Les réseaux programmables 1.0

Olivier Festor, Isabelle Chrisment, Eric Fleury

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Les réseaux programmables 1.0

Olivier FESTOR, Isabelle CHRISMENT, Eric FLEURY

No 3913

Mars 2000

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--3913--FR

Les réseaux programmables 1.0

Olivier Festor, Isabelle Chrisment, Eric Fleury

Thème 1 — Réseaux et systèmes
Projet RESEDAS

Rapport de recherche n 3913 — Mars 2000 — 73 pages

Résumé : Apparu en 1996, le concept de réseau programmable a acquis en 1998 une véritable reconnaissance dans la communauté de recherche en réseaux et télécommunications. De plus, dans une configuration spécifique qui fédère les réseaux programmables, ce concept gagne une reconnaissance industrielle via la création de plusieurs startup et l'implication d'équipementiers dans des processus de standardisation d'interfaces de programmation d'équipements et de réseaux.

Ce rapport a pour objectif de dresser un état de l'art détaillé sur ce domaine en pleine expansion. Sa vocation principale de ce rapport est de fournir un référentiel du domaine aux chercheurs francophones en formalisant les bases et en intégrant régulièrement les innovations du domaine tout en maintenant une bibliographie complète des contributions.

Mots-clé : réseaux actifs, réseaux programmables

(Abstract: pto)

Ce rapport a été initié dans le cadre du projet ANAIS soutenu par les actions Télécom du CNRS. Son extension est soutenue partiellement dans le cadre du projet RNRT AMARRAGE lancé en novembre 1999 pour une durée de 24 mois.

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY (France)
Téléphone : 03 83 59 30 30 - International : +33 3 3 83 59 30 30
Télécopie : 03 83 27 83 19 - International : +33 3 83 27 83 19
Antenne de Metz, technopôle de Metz 2000, 4 rue Marconi, 55070 METZ

Programmable Networks 1.0

Abstract: Initially proposed in 1996, the concept of programmable networks has gained wide acceptance in the networking research community in 1998. Moreover, the subset of this area concerned with programmable networks, has reached an industrial level and several startups as well as equipment providers are now involved in the standardization process of open interfaces.

This report aims at providing a detailled state of the art of this research domain. The main goal of this manuscript is to provide a reference document to the french speaking researchers by formalizing the basis of active networking and by integrating periodically all innovations made in this area as well as maintaining an up-to-date bibliography and field overview.

Key-words: Active Networks, Programmable Networks

Table des matières

Table des figures

1 Introduction

Dans le courant des années 80, les réseaux étaient totalement dédiés à un service unique et spécifique (réseau téléphonique pour la voix, X.25 pour les données, CATV[1] pour la télévision). Ces réseaux ne permettaient pas, sauf dans des proportions très limitées, de déployer d'autres services que ceux pour lesquels ils étaient conçus et déployés.

L'émergence des réseaux large bande a fortement contribué à l'intégration partielle, sur un réseau physique des services de base de transfert de voix et de données. Cependant, le nombre de services offerts reste limité en raison de l'impossibilité d'extension de services dans les composants du réseau. En effet, la plupart des équipements intègrent un ensemble fini de services souvent hardcodés et non extensible.

Le réseau intelligent a marqué une évolution supplémentaire dans le domaine des services en définissant au début des années 90 une infrastructure permettant de coupler le réseau à des bases d'informations afin de traiter de nombreux services qui peuvent être déployés beaucoup plus rapidement que précédemment. Ce type de réseau est cependant totalement orienté vers la téléphonie et les services associés en raison de sa dépendance envers le modèle d'appel utilisé. Celui-ci est bien trop spécifique pour être applicable à d'autres types de réseaux.

Offrir aux opérateurs et fournisseurs de services une réactivité accrue dans les réseaux de transmission de données est aujourd'hui une préoccupation majeure dans la communauté de recherche ainsi que chez les principaux acteurs du domaine. Pour cela, il faut être en mesure de proposer des architectures et technologies ouvertes leur permettant de développer dans des délais records et des conditions technologiques stables de nouvelles offres de services de communication. Ceci leur permettra de mieux répondre aux besoins des usagers.

