de performance entre l'encapsulation XML des données et celle binaire avec directement avec les structures de données du NPF API.

4.5 Mise en œuvre de l'architecture et évaluation de performance

Dans cette section, nous présenterons les deux types de tests relatifs à notre plan d'expérience.

4.5.1 Test fonctionnel et mise en œuvre

Dans cette partie nous allons présenter sous la forme de captures d'écrans le comportement de notre architecture pour les deux sous cas présentés dans notre plan d'expérience.

Sous-cas 1 :

Nous allons présenter trois séries de captures d'écrans pour illustrer le comportement de notre architecture.

La première correspond à l'état de notre système après la phase de pré-association, c'est à dire entre l'étape 3 et l'étape 4 sur les machines 1 et 2; elle est illustré aux figures 4.6 et 4.7.

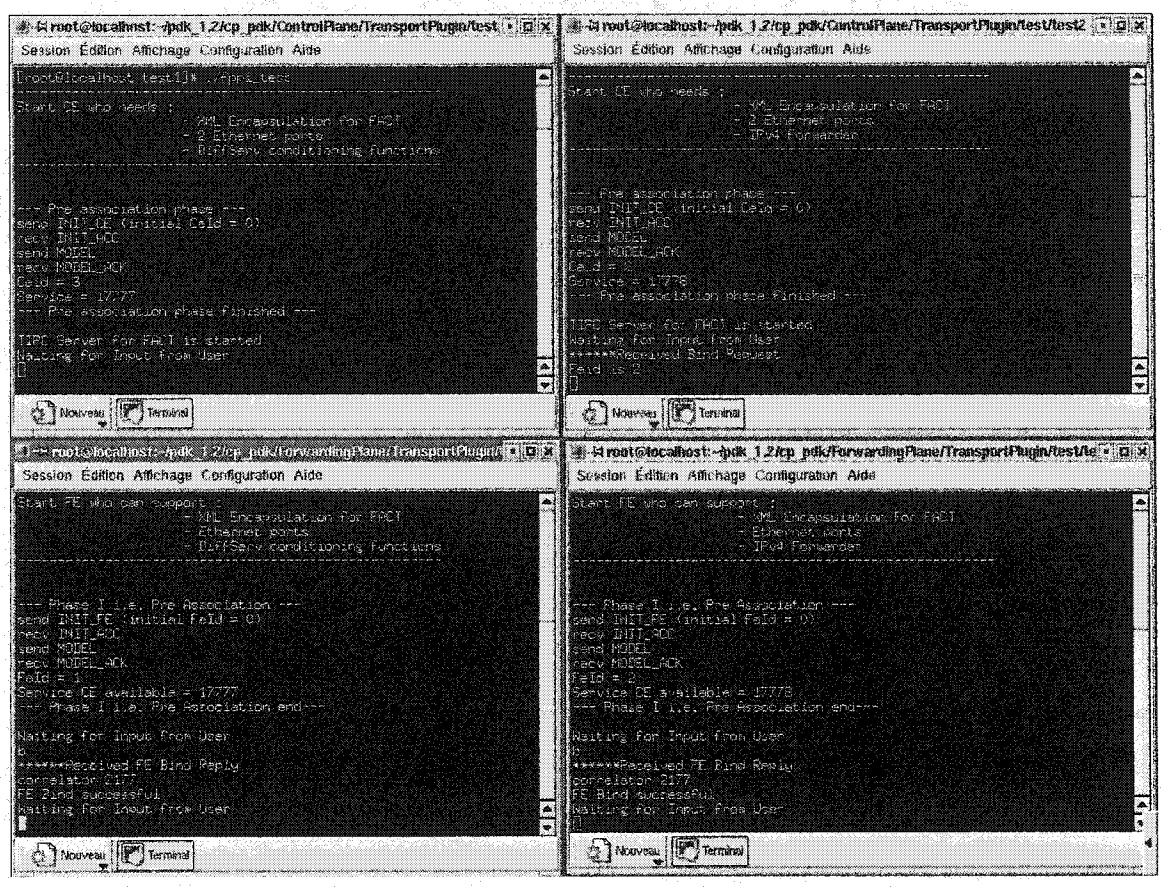


Figure 4.6 État des CEs et FEs après la phase de pré-association (Machine 1)

Comme nous pouvons le voir, le Manager a réussit à attribuer à chaque CE un identifiant mais aussi un nom de port TIPC. La première remarque que nous pouvons faire concerne la gestion de la redondance. Comme nous avons ici deux CEs identiques en terme de *CE Specification* (CE 1 et CE 3), le Manager leur a attribué le même nom de port et fait ainsi de CE 3 le CE de sauvegarde CE 1. Pour ce qui est des FEs, le Manager a bien réussi, grâce à la phase de pré-association, à leur fournir les noms de ports des CEs qui leur sont complémentaires. La Figure 4.7 a surtout pour objectif de nous montrer les données sous forme de documents XML que le Manager reçoit et traite pour assurer la phase de pré-association. Nous pouvons aussi remarquer que les messages de type *Join Request* du protocole FACT ont bien été attribués aux bons couples FE 1 – CE 1 et FE 2 – CE 2.

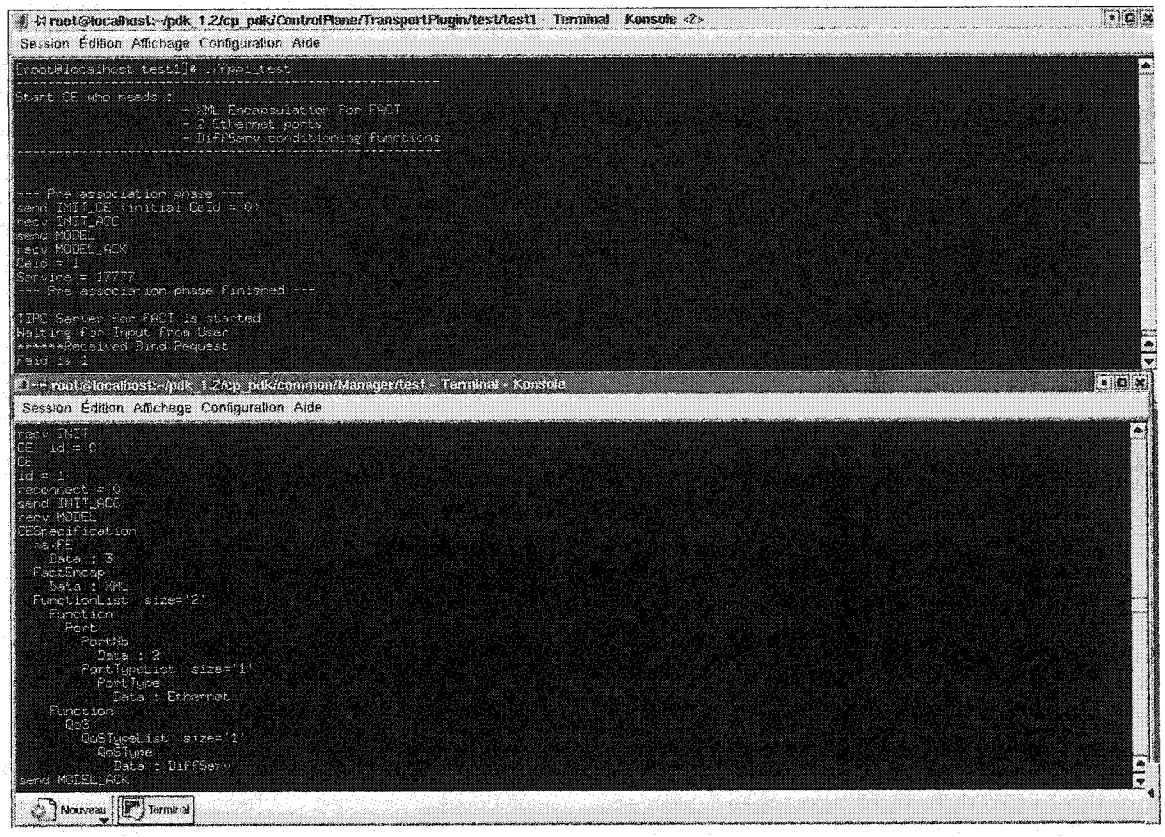


Figure 4.7 État du CE et du Manager après la phase de pré-association (Machine 2)

La seconde série de captures, quant à elle, vise à nous illustrer le comportement des PEs de la Machine 1 après la panne simulée de la Machine 2.

À la Figure 4.8, nous remarquons que les PEs ont bien détecté la panne. D'une part, ils se sont reconnectés au Manager de sauvegarde en recommençant une phase de pré-association dont les messages INIT encapsulent leurs anciennes configurations (nous voyons sur le CE 3 par exemple qu'il indique son CE Id qui est resté identique après l'opération). La Figure 4.9 nous montre l'affichage du Manager de sauvegarde qui gère les connexions des PEs. D'autre part, à la Figure 4.8, nous voyons que le FE 1 ayant détecté la perte de connexion au CE 1, s'est automatiquement reconnecté auprès du même nom de port et a ainsi réussi à envoyer un nouveau message *Join Request* du protocole FACT au CE adéquat.