Ce besoin est principalement dû à la multiplication des services applicatifs (communication de groupe, téléphonie sur IP, Web, ...), à l'explosion du nombre de protocoles de contrôle (MPLS Multi Protocol Label Switching[2], RSVP[3], Differentiated Services[4], GSMP[5], ISUP[6], PNNI[7], ...) dont l'émergence rapide ne permet pas aux constructeurs d'équipements de les intégrer tous dans une échelle de temps adaptée aux besoins. De plus, l'ouverture des marchés au monde concurrentiel pousse les opérateurs et fournisseurs de services à se différencier sur des critères de services, de rapidité de réaction, d'innovation et de prix.

Les réseaux programmables offrent une solution à ce problème en *ouvrant* les équipements et l'ensemble du réseau au travers d'interfaces de programmation et/ou d'interfaces d'accès permettant à des tiers de concevoir, de réaliser et de déployer de nouveaux services sur ces réseaux.

Véritablement lancé en 1995 par l'organisation de *workshops* sur la signalisation ouverte et en 1996 par une proposition de TENNENHOUSE et WETHERALL [TEN 96b] ainsi que par un article de LAZAR sur le besoin de programmation des réseaux de télécommunication [LAZ 97], le concept de réseau programmable existe depuis 1980. À cette date en effet, une expérimentation d'un réseau radio sur lequel les paquets échangés contenaient un programme exécuté sur chaque récepteur avait été menée. Ceci ouvrait la porte au déploiement dynamique de protocoles même si ce n'était pas le but premier dans ce projet. Ce n'est cependant que dans le courant de l'année 1997 que des résultats probants ont abouti à une très large reconnaissance de cette approche autour de laquelle se développe une communauté de plus en plus large. Preuve en est la grande popularité du Workshop OPENARCH sur les réseaux programmables, la publication en 1998 et 1999 de numéros spéciaux d'IEEE Network, d'IEEE Computer, d'IEEE Communications Magazine et de plusieurs articles dans ACM Computer Communications Review sur le sujet. La multiplication des projets de recherche sur ce domaine a également permis l'organisation du premier congrès international sur les réseaux actifs à Berlin en juin 1999.

Aujourd'hui plus de 50 projets de recherche travaillent sur ce thème et regroupeent plus de 80 laboratoires universitaires et industriels à travers le monde. Cet engouement est principalement dû à l'impact potentiel des apports des réseaux programmables sur la conception, le déploiement et les opérations sur les systèmes de communication. Cet apport potentiel est tel que de nombreux industriels et organismes gouvernementaux jugent utile d'investir fortement sur ces approches. Une telle mobilisation universitaire est également justifiée par les problèmes de recherche énormes que soulève cette problématique en ouvrant des pistes depuis la conception de

1. CAble TV
2. `http://www.ietf.org/html.charters/mpls-charter.html`
3. Resource ReSerVation Protocol [BRA 97]
4. `http://www.ietf.org/html.charters/diffserv-charter.html`
5. GSMP : *General Switch Management Protocol [RFC 96].*
6. ISDN User Part
7. Private Network to Node Interface

hardware jusqu'à des problèmes de génie logiciel en passant par des problèmes de langage, de sécurité, de code mobile auquels s'ajoutent bien sûr les problèmes plus traditionnels des télécommunications comme la conception, la validation et le test de protocoles et services. Finalement, la jeunesse du domaine laisse un énorme espace de liberté aux chercheurs. Ceci leur permet d'expérimenter leurs idées sans aucune contrainte normative, ce qui dans le domaine des télécommunications est, il faut bien l'avouer, assez rare.

1.1 Définitions des réseaux programmables

Les réseaux programmables apportent une vision totalement nouvelle des réseaux de télécommunication et de leur opération. Une première définition permet de les situer.

Définition 1 *Un réseau programmable est un réseau de transmission de données ouvert et extensible disposant d'une infrastructure dédiée à l'intégration et à la mise en œuvre rapide de nouveaux services sur l'ensemble de ses composants.*

Cette définition se comprend facilement par opposition aux réseaux traditionnels qui comportent un nombre restreint et fixe de services implantés dans les équipements et qui n'offrent aucun moyen d'en ajouter de nouveaux. En conséquence, ils ne permettent pas de modifier dynamiquement le comportement global du réseau.