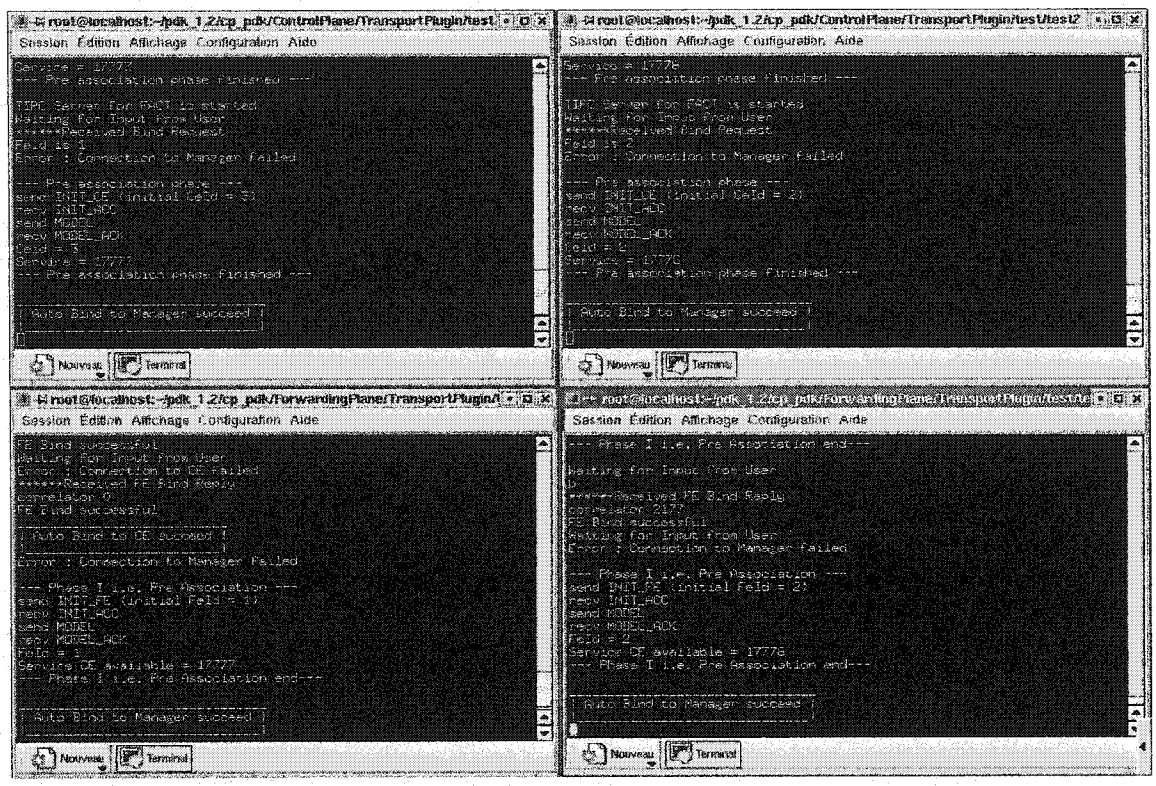


Figure 4.8 État des CEs et FEs après la coupure de connexion (Machine 1)

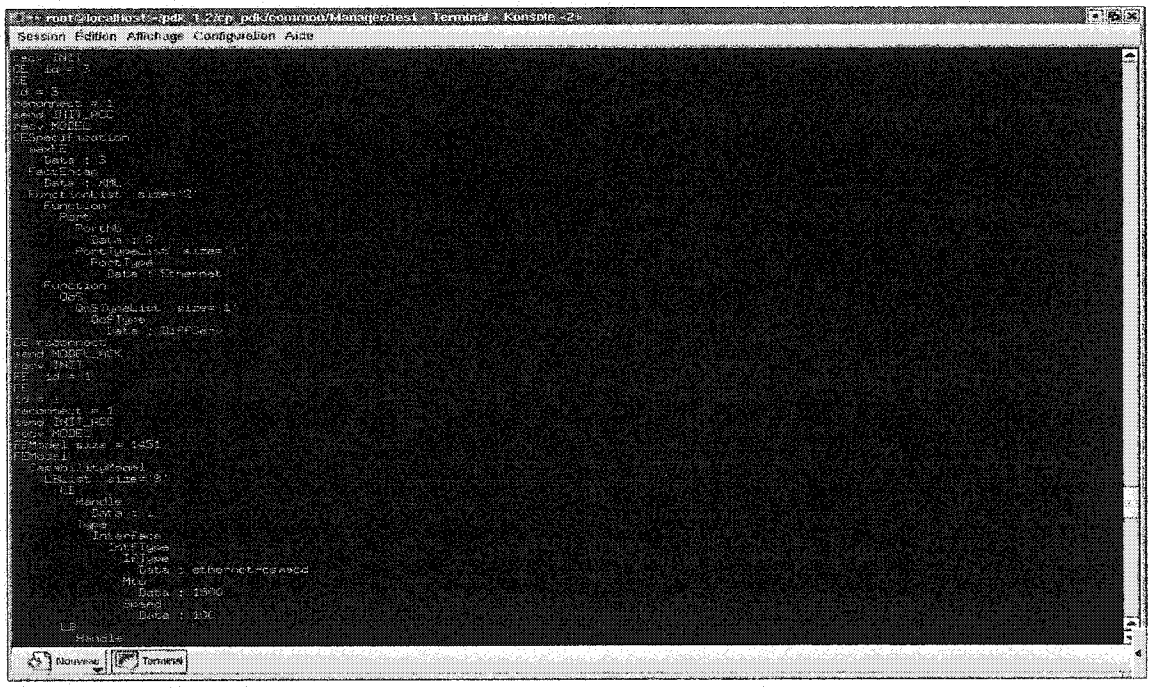


Figure 4.9 État du Manager de sauvegarde (Machine 1)

La dernière série correspond à la vérification du fonctionnement des messages du protocole FACT dont leurs données sont représentées par des documents XML comme nous le constatons à la Figure 4.10.

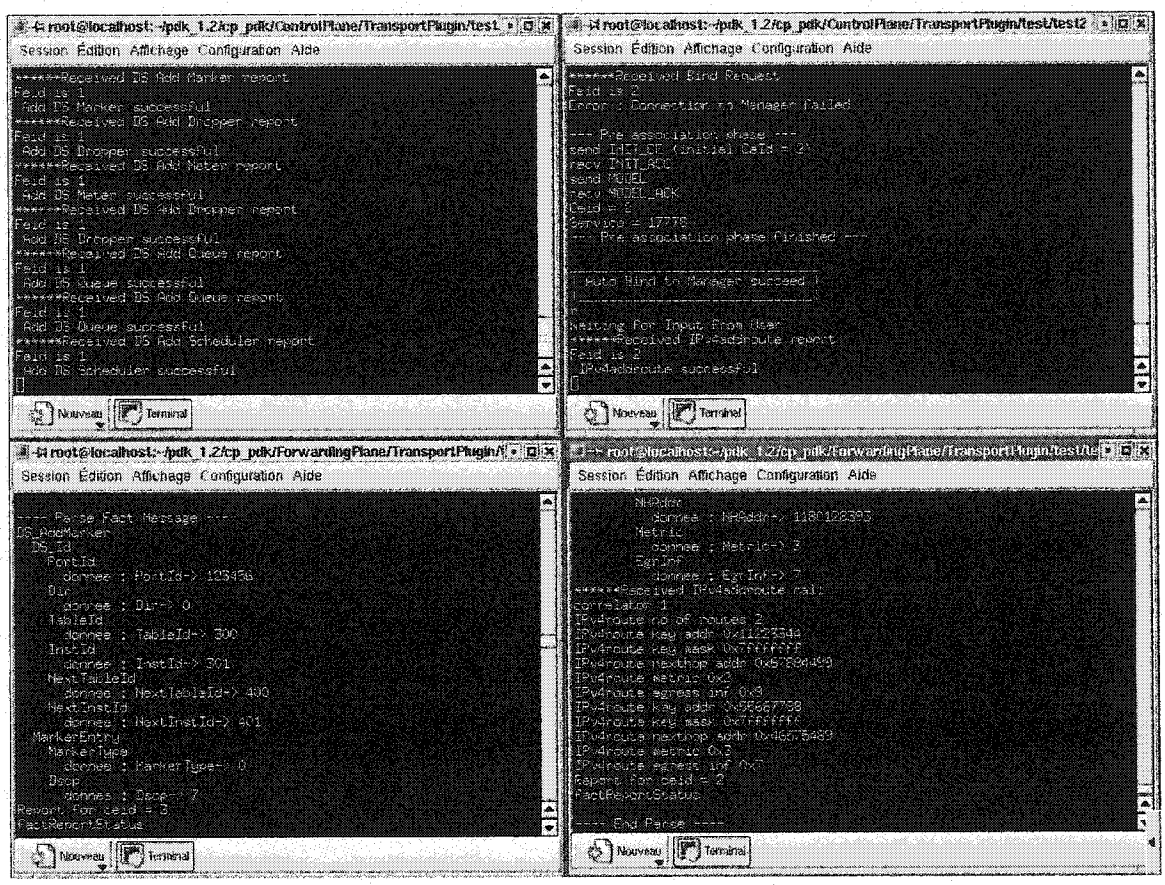


Figure 4.10 Test des messages FACT (Machine 2)

Nous voyons que les messages de création d'une structure *DiffServ* du CE 3 ont été correctement reçus par le FE 1 et que les données ont bien été converties sous forme de structure du NPF API. En réalité, cette structure *DiffServ* est la même que celle présentée au chapitre 2 à la Figure 2.3. De la même manière, le CE 2 a envoyé un message d'ajout de route au FE 2 qui l'a reçu et traduit avec succès.

Sous cas 2 :

Pour ce second test fonctionnel, nous allons réaliser deux captures d'écran pour illustrer le comportement du CE et du FE. La première capture, correspondant à la Figure 4.11, illustre l'état du système après l'étape 3. Nous pouvons aisément voir que le FE, ne possédant pas d'adresse de CE avec lequel rentrer en communication, ne peut envoyer un message FACT de type *Join Request*. En effet, la Figure 4.11 indique que le type du nom de port fourni par le Manager après la phase de pré-association est égale à zéro. Et donc, le FE envoie des messages EVENT_INACTIVE au Manager avec une périodicité de 5 secondes que nous avons nous même fixée.

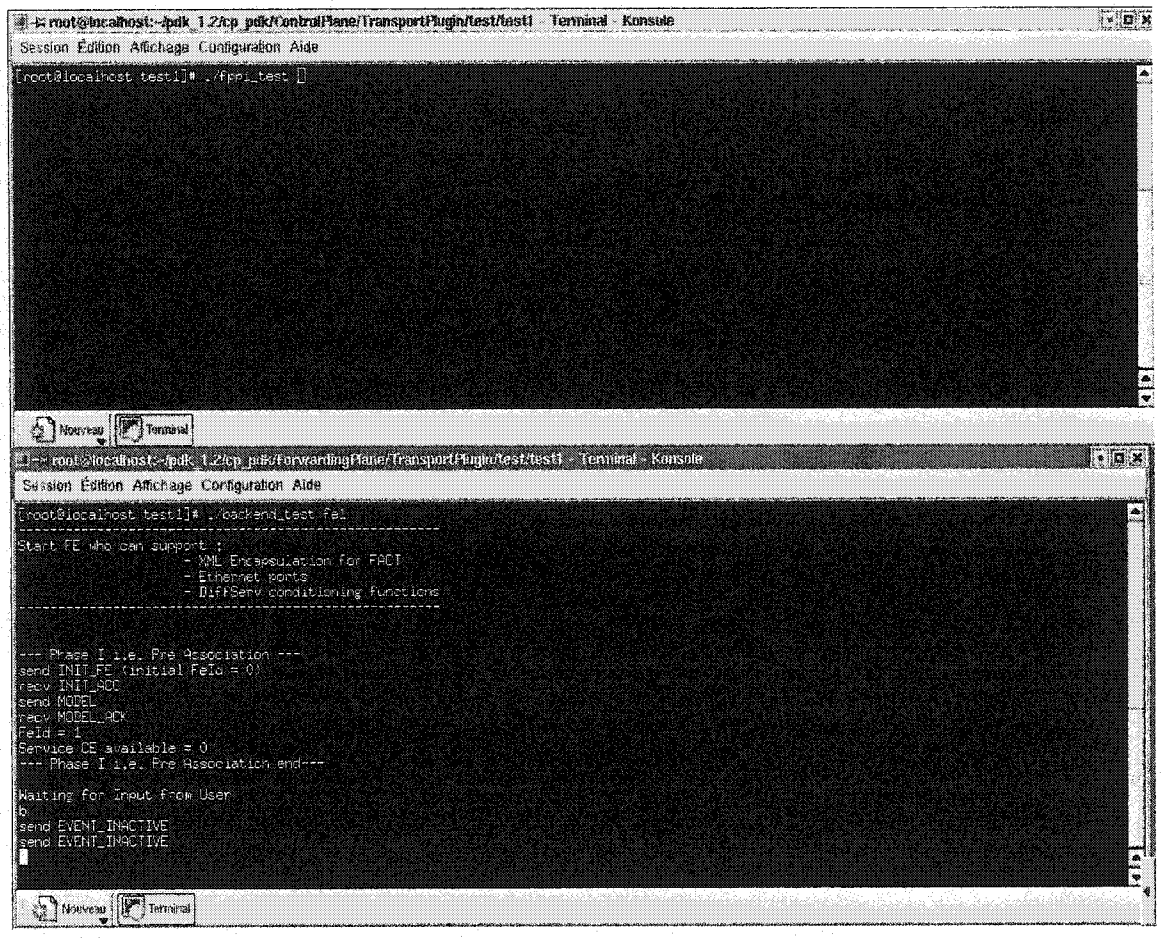


Figure 4.11 Comportement FE avant la connexion du CE

Ensuite, la Figure 4.12 illustre le comportement de notre architecture après l'étape 4 qui correspond à l'entrée en activité du CE. Une fois que le CE a complété sa phase de pré-association, le comportement du Manager va changer vis-à-vis des messages EVENT_INACTIVE du FE. En effet, le Manager va répondre à cette notification, non par un EVENT_INACTIVE_ACK comme précédemment, mais par un MODEL_ACK encapsulant le nom de port TIPC du CE devenu actif. Ainsi, nous pouvons aussi remarquer à la Figure 4.12 que le *Join Request* du FE a bien été reçu et traité par le CE.

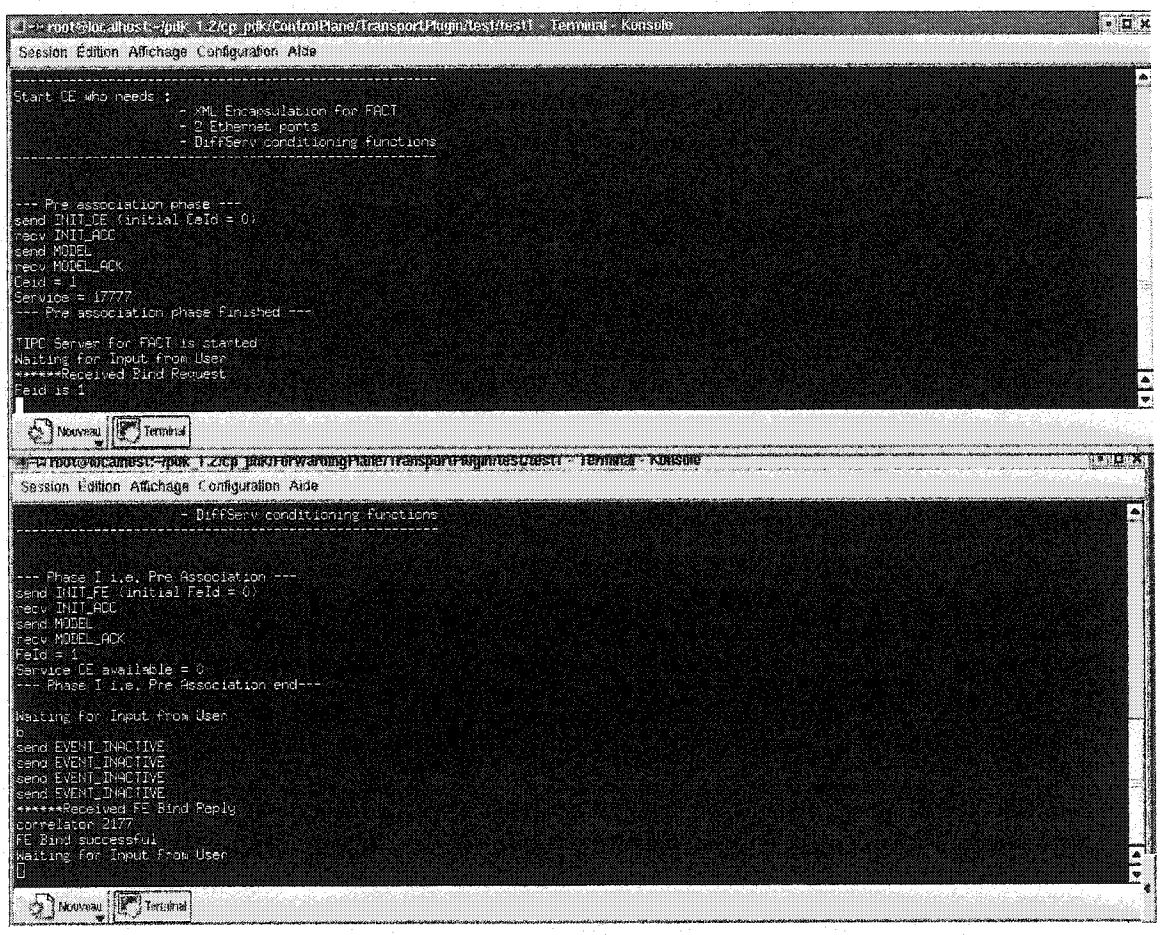


Figure 4.12 Comportement du FE après la connexion du CE

4.5.2 Tests de performance

Maintenant que nous avons pu démontrer la validité de notre architecture vis-à-vis des spécifications que nous nous étions fixées, nous allons dans cette sous-section en étudier les performances.

Cas 1 :

Dans ce cas de test, nous avons repris la structure du test fonctionnel mais simplement avec les CE 1 et 3, le FE 1, et les deux Managers. Le cas de test précédent nous a montré que l'architecture est fonctionnelle mais maintenant il nous faut quantifier le temps mis pour la récupération des connexions.

Tout d'abord, nous avons initié les communications entre CEs et FEs, et avec le Manager. Ensuite, nous coupons la connexion comme dans le test fonctionnel, mais ici nous calculons le temps nécessaire à la reconnexion du CE 3 et du FE 1 au Manager de sauvegarde et pour compléter la nouvelle phase de pré-association. Puis, nous relevons le temps nécessaire pour le FE pour initier un nouveau message de type *Join Request* pour le CE 3 qui est le CE de sauvegarde de CE 1, et recevoir le message de type *Join Response* indiquant que leur connexion est effective.

Le Tableau 4.1 nous indique les résultats relevés au cours de dix essais sur notre architecture pour le CE, alors que le Tableau 4.2 présente les résultats obtenus pour le FE.

Tableau 4.1 Temps de reconnexion pour un CE

CE	Temps moyen (en µs)	Ecart type
Début de la phase de pré-association	96	3
Fin de la phase de pré-association	65543	5487

Tableau 4.2 Temps de reconnexion pour un FE

FE	Temps moyen (en µs)	Ecart type
Début de la phase de pré-association	70	26
Fin de la phase de pré-association	54292	6210
Phase de récupération de la connexion avec un CE	54048	6180

Nous pouvons voir que le temps nécessaire pour conclure la phase de pré-association est de 65 millisecondes pour le CE et de 54 millisecondes pour le FE, et que le temps de récupération de connexion pour le FE avec le CE de sauvegarde est de 54 millisecondes comme illustré à la Figure 4.13.

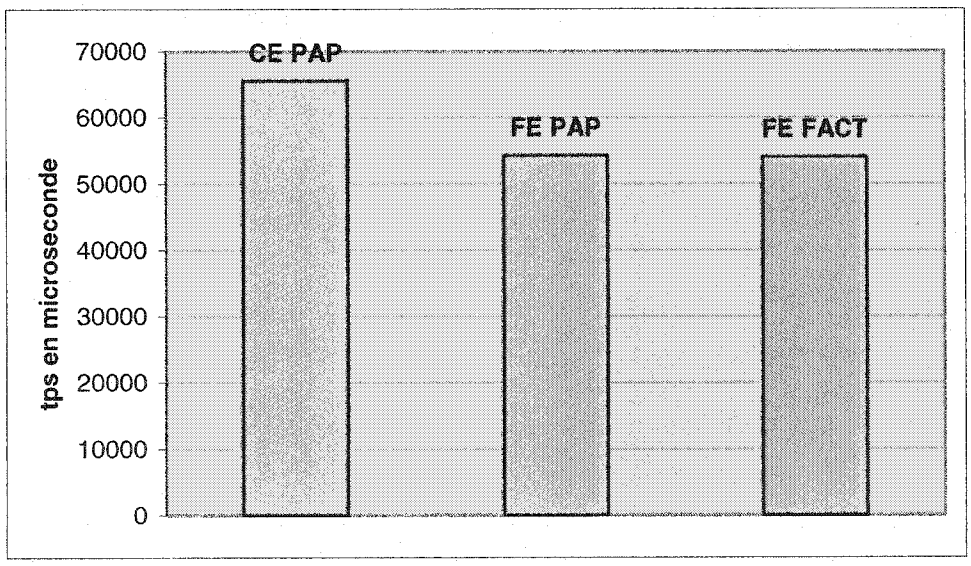


Figure 4.13 Temps de récupération de connexion

Nous pouvons noter que le temps de reconnexion pour le protocole FACT est un temps fixe, alors que le temps pour conclure la phase de pré-association dépend en fait de la taille des documents XML représentant la modélisation de chacun des éléments

mais aussi de la capacité de traitement de la machine sur laquelle est implémenté le Manager.