L'infrastructure d'un réseau programmable peut être vue comme un réseau de transport de données étendu par un environnement de programmation à l'échelle du réseau comportant un modèle de programmation des services, une architecture de déploiement et d'exploitation pour ces derniers et un contexte d'exécution. Pour atteindre un niveau de programmabilité suffisant, il faut nécessairement abstraire les différents composants d'un réseau et en faire des objets informatiques. Ceci nous permet de donner une seconde définition comme suit :

Définition 2 *Un réseau programmable est un réseau dans lequel toute ou partie de l'infrastructure de communication est virtualisée.*

C'est sur cette virtualisation que reposent tous les concepts et architectures développés dans la communauté réseaux.

Cette définition est certes minimaliste mais universelle. Toutes les recherches menées dans le domaine y sont fidèles. Les principales différences entre les approches résident dans le degré de virtualisation et dans le spectre des rôles autorisés à programmer le réseau (usagers, opérateurs, fournisseurs de services, entités tierces), le choix des plans qui sont programmables (signalisation, données, supervision) et naturellement dans le choix des paradigmes et technologies utilisées pour mettre en œuvre cette progammabilité. Ces différences seront développées dans les sections suivantes.

1.2 Quelques motivations profondes

Les concepts liés aux réseaux programmables n'ont pas émergé spontanément du néant. Bien que ceux-ci soient nés dans des contextes et projets de recherche différents, ils partagent tous un ensemble de constats communs sur l'inadéquation des réseaux actuels. De ces constats découlent des motivations pour la mise en place de réseaux programmables. Nous les énumérons ci-dessous :

1. *les délais entre la spécification initiale d'un nouveau service, sa normalisation et/ou standardisation et finalement la réalisation et le déploiement à grande échelle sont devenus gigantesques*[8] *et ne sont plus adaptés aux exigences de réactivité des opérateurs et fournisseurs de services.*

   Ceci est un constat d'inadéquation temporelle[9] entre la fréquence soutenue d'émergence de nouveaux services, la lenteur des processus de standardisation et la réactivité faible des acteurs en aval de cette standardisation au niveau de l'industrialisation de ces services. On peut ajouter à la lenteur des processus de standardisation, des ralentissements induits par les problèmes de compatibilité inverse et de politique de déploiement qu'induisent de nouveaux protocoles. De telles contraintes peuvent avoir un impact énorme sur un protocole, pouvant dans le pire des cas mener à son abandon ;

   Apport des réseaux programmables : pour accélérer le mouvement, la standardisation des services n'est virtuellement plus nécessaire. En effet, au travers d'interfaces de programmation qui sont

8. Il est aujourd'hui aquis que la standardisation d'un service requiert entre 6 et 8 ans.

9. En fait deux rythmes inadéquats s'affrontent : un rythme technologique très rapide (2 à 3 ans) et un rythme politique, économique et stratégique (6 à 8 ans).

standardisées une fois pour toute[10], un fournisseur de services peut développer et déployer rapidement de nouveaux services et protocoles. De plus, il peut grâce à la dynamicité de l'approche faire fonctionner différents protocoles réglant ainsi le problème de la compatibilité ;

2. *le réseau peut profiter des informations issues d'applications sur la nature et la sémantiques de leurs flux.*

   Cette connaissance sur la nature des données échangées entre les applications, le traitement à leur appliquer ainsi que sur la topologie logique des flux entre les applicatifs permet au réseau d'agir de manière optimale sur ces données et d'accorder au mieux la topologie logique des flux de l'application à sa topologie physique. De plus, le placement stratégique de fragments applicatifs dans les nœuds stratégiques du réseau peut en améliorer le fonctionnement [TEN 96a] ;

   Apport des réseaux programmables : dans certaines approches des réseaux programmables, ce problème est directement résolu par la capacité offerte par le réseau à supporter dans ses nœuds des traitements directement fournis et, dans certains cas, déployés par l'application. Dans d'autres approches, la nature ouverte du réseau au travers d'interfaces de programmation permet à un fournisseur de services de développer et déployer rapidement un protocole adapté aux besoins des usagers et de leurs applications ;

3. *les technologies actuelles permettent la conception, le développement et le déploiement de composants logiciels ouverts (codes portables, mobiles) sur des architectures hétérogènes.*

   De plus les travaux sur la sécurité des codes mobiles, leur compilation optimale et rapide [HAR 96], ainsi que sur les systèmes d'exploitation extensibles modulaires et performants ont abouti à de nombreux résultats et des implantations sont aujourd'hui disponibles ;

   Apport des réseaux programmables : les réseaux programmables n'apportent à priori rien à ces technologies mais permettent au monde des télécommunication de tirer partie de ces avancées pour mettre en place des architectures souples, ouvertes et extensibles. C'est précisemment cette évolution des technologies qui permet aujourd'hui aux réseaux programmables de disposer de bases solides, même si tout n'est pas résolu, pour permettre leur mise en œuvre à grande échelle ;

4. *les capacités de traitement et de mémoire des composants internes du réseau suivent une croissance exponentielle. Les coûts des processeurs et de la mémoire chutent de manière drastique.*

   Ces deux effets sont bien sûr étroitement liés et principalement induits par le marché de masse.

   Apport des réseaux programmables : si cette évolution encourage la mise à disposition de ces ressources de calcul pour déployer de nouveaux services dans les équipements des réseaux, cette distribution de la logique de service permet également une meilleure utilisation de ces capacités de traitement et une meilleure utilisation globale du réseau en optimisant les flux grâce à des traitements dans les nœuds intermédiaires [LEG 98b] ;

5. *on assiste à un accroissement des fonctionnalités applicatives mises dans les composants du réseau pour mieux répondre aux besoins de services des usagers (ex. Firewall, caches Web, composants d'annuaire, proxy mobiles)*

   Apport des réseaux programmables : offrir une infrastructure unifiée et intégrée pour réaliser des applicatifs qui infèrent directement sur le comportement du réseau et ceci à différents niveaux (applicatif, présentation, session, transport, réseau).

En résumé, les apports des réseaux actifs face aux attentes des opérateurs, usagers et fournisseurs de services sont virtuellement énormes.

S'ajoutent à ces motivations principales, des intérêts dérivés. Par exemple, les réseaux actifs offrent une formidable base d'expérimentation pour de nouveaux services et protocoles, le développement de code portable, les traitements au cœur du réseau ainsi que la validation des idées liées à la reconfiguration dynamique et sûre de l'ensemble d'un réseau. À ce titre, ils peuvent être extrêmement utiles aux chercheurs et aux enseignants en réseaux et télécommunication leur fournissant une infrastructure souple et peu onéreuse pour concevoir, développer, déployer et expérimenter sur des réseaux réels des protocoles et services innovants.

10. Il y a tout de même ici une hypothèse forte qui suppose qu'il est plus facile de standardiser une API et une abstraction de ressources que des protocoles de signalisation pour la QoS ou d'autres services. Lorsque l'on voit ce qui se passe en supervision avec les modèles de l'information, ce pari n'est pas forcémment gagné d'avance.

1.3 Objectifs et organisation du rapport

Ce rapport a pour objectif de fournir une référence claire, complète et didactique sur les réseaux programmables. L'évolution rapide des recherches dans ce domaine requiert une nature active[a] *du rapport. La version actuelle reflète les résultats les plus récents. Les évolutions à venir seront inclues dans des versions ultérieures de ce rapport. Le rapport s'appuye également sur un site Web maintenu par les membres du projet. Ce site référence l'ensemble des liens sur le sujet dont bien sûr, tous ceux mentionnés dans ce document. Certains projets qui ne sont pas mentionnés dans cette version du rapport, sont accessibles par le siteWeb. Son URL est la suivante:* `http://www.loria.fr/~festor/RAF/RAF.html`

[a] Comme rapporté à juste titre par l'un des auteurs, ce document est un méta-rapport auto-récursif.

Le rapport est organisé de la manière suivante. La section 2 affine la définition des réseaux programmables donnée dans l'introduction et présente une classification qui nous servira tout au long de ce rapport.

La section 3 présente l'ensemble des initiatives autour d'une famille d'approches appelée réseaux programmables. La section 4 présente l'ensemble des architectures et projets existants autour des réseaux actifs à ce jour. Les architectures de standardisation autour des réseaux actifs sont présentées dans la section 5. L'ensemble des domaines d'applications de ces deux familles d'approches est détaillé dans la section 6. Les prototypes et logiciels disponibles à ce jour sont présentés dans la section 7. Les systèmes d'exploitation dédiés aux réseaux actifs sont présentés dans la section 8. Les aspects liés à l'industrialisation de solutions programmables ainsi que les freins actuels sont traités dans la section 9. La section 10 conclut la présentation.