Cas 2 :

Ce troisième cas de test est orienté vers le poids de la sécurisation des connexions entre CEs et FEs. Pour compléter ce test, nous avons repris le CE 1 et le FE 1 du test fonctionnel. Ensuite, nous avons mesuré le temps nécessaire pour que le CE indique et reçoive un accusé de réception de la création d'une structure *DiffServ* présenté pour le test fonctionnel. Pour cela, nous avons effectué 20 fois cette manipulation, tout d'abord en usant de messages TIPC directement encapsulés dans des trames Ethernet, ensuite dans des datagrammes UDP, et enfin avec UDP et IPSec. Les résultats de cette expérimentation ont été regroupés au Tableau 4.3.

Tableau 4.3 Temps de création d'une structure DiffServ

	Temps moyen (en ms)	Ecart type
TIPC avec Ethernet	12	3
TIPC avec UDP	970	260
TIPC avec UDP et IPSec	1268	263

Tout d'abord, l'analyse de ces résultats nous montre une grande différence de performance entre l'usage de Ethernet et de UDP. Bien que nous n'en voyions pas l'origine exacte, le concepteur de TIPC nous a indiqué que cette différence est due à un dysfonctionnement de l'adaptation UDP de TIPC sous Linux, mais que ce problème ne se présente pas sur d'autres systèmes d'exploitation. D'ailleurs, ce point serait probablement dû à une mauvaise implémentation de la pile de UDP dans le noyau 2.4.x de Linux mais devrait être corrigée avec la version 2.6.x.

Pour ce qui est de la comparaison entre les performances obtenues avec UDP et UDP avec IPSec, nous remarquons une perte de performance de l'ordre de 30%. Ce résultat illustre bien le fait que l'adjonction d'un mécanisme de sécurisation à un coût, mais dans notre cas ce facteur de 30% est tolérable.

Cas 3 :

Pour ce dernier cas de test de performance, nous nous sommes intéressés à l'impact du choix du langage XML pour modéliser les structures de données nécessaires au protocole FACT. Pour cela, nous avons comparé les temps nécessaires à l'envoi et à la réception d'un accusé de réception de 500 messages de type *Configure Logic Component* encapsulant chacun 50 routes pour une table de routage IPv4. Dans les faits, nous avons utilisé ici une seule machine sur laquelle sont implémentés un CE, un FE ainsi qu'un Manager. Les résultats présentés au Tableau 4.4 compare le temps nécessaire pour cette série d'opérations en présentant les données de manière binaire (avec les structures C du NPF API encapsulé directement dans les messages) et sous forme de structure XML.

Tableau 4.4 Performances d'un ajout de route en binaire et en XML

	Temps moyen (en µs)	Moyenne des ajouts de routes par secondes
Encapsulation Binaire	29592	92100
Encapsulation XML	216020	11605

Nous obtenons bien entendu un meilleur ratio d'ajout de route par seconde pour l'encapsulation binaire, comme nous l'avions présagé au chapitre précédent. Tout comme pour l'ajout de fonctionnalité de sécurité, la flexibilité nécessite un compromis sur les performances. Nous pouvons aussi noter que nous avons réalisé un test similaire sur le nombre d'ajouts de route par seconde sous forme de documents XML, mais cette fois sur deux machines toutes deux pourvues de processeurs Pentium IV cadencés à 2 GHz. Or lors de ce test, nous avions un ratio de 42000 ajouts de route par seconde. Ceci nous montre bien la corrélation existant entre les performances du *parser* XML et la capacité de calcul des machines utilisées.

CHAPITRE V
CONCLUSION

Pour conclure notre mémoire, nous allons dans ce chapitre synthétiser notre proposition afin de mieux en comprendre l'impact sur l'architecture proposée par le groupe de travail ForCES ainsi que ses limitations. Enfin, nous indiquerons différents travaux futurs sur ce thème.

5.1 Synthèse de nos contributions

Comme nous avons pu le voir le long de ce mémoire, notre objectif de recherche consiste à apporter une solution évolutive et flexible pour la séparation entre la partie *Contrôle* et la partie *Acheminement.* Pour cela, nous avons essayé de synthétiser les propositions faites au sein de l'IETF mais aussi du NP Forum. L'analyse des requis amenés par le groupe de travail ForCES nous ont amené à nous focaliser sur deux points importants que sont la robustesse et la modélisation.

Ainsi, nous avons proposé l'utilisation combinée du protocole TIPC comme support de communication au sein d'un élément de réseau, et du langage XML pour la modélisation des données des phases de pré et de post-association. Il est important de noter que, dans notre analyse, nous avons fait le choix de nous restreindre à une grappe d'ordinateurs. En effet, ce choix a été motivé par le fait que, dans les implémentations en cours des routeurs basés sur des processeurs réseau, les différents éléments se trouvent justement être des grappes d'ordinateurs.

Le choix de XML s'avère pertinent lorsque l'on considère les évolutions de la séparation entre la partie *Contrôle* et la partie *Acheminement.* L'interopérabilité entre ces deux parties est une préoccupation essentielle pour l'avenir de cette architecture.

Pour valider notre approche, nous avons mis en place un cas simple mais représentatif de l'importance de la flexibilité et de la robustesse exigée par cette séparation. Les résultats obtenus nous indiquent que les fonctionnalités de notre architecture sont concluantes. D'ailleurs, nous avons pu proposer deux documents auprès du groupe de travail ForCES, le premier au sujet de la phase de pré-association [NAIT 2003a] et le second sur la modélisation XML des FEs et CEs [NAIT 2003b]. Ces deux documents ont été bien accueillis par cette communauté qui nous a fourni un grand nombre de commentaires judicieux. Nous avons pu aussi faire une démonstration de XML pour le protocole FACT au 56ème congrès de l'IETF, et aussi une autre du protocole de pré-association et de la modélisation lors du 57ème congrès.

5.2 Limitations de nos propositions

La principale limitation observée lors de notre étude est la perte de performance lors de l'usage du langage XML pour encoder les structures de données du protocole FACT. Nous devons tout d'abord noter que les performances des *parsers* XML sont fortement couplées aux capacités de calculs des machines, mais aussi que le choix de la flexibilité implique une baisse de performance. D'autre part, nous pouvons voir que dans l'architecture du groupe de travail ForCES, le cadre des responsabilités des CE et FE Managers se borne juste à proposer des listes d'adresses aux PEs. Dans notre travail, nous avons vu que leurs responsabilités sont plus grandes car ils sont les seules entités logiques à posséder une vision globale de l'élément de réseau et peuvent le configurer de manière optimale. Or, les connexions FACT sont laissées à la responsabilité des CEs qui peuvent les accepter ou les refuser. Ce constat nous amène à une autre limitation de notre architecture. Il faudrait définir de manière plus précise les responsabilités de Managers dans l'architecture ForCES pour assurer une meilleure cohésion du protocole de pré-association avec le reste des concepts.

5.3 Indications de travaux futurs

Notre étude des limitations nous amène enfin à nous tourner vers l'avenir et déceler les futurs travaux qui pourraient améliorer notre proposition. Au-delà de notre remarque sur les responsabilités des Managers qui, dans les faits, est liée à la vision et aux décisions de l'IETF, nous pouvons proposer trois points d'amélioration.

Le premier point concerne la formalisation générique des données XML. Nous avons opté lors la formalisation de la syntaxe pour les DTDs comme premier choix de conception. Dans l'avenir, il serait intéressant de s'orienter vers un outil plus puissant qu'est le Schéma XML pour garantir plus de précision dans notre modélisation.

Autre point concernant la modélisation et le langage XML, il serait important de faire évoluer notre représentation du *FE Model* pour qu'il suive les évolutions rapides des concepts que nous trouvons dans le groupe de travail ForCES. Nous pouvons aussi noter que ce type de recherche peut aussi s'appliquer aux données utilisées par le protocole FACT avec XML. En effet, de nouveaux APIs tels que ceux concernant la mobilité IP commencent à émerger et il serait important de faire évoluer nos documents XML pour qu'il puisse en prendre compte par le biais de nouvelles structures. Parallèlement à cela, il serait important de s'atteler à l'étude du type de statistiques dont les CEs peuvent avoir besoin, et à leur modélisation en XML.

Finalement, il serait très pertinent de pousser plus loin notre implémentation en l'intégrant au sein d'un véritable prototype de routeur. L'étude de performance qui en découlerait permettrait de connaître de manière plus réaliste l'impact du choix de XML pour le protocole FACT. En effet, le développement sur les processeurs réseau a déjà pu montrer que les mécanismes de vérification de l'intégrité des données de chaque interaction entre CE et FE sont particulièrement dispendieux.

BIBLIOGRAPHIE

[ADIL 2002] M. Adiletta, M. Rosenbluth, D. Bernstein, G. Wolrich, H. Wilkinson, "The Next Generation of Intel© IXP Network Processors", *Intel© Technology Journal*, Vol. 6, No. 3, Août 2002, pp. 6-18.

[AUDU 2003] A. Audu, R. Gopal, *ForwArding and Control ElemenT protocol (FACT)*, draft-gopal-forces-fact-04.txt, Juin 2003.

[BAKE 1995] F. Baker, *Requirements for IP Version 4 Routers*, RFC 1812, Juin 1995.

[BAKE 2002] F. Baker, K. Chan, A. Smith, *Management Information Base for the Differentiated Services Architecture*, RFC 3289, Mai 2002.

[BLAK 1998] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, *An Architecture for Differentiated Services*, RFC 2475, Décembre 1998.

[BRAD 1994] R. Braden, D. Clark, S. Shenker, *Integrated Services in the Internet Architecture: an Overview*, RFC 1633, Juin 1994.

[BRAD 1997] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, *Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification*, RFC 2205, Septembre 1997.

[BRAY 2000] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, "Extensible Markup Language (XML) 1.0 (Second Edition)", http://www.w3.org/TR/REC-xml, Octobre 2000.

[CHAN 2001a] B. Chang, al, "Document Object Model (DOM) Requirements", http://www.w3.org/TR/DOM-Requirements, Avril 2001.

[CHAN 2001b] K. Chan, al, *COPS Usage for Policy Provisioning (COPS-PR)*, RFC 3084, Mars 2001.

[DECA 2000] D. Decasper, Z. Dittia, G. Parulkar, B. Plattner, "Router Plugins: A Software Architecture for Next-Generation Routers", *IEEE/ACM Transactions on Networking*, Vol. 8, No. 1, Février 2000, pp. 2-15.

[FREE 2003] FreeS/WAN, http://www.freeswan.org, consulté en Août 2003.

[HEDR 1988] C. Hedrick, *Routing Information Protocol*, RFC 1058, Juin 1988.

[HJAL 2000] G. Hjalmtysson, "The Pronto Platform: A Flexible Toolkit for Programming Networks using a Commodity Operating System", *OPENARCH 2000*, Mars 2000, Tel-Aviv, Israel, pp. 98-107.