2 Une classification des approches

Classifier les réseaux programmables n'est pas une tâche triviale. Différents auteurs ont défini différentes classifications ou sous-classifications [CAM 99c, TEN 97, PSO 99]. Cette section reprend ces propositions, en identifie les points communs, et les regroupe dans une nomenclature unique utilisée tout au long de l'ouvrage. La figure 1 fournit un découpage entre les différentes familles d'approches. Dans la suite de cette section nous détaillons ces différentes composantes.

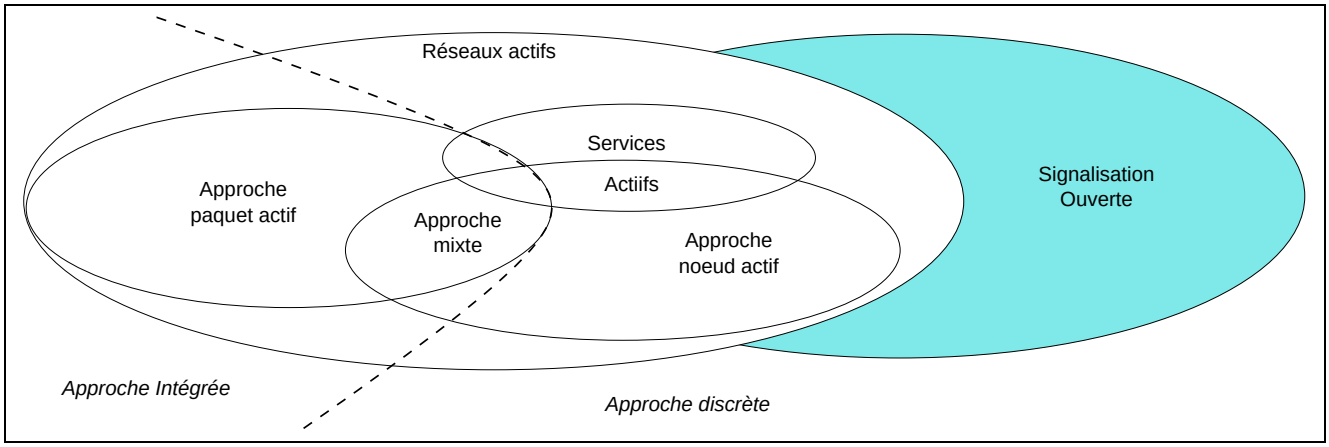

FIG. 1 – *Nomenclature de classification*

Il existe aujourd'hui deux grandes familles d'approches des réseaux programmables. La première, qui est appelée **signalisation ouverte (OPENSIG)** est centrée autour de l'ouverture des réseaux actuels via la définition d'interfaces de programmation offrant l'accès au plan de signalisation des équipements. Ceci permet notamment à des entités tierces (autres que les constructeurs d'équipements) de proposer des protocoles de contrôles et des services à valeur ajoutée directement intégrables dans le réseau de l'opérateur ou le fournisseur de services comme nous le verrons par la suite.

Définition 3 *Une approche de signalisation ouverte est une approche dans laquelle le plan de signalisation est remplacé par un environnement logiciel distribué permettant la programmation de nouveaux protocoles et de services.*

Dans ce type d'approche, le réseau est programmé de manière *out-of-band*[11] par l'opérateur ou le fournisseur de services et les plans de données et de supervision ne sont pas modifiés. Se basant le plus souvent sur une architecture CORBA, cette vision de la programmation du réseau est aujourd'hui la plus avancée. Elle est entrée dans un processus lourd de formation de consortiums industriels, de standardisation d'interfaces et de batailles technologiques.

La seconde, qui est plus généralement connue sous le terme **réseau actif**, est plus ouverte que l'approche réseau programmable dans ses domaines d'application, dans sa vision de l'architecture de nœuds et de la programmation du réseau.

Définition 4 *Un réseau actif est un réseau dans lequel tout ou partie de ses composants dans les différents plans (signalisation, supervision, données) sont programmables dynamiquement par des entités tierces (opérateur, fournisseur de services, applications, usagers).*

Cette approche est plutôt issue des travaux sur l'insertion de services applicatifs dans le réseau Internet, par opposition aux réseaux programmables qui sont issus de travaux sur la signalisation pure dans les réseaux de télécommunication de type ATM. Elle étend le concept de programmation en partant du principe que virtuellement tout usager peut concevoir, déployer et utiliser un nouveau service de manière dynamique dans le réseau.

Son caractère ouvert lui laisse également une liberté bien plus grande sur les choix d'architecture et de technologie. Cette famille d'approche peut être affinée en deux classes : celles qui se basent sur une approche **paquet actif** et celles qui se basent sur une approche **nœud actif**.

Dans les approches paquet actif, le code des services déployés dans les nœuds du réseau est transporté jusqu'aux nœuds dans le même flux que les données traitées par le service visé. On parle alors d'une **approche intégrée**. Le cas standard consiste en l'envoi de paquets de code pour le déploiement du service avant l'envoi de paquets de données. Le cas extrême permet à chaque paquet de données de véhiculer également un «mini-programme» contenant la logique du service à lui appliquer.

Définition 5 *Une approche intégrée est une approche réseau actif dans laquelle le déploiement des services dans les nœuds est conceptuellement intégré dans le flux usager, et dans laquelle la durée de vie d'un service est conceptuellement liée à celle du trafic affecté.*

Dans les approches nœud actif, les services sont également déployés dynamiquement sur les nœuds du réseau mais ce déploiement ne se fait pas conceptuellement dans le même flux que celui des données utilisateurs. Ceci se rapproche fortement du concept de réseau programmable. La différence avec ce dernier repose sur le fait que dans l'approche nœud actif, les composants de service sont déployés physiquement sur les nœuds du réseau, le plus souvent grâce à des techniques de code mobile. Dans les réseaux programmables, la logique de service et son implantation ne nécessitent la plupart du temps pas de déploiement sur les nœuds, celle-ci étant exécutée au sein de l'environnement de programmation du réseau. Cette programmation externe permet également de donner le nom d'**approche discrète** à cette famille qui intègre également un sous-ensemble des propositions autour de la signalisation ouverte.

Définition 6 *Une approche discrète est une approche de réseau actif dans laquelle les services sont déployés dynamiquement dans les nœuds du réseau mais en dehors des flux que ces services traitent.*

Conceptuellement, l'approche paquet actif est beaucoup plus ouverte que l'approche nœud actif mais dans la première, les codes déployés sont nécessairement de plus petite taille[12] et ne peuvent donc être utilisés que dans un nombre restreint de situations.

Une dernière catégorie qui n'apparaît pas dans la classification précédente est celle des **services actifs**. Elle regroupe toutes les architectures qui prônent l'actif mais à un niveau applicatif uniquement en proposant des environnements de passerelles actives sur lesquelles des services peuvent être conçus, développés et déployés via une plate-forme dédiée et ceci sans modifier le réseau sous-jacent. Cette catégorie se place entre les réseaux actifs et les réseaux programmables en tirant profit des 2 principes.

Finalement, on retrouve souvent dans le contexte des réseaux actifs, l'utilisation de plates-formes à agents mobiles. Pour clarifier la confusion qui apparaît souvent entre actif et agent mobile nous donnons la définition suivante :

Définition 7 *Les paquets actifs dans des réseaux actifs représentent un sous-ensemble de la famille des agents mobiles dont ils diffèrent en raison de l'orientation forte vers la communication et la soumission forte à des*

11. En dehors des flux usagers
12. Tout simplement pour des raisons de performance.

contraintes d'efficacité en terme de QoS, CPU et bande passante qui s'applique à cette aproche dite paquets actifs.

Conceptuellement proches, les paquets actifs ne sont pas considérés de la même manière que des agents mobiles en raison des contraintes qu'ils doivent satisfaire. De plus, les paquets actifs se déplacent rarement avec leur contexte d'exécution, contrairement aux agents mobiles. Finalement, les agents mobiles sont souvent perçus comme des entités applicatives et de ce fait n'ont pas accès aux informations de niveaux réseau ou inférieur nécessaires aux traitements dans les réseaux actifs. Cela dit, cette différence reste très subjective.

Cette classification en deux familles définit l'organisation de ce rapport. Les réseaux programmables pour la signalisation ouverte dans les réseaux de télécommunications sont les premiers présentés (section suivante). Les réseaux actifs sont détaillés dans des sections ultérieures.