[KENT 1998a] S. Kent, R. Atkinson, *Security Architecture for the Internet Protocol*, RFC 2402, Novembre 1998.

[KENT 1998b] S. Kent, R. Atkinson, *IP Authentication Header*, RFC 2402, Novembre 1998.

[KENT 1998c] S. Kent, R. Atkinson, *IP Encapsulating Security Payload*, RFC 2406, Novembre 1998.

[KHOS 2003] H. Khosravi, T. Anderson, *Requirements for Separation of IP Control and Forwarding*, draft-ietf-forces-requirements-10.txt, Juillet 2003.

[MALK 1998] G. Malkin, *RIP Version 2*, RFC 2453, Novembre 1998.

[MERU 2000] S. Merugu, S. Bhattacharjee, E. Zegura, K. Calvert, "Bowman: A Node OS for Active Networks", *IEEE INFOCOM 2000*, Mars 2000, Tel Aviv, Israel, Vol. 3, pp. 1127-1136.

[MOY 1998] J. Moy, *OSPF Version 2*, RFC 2328, Avril 1998.

[NAIT 2003a] R. Nait Takourout, *ForCES Pre Association Phase Protocol (PAP)*, draft-nait-forces-pap-00.txt, Juin 2003.

[NAIT 2003b] R. Nait Takourout, *ForCES XML based Model for Control and Forward Element Separation*, draft-nait-forces-xml-model-00.txt, Juin 2003.

[NPF 2003] Network Processing Forum, http://www.npforum.org, consulté en Août 2003

[REKH 1995] Y. Rekhter, T. Li, *A Border Gateway Protocol 4 (BGP-4)*, RFC 1771, Mars 1995.

[ROSE 2001] E. Rosen, A. Viswanathan, R. Callon, *Multiprotocol Label Switching Architecture*, RFC 3031, Janvier 2001.

[SPAL 2001] T. Spalink, S. Karlin, L. Peterson, Y. Gottlieb, "Building a Robust Software-Based Router Using Network Processors", *ACM SIGOPS Operating Systems Review*, Banff, Canada, Vol. 35, No. 5, Octobre 2001, pp. 216-229.

[TEJA 2002] Teja Technologies Inc., "Intel© IXP 1200 Integrated Router Application", Février 2002.

[TIPC 2003] TIPC, http://tipc.sourceforge.net, consulté en Août 2003.

[YAN 2001] D. K. Y. Yan, X. Chen, "Resource Management in Software-Programmable Router Operating Systems", *IEEE Journal on selected areas in communications*, Vol. 19, No. 3, Mars 2001, pp. 488-500.

[YANG 2003a] L. Yang, R. Dantu, T. Anderson, R. Gopal, *ForCES Architectural Framework*, draft-ietf-forces-framework-06.txt, Juin 2003.

[YANG 2003b] L.Yang, J. Halpern, R. Gopal, A. DeKok, *ForCES Forwarding Element Functional Model*, draft-yang-forces-model-02.txt, Juin 2003.

ANNEXE A

Spécification du CE

```
<!ELEMENT CESpecification (CEId*, MaxFE, EncapType, FunctionList)>

<!ELEMENT CEId (#PCDATA)>
<!ELEMENT MaxFE (#PCDATA)>
<!ELEMENT EncapType (#PCDATA)>

<!-- Number of FE that the CE could support-->
<!ELEMENT FunctionList (Function+)>
<!ATTLIST FunctionList size CDATA>
<!ELEMENT Function (Port|Forward|QoS|VendorSpecific|Encap|Security)>

<!ELEMENT Port (PortNb, PortTypeList)>
<!ELEMENT PortNb (#PCDATA)>
<!-- Approxymation of the port number needed -->
<!ELEMENT PortTypeList (PortType+)>
<!ATTLIST PortTypeList size CDATA>
<!ELEMENT PortType (#PCDATA)>
<!-- ATM, Ethernet, and so on -->

<!ELEMENT Forward (L3TypeList)>
<!ELEMENT L3TypeList (L3Type+)>
<!ATTLIST L3TypeList size CDATA>
<!ELEMENT L3Type (IpType|OtherL3Type)>
<!ELEMENT OtherL3Type EMPTY>
<!ELEMENT IpType (IpVersion, IpCast)>
<!ELEMENT IpVersion (#PCDATA)>
<!ELEMENT IpCast (#PCDATA)>
<!-- Unicast or Multicast -->

<!ELEMENT QoS (QosTypeList)>
<!ELEMENT QosTypeList (QoSType+)>
<!ATTLIST QosTypeList size CDATA>
<!ELEMENT QoSType (#PCDATA)>
<!-- Could be IntServ, DiffServ -->

<!ELEMENT VendorSpecific (#PCDATA)>
<!-- Another dtd could be defined -->
```

```
<!ELEMENT Encap (EncapTypeList)>
<!ELEMENT EncapTypeList (EncapType+)>
<!ATTLIST EncapTypeList size CDATA>
<!ELEMENT EncapType (#PCDATA)>

<!ELEMENT Security (SecTypeList)>
<!ELEMENT SecTypeList (SecType+)>
<!ATTLIST SecTypeList size CDATA>
<!ELEMENT SecType (#PCDATA)>
<!-- Could be IPSec, etc -->
```

Modèle du FE

```
<!ELEMENT FEModel (FEId*, EncapType , CapabilityModel, StateModel)>

<!ELEMENT EncapType (#PCDATA)>
<!ELEMENT FEId (#PCDATA)>
<!ELEMENT CapabilityModel (LB+)>
<!ATTLIST CapabilityModel size CDATA #REQUIRED>

<!ELEMENT LB (Handle, Type, StatType*)>
<!ELEMENT Handle (#PCDATA)>
<!ELEMENT Type (Interface
      | IpPrefixForwarder| IpNHForwarder| IptoL2
      | DS_Classifier| DS_Meter| DS_Action| DS_Queue| DS_Scheduler| DS_Shaper
)>
<!ELEMENT StatType (#PCDATA)>
<!-- To clarify... -->

<!ELEMENT StateModel (Link+)>
<!ATTLIST StateModel size CDATA #REQUIRED>

<!ELEMENT Link (DownLBHandle?, UpLBHandle?) >
<!ELEMENT DownLBHandle (#PCDATA)>
<!ELEMENT UpLBHandle (#PCDATA)>

<!--
      Unicast IP Forwarder (Prefix and NH Table separation)
-->
<!ELEMENT IpPrefixForwarder (Version, MaxRoute, MaxTable)>
<!ELEMENT Version (#PCDATA)>
```

```
<!-- Could be v4 or v6 -->
<!ELEMENT MaxRoute (#PCDATA)>
<!-- Maximun route managed -->
<!ELEMENT MaxTable (#PCDATA)>
<!-- Maximun number of IpPrefix table -->

<!ELEMENT IpNHForwarder (Version, MaxRoute, MaxTable)>

<!--
      Layer 2 Manager (IP to L2 mapping)
-->
<!ELEMENT IptoL2 (Version, MaxEntry)>
<!ELEMENT MaxEntry (#PCDATA)>
<!-- Maximun number of entries -->

<!--
      Interface
-->
<!ELEMENT Interface (IntfType)>
<!ELEMENT IntfType (IfType, Mtu, Speed)>
<!ELEMENT IfType (#PCDATA)>
<!-- Could be (like 1.3.6.1.2.1.2.2.1.3 MIB)     other(1), regular1822(2), hdh1822(3),
ddn-x25(4), rfc877-x25(5), ethernet-csmacd(6), iso88023-csmacd(7), iso88024-
tokenBus(8), iso88025-tokenRing(9), iso88026-man(10), starLan(11),proteon-
10Mbit(12), proteon-80Mbit(13), hyperchannel(14), fddi(15), lapb(16), sdlc(17),
ds1(18),e1(19), basicISDN(20), primaryISDN(21), propPointToPointSerial(22),
ppp(23), softwareLoopback(24), eon(25), ethernet-3Mbit(26), nsip(27), slip(28),
ultra(29), ds3(30), sip(31), frame-relay(32)-->
<!ELEMENT Mtu (#PCDATA)>
<!ELEMENT Speed (#PCDATA)>

<!--
      DiffServ
-->
<!ELEMENT DS_Classifier (ClfrType)>
<!ELEMENT ClfrType (Dir, ClfrCap)>
<!ELEMENT Dir (#PCDATA)>
<!-- Could be Ingress or Egress -->
<!ELEMENT ClfrCap (#PCDATA)>
<!-- Could be v4srcAddr, v4dstAddr, Proto, dscp, L4, flowid, v6srcAddr, v6dstAddr -->


<!ELEMENT DS_Meter (MeterType)>
<!ELEMENT MeterType (Dir, MeterCap, FailIntfIdList)>
```

```
<!ELEMENT MeterCap (#PCDATA)>
<!-- Could be simpleTB, avgRate, SrTCMBlind, SrTCMAware, TrTCMBlind,
TrTCMAware, TswTCM -->
<!ELEMENT FailIntfIdList (IntfId+)>
<!ATTLIST FailIntfIdList size CDATA #REQUIRED>
<!-- Identifier for LB that could follow on failure (checking after parsing TBD) -->

<!ELEMENT DS_Action (ActionType)>
<!ELEMENT ActionType (Dir, (Counter | Marker | AlgoDropper))>
<!ELEMENT Counter EMPTY>
<!-- To Clarify... -->
<!ELEMENT Marker (#PCDATA)>
<!-- Could be Dscp, Mpls, ... TBD -->
<!ELEMENT AlgoDropper (DropperType, mQCount)>
<!ELEMENT DropperType (#PCDATA)>
<!-- Could be tailDrop, headDrop, randomDrop, alwaysDrop or mQDrop -->
<!ELEMENT mQCount (#PCDATA)>
<!-- Number of queues measured -->

<!ELEMENT DS_Queue (QueueType)>
<!ELEMENT QueueType (Dir, MinQSize, MaxQSize, TotalQSize)>
<!ELEMENT MinQSize (#PCDATA)>
<!-- Minimum configuration size for a queue -->
<!ELEMENT MaxQSize (#PCDATA)>
<!-- Maximum configuration size for a queue -->
<!ELEMENT TotalQSize (#PCDATA)>
<!-- Total available size for a queue -->

<!ELEMENT DS_Scheduler (SchedType)>
<!ELEMENT SchedType (Dir, SchedMethod, MaxInputs, MinRate*, MaxRate*)>
<!ELEMENT SchedMethod (#PCDATA)>
<!-- Could be Priority, WRR, WFQ or DRR -->
<!ELEMENT MaxInputs (#PCDATA)>
<!-- Maximum number of input queues -->
<!ELEMENT MinRate (#PCDATA)>
<!-- Minimum rate of handling inputs -->
<!ELEMENT MaxRate (#PCDATA)>
<!-- Maximum rate of handling inputs -->

<!ELEMENT DS_Shaper (ShaperType)>
<!-- Not really needed with rated Scheduler and Queue-->
<!ELEMENT ShaperType (Dir, MaxLevels)>
<!ELEMENT MaxLevels (#PCDATA)>
<!-- Maximum number of levels -->
```