3 Les réseaux programmables

L'expérience des réseaux de télécommunication montre qu'il n'existe pas à ce jour et qu'il n'existera probablement jamais de protocole unique dont les caractéristiques sont telles qu'il est idéal dans toutes les situations (pour tous supports et pour toutes les applications). Par exemple, TCP tel que nous le connaissons actuellement sur IP, n'est pas idéal dans des environnements sans fil et satellitaires et IP n'est pas adapté pour des conditions de mobilité forte et des procédures de *handoff*. Partant de ce constat, il est nécessaire de tenter au niveau des architectures, non plus de forcer des protocoles donnés, mais de rendre celles-ci ouvertes afin de permettre le déploiement dynamique de protocoles existants adaptés et, le cas échéant, de nouvelles solutions.

L'objectif principal des réseaux programmables est d'aboutir à la définition d'un *système d'exploitation réseau* offrant des interfaces de programmation ouvertes permettant la conception, le développement et l'introduction rapide de nouveaux services. Concrètement, ce type d'architecture doit permettre aux opérateurs et fournisseurs de services de répondre rapidemment aux besoins des usagers et d'innover constamment dans le domaine des services offerts, ceci bien sûr à moindre coût.

Les réseaux programmables ont ces dernières années retenus l'attention de nombreux chercheurs et industriels de télécommunications. De nombreux projets ont aboutis à des résultats d'une qualité irréprochable permettant à ce type d'architecture d'entammer une phase d'industrialisation, aujourd'hui bien avancée. Cette section est consacrée à ces projets. Il détaille les principales architectures étudiées à travers le monde et présente les différents efforts de standardisation en cours.

3.1 L'initiative P1520

Initié en 1997 au sein de l'IEEE[13], le projet PIN P1520[14] (*Programmable Interfaces for Networks*) définit un modèle générique et standard d'interfaces de programmation pour les réseaux et propose des instanciations de ce modèle générique pour différentes technologies réseaux tels qu'ATM, IP, la plupart des services pour la gestion de la QoS dans IP ainsi que les réseaux intelligents.

L'objectif principal de cette initiative est de séparer l'architecture physique du réseau de celle des services et de la signalisation associée [BIS 99]. L'intérêt visé est de séparer les marchés du *hardware* des télécommunications de celui de la signalisation et des services. Ceci n'est possible qu'en fournissant une abstraction des ressources de communication sous forme d'interfaces de programmation au sein d'une infrastructure distribuée basée sur des technologies objets.

Dans une telle approche, le réseau devient un système distribué programmable permettant aux opérateurs et fournisseurs de services de s'affranchir des contraintes mises par les équipementiers sur les services proposés et de faire appel à des tiers pour développer des composants de services à valeur ajoutée répondant aux besoins des fournisseurs et fournissant une base solide dans un contexte de compétition acharnée. Fournissant une base pour les services à valeur ajoutée, une telle approche permet également de bénéficier des dernières évolutions dans le domaine du génie logiciel, de la modélisation objet et des plates-formes à objets distribués.

P1520 se différencie de la supervision en placant dans ses critères, une contrainte de réactivité temporelle de l'infrastructure forte. En effet, celle-ci est fixée à l'échelle de la signalisation (quelques millisecondes) et non de la supervision (quelques secondes à quelques minutes)[15]. P1520 se positionne également par rapport à des

13. `http://www.ieee.org`
14. `http://www.ieee-pin.org`
15. Il n'est d'ailleurs pas exclu que si une telle interface se développe à grande échelle, celle-ci puisse remplacer à terme l'interface de management.

approches comme *TINA (Telecommunications Information Networking Architecture)*[16] en indiquant que les préoccupations de P1520 se situent sur des interfaces de plus bas niveau (équipement) alors que *TINA* propose des interfaces de plus haut niveau[17].

3.1.1 Modèle de référence

La figure 2 comporte le modèle générique (ou modèle de référence) de l'organisation des interfaces de programmation ouvertes. Celui-ci est subdivisé en quatre couches offrant chacune une interface de programmation ouverte. Les deux couches les plus basses représentent le niveau d'un équipement de commutation ou d'un routeur.