ANNEXE B

Exemple de modélisation d'un FE

```
<FEModel>
    <EncapType>XML</ EncapType >
    <CapabilityModel>
        <LBList size="9">
        <LB>
            <Handle>1</Handle>
            <Type>
                <Interface>
                    <IntfType>
                        <IfType>ethernet-csmacd</IfType>
                        <Mtu>1500</Mtu>
                        <Speed>100</Speed>
                    </IntfType>
                </Interface>
            </Type>
        </LB>
        <LB>
            <Handle>2</Handle>
            <Type>
                <Interface>
                    <IntfType>
                        <IfType>frame-relay</IfType>
                        <Mtu>1500</Mtu>
                        <Speed>100</Speed>
                    </IntfType>
                </Interface>
            </Type>
        </LB>
        <LB>
            <Handle>3</Handle>
            <Type>
                <DS_Classifier>
                    <ClfrType>
                        <Dir>In</Dir>
                        <ClfrCap>dscp</ClfrCap>
                    </ClfrType>
                </DS_Classifier>
            </Type>
        </LB>
        <LB>
            <Handle>4</Handle>
            <Type>
                <DS_Meter>
                    <MeterType>
```

```
                    <Dir>In</Dir>
                    <MeterCap>simpleTB</MeterCap>
                    <HandleFail>5</HandleFail>
                </MeterType>
            </DS_Meter>
        </Type>
    </LB>
    <LB>
        <Handle>6</Handle>
        <Type>
            <DS_Action>
                <ActionType>
                    <Dir>In</Dir>
                    <AlgoDropper>
                <DropperType>alwaysDrop</DropperType>
                    </AlgoDropper>
                </ActionType>
            </DS_Action>
        </Type>
    </LB>
    <LB>
        <Handle>7</Handle>
        <Type>
            <DS_Queue>
                <QueueType>
                    <Dir>In</Dir>
                    <MinQSize>0</MinQSize>
                    <MaxQSize>15000</MaxQSize>
                    <TotalQSize>15000</TotalQSize>
                </QueueType>
            </DS_Queue>
        </Type>
    </LB>
    <LB>
        <Handle>8</Handle>
        <Type>
            <DS_Action>
                <ActionType>
                    <Dir>In</Dir>
                    <Marker>dscp</Marker>
                </ActionType>
            </DS_Action>
        </Type>
    </LB>
    <LB>
        <Handle>9</Handle>
        <Type>
            <DS_Queue>
                <QueueType>
                    <Dir>In</Dir>
```

```
                        <MinQSize>0</MinQSize>
                        <MaxQSize>15000</MaxQSize>
                        <TotalQSize>15000</TotalQSize>
                    </QueueType>
                </DS_Queue>
            </Type>
        </LB>
        <LB>
            <Handle>10</Handle>
            <Type>
                <DS_Scheduler>
                    <SchedType>
                        <Dir>In</Dir>
                        <SchedMethod>WFQ</SchedMethod>
                        <MaxInputs>3</MaxInputs>
                    </SchedType>
                </DS_Scheduler>
            </Type>
        </LB>
        </LBList>
</CapabilityModel>
 <StateModel>
        <LinkList size="9">
        <Link>
            <DownLBHandle>1</DownLBHandle>
            <UpLBHandle>3</UpLBHandle>
        </Link>
        <Link>
            <DownLBHandle>3</DownLBHandle>
            <UpLBHandle>4</UpLBHandle>
            <UpLBHandle>8</UpLBHandle>
        </Link>
        <Link>
            <DownLBHandle>4</DownLBHandle>
            <UpLBHandle>7</UpLBHandle>
        </Link>
        <Link>
            <DownLBHandle>5</DownLBHandle>
            <UpLBHandle>6</UpLBHandle>
        </Link>
        <Link>
            <DownLBHandle>8</DownLBHandle>
            <UpLBHandle>9</UpLBHandle>
        </Link>
        <Link>
            <DownLBHandle>7</DownLBHandle>
            <DownLBHandle>9</DownLBHandle>
            <UpLBHandle>10</UpLBHandle>
        </Link>
        <Link>
```

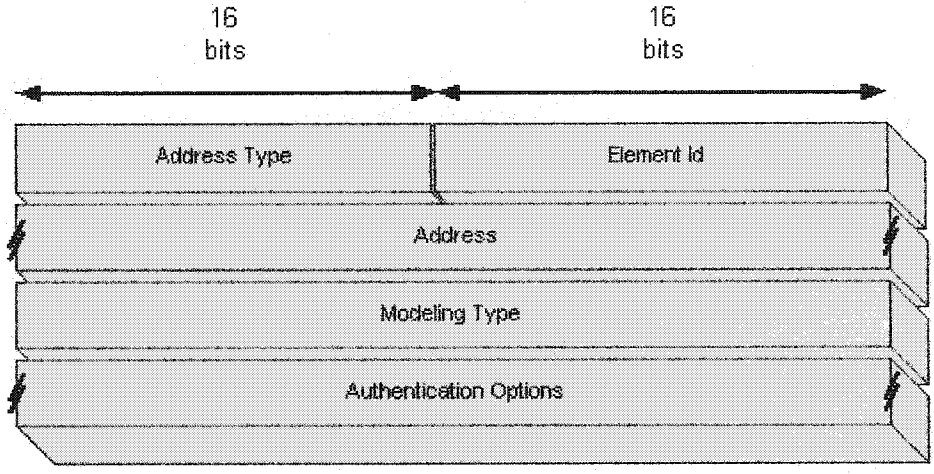
```
                <DownLBHandle>10</DownLBHandle>
                <UpLBHandle>2</UpLBHandle>
            </Link>
            </LinkList>
        </StateModel>
</FEModel>
```

ANNEXE C

Encapsulation des données pour le protocole de pré-association

Pour chacun des types de messages, nous allons dans cette annexe décrire et illustrer leurs champs Value. Dans les illustrations, un champ est représenté avec deux traits sur les cotés lorsque sa longueur dépend d'autres paramètres. Par exemple, un champ d'adressage peut être dimensionné pour une adresse IPv4 ou bien IPv6.

Message INIT_CE :



Adress Type : Champ de 16 bits indiquant le type d'adresse.

Element Id : Identifiant d'un CE.

Address : Adresse du CE.

Modeling Type : Ce champ de 32 bits indiquant le type de modélisation choisi pour le *CE Specification* (XML, OIDs).

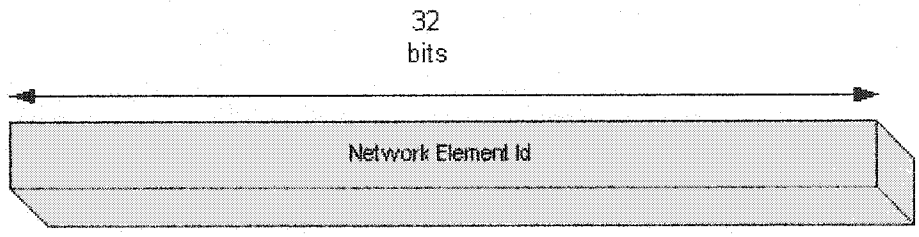
Message INIT_FE :

Les données encapsulé pour dans ce type de message sont les mêmes que pour INI_CE, à la différence que c'est l'identifiant du FE qui est contenue dans le champ Element Id.

Message INIT_MANAGER :

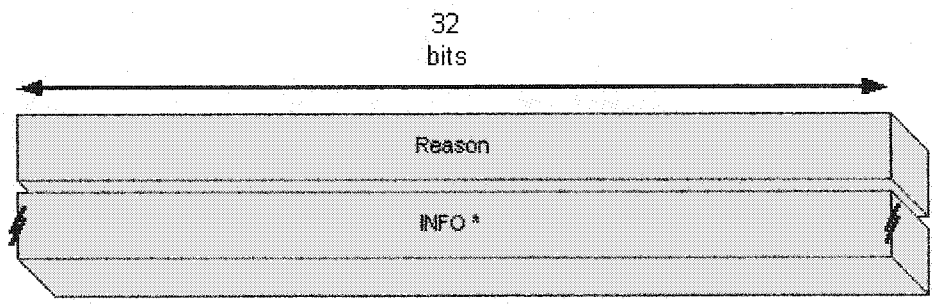
Les données encapsulé pour dans ce type de message sont les mêmes que pour les messages INIT_CE.

Message INIT_ACC :



Network Element Id : Ce champ de 32 bits décrit l'identifiant de l'élément de réseaux.

Message INIT_REJ :

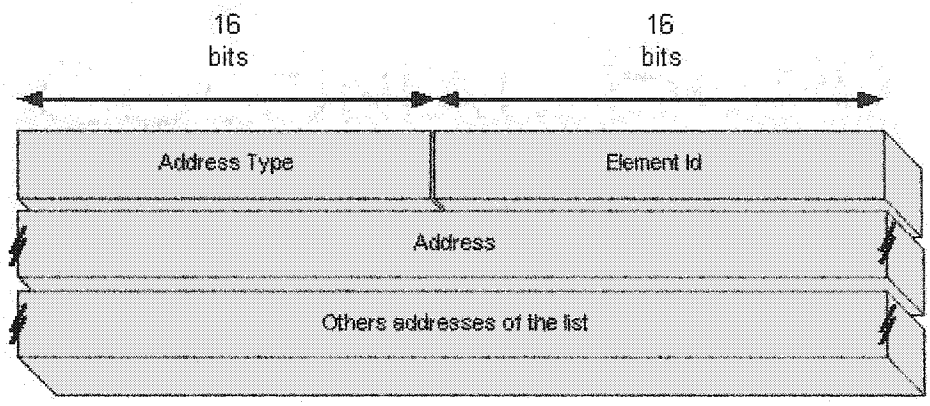


Reason : Ce champ indique la raison du refus du Manager (erreurs d'authentification, élément de réseau occupé, représentation de modélisation non supporté)
INFO : Ce champ optionnel précisant sous forme de chaîne de caractère la raison du refus.

Message MODEL :

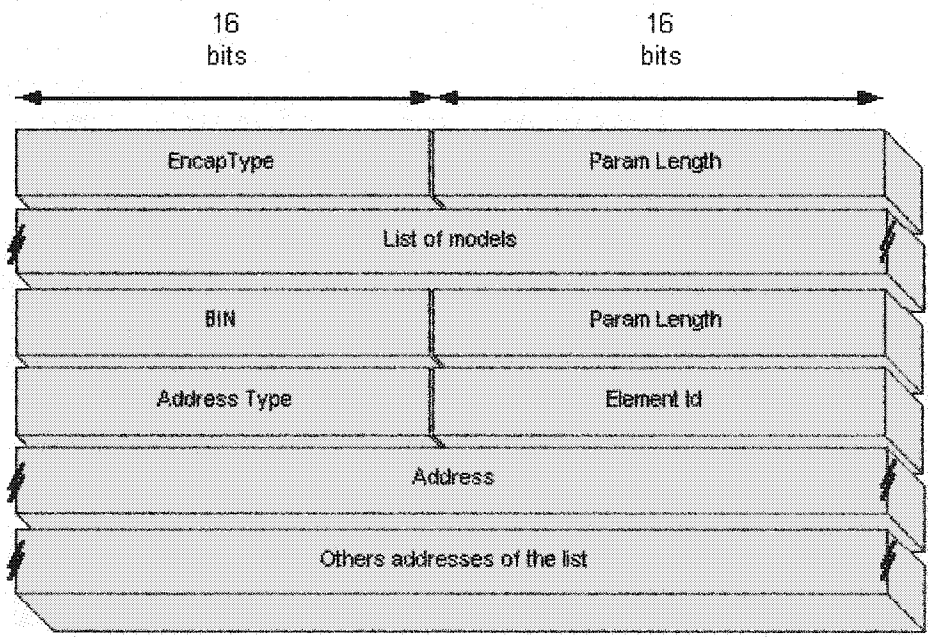
Ce type de message contient simplement dans son champ Value la modélisation envoyée.

Message MODEL_ACK :



Les champs ici présents ont la même signification que ceux du message INIT_CE.

Message LIST :



La charge utile du type de messages LIST contient : une liste de modélisation et une liste d'adresses. Ces deux types d'information sont représentés par deux paramètres.

Les valeurs des champs de ce type de messages possèdent les mêmes significations que pour les messages MODEL et MODEL_ACK.

Message LIST_ACK :

Ce type de message est un simple accusé de réception et ne possède pas de charge utile.

Message EVENT_INACTIVE :

Le format de ce type de message est identique au message INIT_REJ, mais le champ Reason, contient la raison de l'inactivité (l'élément est incapable de joindre ses correspondants, l'élément est occupé, etc.).

Message EVENT_INACTIVE_ACK :

Ce type de message ne possède pas de charge utile.

Message EVENT_LEAVE :

Le format de ce type de message est identique au message INIT_REJ, mais le champ Reason, contient la raison du départ qui, est en fait, ceux utilisé pour les messages Leave Request du protocole FACT.

Message EVENT_INACTIVE_ACK :

Le format de ce type de message est identique au message EVENT_LEAVE

ANNEXE D

DTD des messages FACT du type '*Configure Logic Components*'

```
<!-- IP Forwarder -->
<!ELEMENT RouteList (Route+)>
<!ATTLIST RouteList vers CDATA #REQUIRED size CDATA #REQUIRED>
<!ELEMENT Route (IPAddr, Mask, NHRouteList)>
<!ELEMENT IPAddr (#PCDATA)>
<!ELEMENT Mask (#PCDATA)>
<!ELEMENT NHRouteList (NHRoute+)>
<!ATTLIST NHRouteList size CDATA #REQUIRED>
<!ELEMENT NHRoute (IPAddr, EgrInf, Metric)>
<!ELEMENT EgrInf (#PCDATA)>
<!ELEMENT Metric (#PCDATA)>

<!ELEMENT PrefixList (Prefix+)>
<!ATTLIST PrefixList
        vers CDATA #REQUIRED
        size CDATA #REQUIRED>
<!ELEMENT Prefix (IPAddr, PrefLen, NHIndex)>
<!ELEMENT PrefLen (#PCDATA)>
<!ELEMENT NHIndex (#PCDATA)>

<!ELEMENT NHList (NHIndex, NH+)>
<!ATTLIST NHList
        vers CDATA #REQUIRED
        size CDATA #REQUIRED>

<!ELEMENT NH (FEId*, L2Id*, Weight, EgrInf, IPAddr)>
<!ATTLIST NH type CDATA #REQUIRED >
<!ELEMENT FEId (#PCDATA)>
<!ELEMENT L2Id (#PCDATA)>
<!ELEMENT Weight (#PCDATA)>
<!ELEMENT EgrInf (#PCDATA)>

<!ELEMENT L2EntryList (L2Entry+)>
<!ATTLIST L2EntryList
        size CDATA #REQUIRED
        vers CDATA #REQUIRED >
<!ELEMENT L2Entry (IPAddr, IfHandle, MediaAddr)>
```

```
<!ELEMENT IfHandle (#PCDATA)>
<!ELEMENT MediaAddr (#PCDATA)>
<!ATTLIST MediaAddr type CDATA #REQUIRED >
<!-- Specific to IPv4 => Must be clarified -->

<!-- DiffServ Classifier -->
<!ELEMENT DS_Filter (DS_Id, FilterEntry)

<!ELEMENT DS_Id (
      PortId, Dir?,
      TableId, InstId,
      NextTableId, NextInstId)>
<!ELEMENT PortId (#PCDATA)>
<!ELEMENT Dir (#PCDATA)>
<!ELEMENT TableId (#PCDATA)>
<!ELEMENT InstId (#PCDATA)>
<!ELEMENT NextTableId (#PCDATA)>
<!ELEMENT NextInstId (#PCDATA)>

<!ELEMENT FilterEntry (
      Precedence, Negation,
      DstAddr, DstAddrMask,
      SrcAddr, SrcAddrMask,
      Dscp, Protocol, flowid?,
      DstL4PortMin, DstL4PortMax,
      SrcL4PortMin, SrcL4PortMax)>
<!ATTLIST FilterEntry AddrType CDATA #REQUIRED>
<!ELEMENT Precedence (#PCDATA)>
<!ELEMENT Negation (#PCDATA)>
<!ELEMENT DstAddr (#PCDATA)>
<!ELEMENT DstAddrMask (#PCDATA)>
<!ELEMENT SrcAddr (#PCDATA)>
<!ELEMENT SrcAddrMask (#PCDATA)>
<!ELEMENT Dscp (#PCDATA)>
<!ELEMENT Protocol (#PCDATA)>
<!ELEMENT flowid (#PCDATA)>
<!ELEMENT DstL4PortMin (#PCDATA)>
<!ELEMENT DstL4PortMax (#PCDATA)>
<!ELEMENT SrcL4PortMin (#PCDATA)>
<!ELEMENT SrcL4PortMax (#PCDATA)>

<!-- DiffServ Meter -->
<!ELEMENT DS_Meter (DS_Id, FailNextTableId, FailNextInstId, MeterEntry)>
<!ELEMENT FailNextTableId (#PCDATA)>
```

```
<!ELEMENT FailNextInstId (#PCDATA)>
<!ELEMENT MeterEntry (Specific, TB_Param?)>
<!ELEMENT Specific (#PCDATA)>
<!-- Specific Type could be
      TB_Param
-->
<!ELEMENT TB_Param (MeterType, Rate, BurstSize, Interval)>
<!ELEMENT MeterType (#PCDATA)>
<!-- Type could be
      SimpleTB,
AvgRate,
SrTCMBlind,
SrTCMAware,
TrTCMBlind,
TrTCMAware,
TswTCM
-->
<!ELEMENT Rate (#PCDATA)>
<!ELEMENT BurstSize (#PCDATA)>
<!ELEMENT Interval (#PCDATA)>

<!-- DiffServ Scheduler -->
<!ELEMENT DS_Scheduler (DS_Id, SchedulerEntry)>
<!ELEMENT SchedulerEntry (MinRate?, MaxRate*)>
<!ATTLIST SchedulerEntry MRCount CDATA #REQUIRED>

<!ELEMENT MinRate (Priority, Absolute, Relative)>
<!ELEMENT Priority (#PCDATA)>
<!ELEMENT Absolute (#PCDATA)>
<!ELEMENT Relative (#PCDATA)>

<!ELEMENT MaxRate (Level, Absolute, Relative, Threshold)>
<!ELEMENT Level (#PCDATA)>
<!ELEMENT Threshold (#PCDATA)>

<!-- DiffServ Queue -->
<!ELEMENT DS_Queue (DS_Id, QueueEntry)>
<!ELEMENT QueueEntry (MinRate?, MaxRate*)>
<!ATTLIST QueueEntry MRCount CDATA #REQUIRED>

<!-- DiffServ Marker -->
<!ELEMENT DS_Marker (DS_Id, MarkerEntry)>
<!ELEMENT MarkerEntry (MarkerType, Dscp?)>
<!ELEMENT MarkerType (#PCDATA)>
```

```
<!-- DiffServ AlgoDropper -->
<!ELEMENT DS_AlgoDropper (DS_Id, AlgDropEntry+)>
<!ATTLIST DS_AlgoDropper Size CDATA #REQUIRED>
<!ELEMENT AlgDropEntry (AlgType, qMeasureTableId, qMeasureInstId, qThreshold,
AlgSpecific)>
<!ELEMENT AlgType (#PCDATA)>
<!ELEMENT qMeasureTableId (#PCDATA)>
<!ELEMENT qMeasureInstId (#PCDATA)>
<!ELEMENT qThreshold (#PCDATA)>
<!ELEMENT AlgSpecific (SpecType, MinThreshBytes, MinThreshPkts,
MaxThreshBytes, MaxThreshPkts, ProbMax, Weight, SamplingRate)>
<!ELEMENT SpecType (#PCDATA)>
<!ELEMENT MinThreshBytes (#PCDATA)>
<!ELEMENT MinThreshPkts (#PCDATA)>
<!ELEMENT MaxThreshBytes (#PCDATA)>
<!ELEMENT MaxThreshPkts (#PCDATA)>
<!ELEMENT ProbMax (#PCDATA)>
<!ELEMENT Weight (#PCDATA)>
<!ELEMENT SamplingRate (#PCDATA)>

<!-- DiffServ Block delete and purge -->
<!ELEMENT DS_Block (PortId, TableId, InstId*)>


<!ELEMENT L3AttrsList (PortNo, L3Attrs+)>
<!ATTLIST L3AttrsList size CDATA #REQUIRED>
<!ELEMENT L3Attrs(PortNo,(L3Attrs_All|L3Attrs_IPAddr|L3Attrs_IPFwd))>
<!ELEMENT PortNo (#PCDATA) >
<!ELEMENT L3Attrs_IPAddr (IPAddr, Mask) >
<!ELEMENT L3Attrs_IPFwd (IPFwding) >
<!ELEMENT IPFwding (#PCDATA) >
<!ELEMENT L3Attrs_All (IPAddr, Mask, IPFwding) >

<!ELEMENT L2AttrsList (L2Attrs+)>
<!ATTLIST L2AttrsList size CDATA #REQUIRED>
<!ELEMENT L2Attrs (PortNo,
(L2Attrs_All|L2Attrs_Mtu|L2Attrs_LinkSpeed|L2Attrs_Status))>
<!ELEMENT L2Attrs_Mtu (#PCDATA)>
<!ELEMENT L2Attrs_LinkSpeed (#PCDATA)>
<!ELEMENT L2Attrs_Status (#PCDATA)>
<!ELEMENT L2Attrs_All (L2Attrs_Mtu,L2Attrs_LinkSpeed, L2Attrs_Status)>
```

```
<!ELEMENT IfGeneric (Speed, AdminStatus,
(IPv4_Attr|LAN_Attr|ATM_Attr|POS_Attr))>
<!ATTLIST IfGeneric type CDATA #REQUIRED >
<!ELEMENT Speed (#PCDATA)>
<!ELEMENT AdminStatus (#PCDATA)>
<!ELEMENT IPv4_Attr (IPAddr, Mtu, FIB_Handle)>
<!ELEMENT Mtu (#PCDATA)>
<!ELEMENT FIB_Handle (#PCDATA)>
<!ELEMENT LAN_Attr (Port, Promisc, Speed)>
<!ELEMENT Port (#PCDATA)>
<!ELEMENT Promisc (#PCDATA)>
<!ELEMENT ATM_Attr (Port)>
<!ELEMENT POS_Attr (Port)>
<!---->
```

DTD des messages FACT du type '*Configure Logic Components Response*'

```
<!ELEMENT IPKeyList (Route+)>
<!ATTLIST IPKeyList vers CDATA #REQUIRED size CDATA #REQUIRED>
<!ELEMENT PrefixKey (IPAddr, NetLen, Status)>
<!ELEMENT IPAddr (#PCDATA)>
<!ELEMENT NetLen (#PCDATA)>
<!ELEMENT Status (#PCDATA)>
```

DTD des messages FACT du type '*Query Request*'

```
<!ELEMENT IPAddrList (IPAddr+)>
<!ATTLIST IPAddrList
        vers CDATA #REQUIRED
        size CDATA #REQUIRED >
<!ELEMENT IPAddr (#PCDATA)>

<!ELEMENT MediaAddrList (MediaAddr+)>
<!ATTLIST MediaAddrList
        size CDATA #REQUIRED >
<!ELEMENT MediaAddr (#PCDATA)>
<!ATTLIST MediaAddr type CDATA #REQUIRED >
```

```
<!ELEMENT Properties (PortNo)>
<!ELEMENT PortNo (#PCDATA) >
```

DTD des messages FACT du type '*Query Response*'

```
<!ELEMENT ARPEntryList (ARPEntry+)>
<!ATTLIST ARPEntryList size CDATA #REQUIRED >
<!ELEMENT ARPEntry (IPAddr, MACAddr, EgrInf, Aging)>
<!ELEMENT IPAddr (#PCDATA)>
<!ELEMENT MACAddr (#PCDATA)>
<!ELEMENT EgrInf (#PCDATA)>
<!ELEMENT Aging (#PCDATA)>

<!ELEMENT L2StatsList (L2StatsEntry+)>
<!ATTLIST L2StatsList size CDATA #REQUIRED >
<!ELEMENT L2StatsEntry (
      PortId, dot3StatsAlignmentErrors, dot3StatsFCSErrors,
dot3StatsSingleCollisionFrames,
      dot3StatsMultipleCollisionFrames, dot3StatsSQETestErrors,
dot3StatsDeferredTransmissions,
      dot3StatsLateCollisions, dot3StatsExcessiveCollisions,
dot3StatsInternalMacTransmitErrors,
      dot3StatsCarrierSenseErrors, dot3StatsFrameTooLongs,
dot3StatsInternalMacReceiveErrors,
      dot3StatsSymbolErrors, dot3StatsDuplexStatus)>
<!ELEMENT PortId (#PCDATA)>
<!ELEMENT dot3StatsAlignmentErrors (#PCDATA)>
<!ELEMENT dot3StatsFCSErrors (#PCDATA)>
<!ELEMENT dot3StatsSingleCollisionFrames (#PCDATA)>
<!ELEMENT dot3StatsMultipleCollisionFrames (#PCDATA)>
<!ELEMENT dot3StatsSQETestErrors (#PCDATA)>
<!ELEMENT dot3StatsDeferredTransmissions (#PCDATA)>
<!ELEMENT dot3StatsLateCollisions (#PCDATA)>
<!ELEMENT dot3StatsExcessiveCollisions (#PCDATA)>
<!ELEMENT dot3StatsInternalMacTransmitErrors (#PCDATA)>
<!ELEMENT dot3StatsCarrierSenseErrors (#PCDATA)>
<!ELEMENT dot3StatsFrameTooLongs (#PCDATA)>
<!ELEMENT dot3StatsInternalMacReceiveErrors (#PCDATA)>
<!ELEMENT dot3StatsSymbolErrors (#PCDATA)>
<!ELEMENT dot3StatsDuplexStatus (#PCDATA)>

<!ELEMENT L3StatsList (L3StatsEntry+)>
```

```
<!ELEMENT L3StatsEntry (PortId, ifInOctets, ifInUcastPkts,
      ifInDiscards, ifInErrors, ifInUnknownProtos,
      ifOutOctets, ifOutUcastPkts, ifOutDiscards, ifOutErrors, ifInMulticastPkts,
OutMulticastPkts,
      OutBroadcastPkts, HCInOctets ,HCInUcastPkts ,HCInMulticastPkts
,HCInBroadcastPkts ,HCOutOctets ,     HCOutUcastPkts ,HCOutMulticastPkts
,HCOutBroadcastPkts)>
<!-- Version 64 bits for > 20 Mbps ?? -->
<!ELEMENT ifInOctets (#PCDATA)>
<!ELEMENT ifInUcastPkts (#PCDATA)>
<!ELEMENT ifInDiscards (#PCDATA)>
<!ELEMENT ifInErrors (#PCDATA)>
<!ELEMENT ifInUnknownProtos (#PCDATA)>
<!ELEMENT ifOutOctets (#PCDATA)>
<!ELEMENT ifOutUcastPkts (#PCDATA)>
<!ELEMENT ifOutDiscards (#PCDATA)>
<!ELEMENT ifOutErrors (#PCDATA)>
<!ELEMENT ifInMulticastPkts (#PCDATA)>
<!ELEMENT OutMulticastPkts (#PCDATA)>
<!ELEMENT OutBroadcastPkts (#PCDATA)>
<!ELEMENT HCInOctets (#PCDATA)>
<!ELEMENT HCInUcastPkts (#PCDATA)>
<!ELEMENT HCInMulticastPkts (#PCDATA)>
<!ELEMENT HCInBroadcastPkts (#PCDATA)>
<!ELEMENT HCOutOctets (#PCDATA)>
<!ELEMENT HCOutUcastPkts (#PCDATA)>
<!ELEMENT HCOutMulticastPkts (#PCDATA)>
<!ELEMENT HCOutBroadcastPkts (#PCDATA)>

<!ELEMENT ICMPStatsList (ICMPStatsEntry+)>
<!ELEMENT ICMPStatsEntry (PortId,
      icmpInMsgs, icmpInErrors, icmpInDestUnreach, icmpInTimeExceeds,
icmpInParamProbs,
      icmpInSrcQuenches, icmpInRedirects, icmpInEchos, icmpInEchoReps,
icmpInTimeStamps, icmpInTimeStampsReps, icmpInAddrMasks,
icmpInAddrMaskReps, icmpOutMsgs, icmpOutErrors, icmpOutTimeExceeds,
icmpOutParamProbs,
      icmpOutSrcQuenches, icmpOutRedirects, icmpOutEchos, icmpOutEchoReps,
icmpOutTimeStamps, icmpOutTimeStampsReps,
      icmpOutAddrMasks, icmpOutAddrMaskReps)>
<!ELEMENT icmpInMsgs (#PCDATA)>
<!ELEMENT icmpInErrors (#PCDATA)>
<!ELEMENT icmpInDestUnreach (#PCDATA)>
<!ELEMENT icmpInTimeExceeds (#PCDATA)>
```

<!ELEMENT icmpInParamProbs (#PCDATA)>
<!ELEMENT icmpInSrcQuenches (#PCDATA)>
<!ELEMENT icmpInRedirects (#PCDATA)>
<!ELEMENT icmpInEchos (#PCDATA)>
<!ELEMENT icmpInTimeStamps (#PCDATA)>
<!ELEMENT icmpInTimeStampsReps (#PCDATA)>
<!ELEMENT icmpInAddrMasks (#PCDATA)>
<!ELEMENT icmpInAddrMaskReps (#PCDATA)>
<!ELEMENT icmpOutMsgs (#PCDATA)>
<!ELEMENT icmpOutErrors (#PCDATA)>
<!ELEMENT icmpOutDestUnreach (#PCDATA)>
<!ELEMENT icmpOutTimeExceeds (#PCDATA)>
<!ELEMENT icmpOutParamProbs (#PCDATA)>
<!ELEMENT icmpOutSrcQuenches (#PCDATA)>
<!ELEMENT icmpOutRedirects (#PCDATA)>
<!ELEMENT icmpOutEchos (#PCDATA)>
<!ELEMENT icmpOutTimeStamps (#PCDATA)>
<!ELEMENT icmpOutTimeStampsReps (#PCDATA)>
<!ELEMENT icmpOutAddrMasks (#PCDATA)>
<!ELEMENT icmpOutAddrMaskReps (#PCDATA)>
*

<!ELEMENT DS_ClfrStatsEntry (defaultPackets, defaultBytes, totalPackets, totalBytes)>
<!ELEMENT defaultPackets (#PCDATA)>
<!ELEMENT defaultBytes (#PCDATA)>
<!ELEMENT totalPackets (#PCDATA)>
<!ELEMENT totalBytes (#PCDATA)>