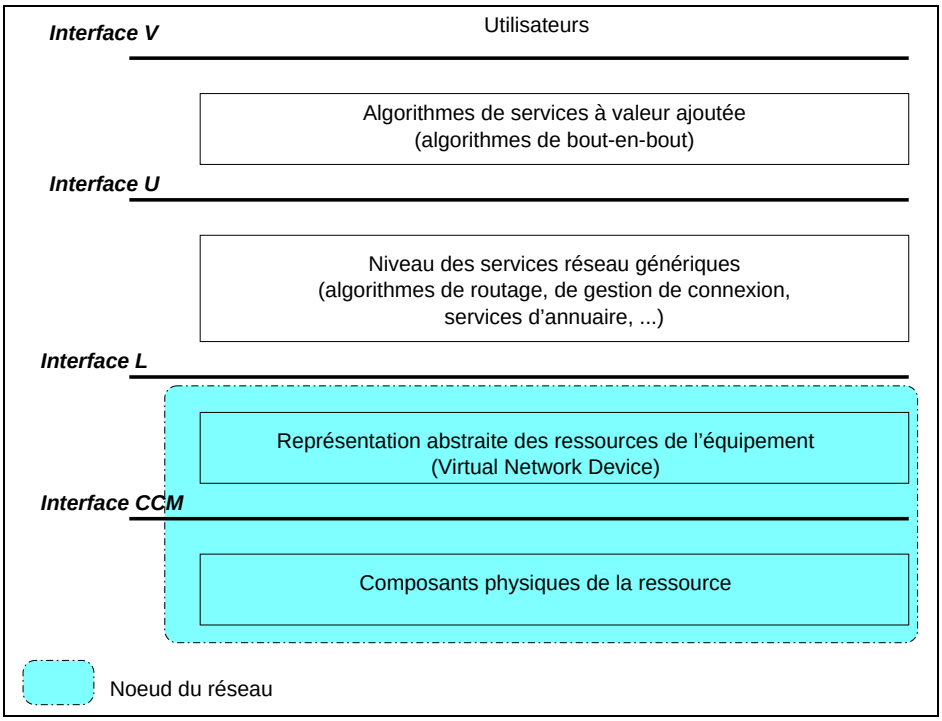

FIG. 2 – *Le modèle de référence P1520*

Deux interfaces y sont identifiées :

- **CCM** (*Connection Control and Management*) définit l'accès aux états physiques de l'équipement sous-jacent. Au travers de cette interface, qui n'est pas standardisée, sont définis les protocoles qui permettent à un tiers externe à l'équipement d'échanger avec ce dernier des informations de contrôle et d'état. Par exemple cette interface peut être de type GSMP (*General Switch Management Protocol* [RFC 96]) sur des commutateurs ATM ;
- **L** (*Lower*) définit une interface abstraite de l'ensemble des ressources de l'équipement (CPU, mémoire, bande passante, VC/VP ATM, ...). Celle-ci est réalisée par la définition d'un objet abstrait et de ses interfaces.

À un niveau supérieur et offrant une vision de l'ensemble du réseau, on retrouve également deux interfaces :

- **U** (*Upper*) offre un accès aux algorithmes génériques de service du réseau. Ces algorithmes sont représentés sous forme d'objets qui interconnectent les objets présents sous l'interface **L**. Ces objets représentent des composants de routage, de réservation/différenciation de services dans le cadre IP, des fonctions de gestion de routes et de connexion dans un contexte ATM ;

16. `http://www.tinac.com`
17. *NDLR : cet argument est certes valable dans les premières phases du projet mais il est de notre point de vue d'une mauvaise foi totale car tous les participants travaillent sur la base du modèle de référence sur des interfaces de haut niveau pour la création et le déploiement de services, domaine cible de TINA.*

– **V** (*Value-added*) est l'interface de niveau le plus élevé. Cette interface au travers de sa réalisation dans la couche sous-jacente vise à offrir un ensemble riche d'interfaces permettant de développer des services à valeur ajoutée (ex. service de diffusion multimédia garantissant la synchronisation image/voix, facilités de pause, avancée/retour rapide sur un flux vidéo, ...).

Sur la base de ce modèle de référence, les objectifs du projet P1520 sont de faire du *reverse engineering*[18] sur les plans de contrôle des réseaux ATM, IP et *SS7 (Signalling System number 7)* de manière séparée en encapsulant les différents protocoles de signalisation et en ne se focalisant que sur les niveaux **CCM** et **L**, même si les niveaux supérieurs ne sont pas exclus du projet.

L'ensemble de ces interfaces sont spécifiées en CORBA IDL (*Common Object Request Broker Architecture Interface Definition Language*[19]) et permettent l'utilisation de tout langage de programmation pour leur implantation effective. Les sections suivantes présentent les travaux dans les différents domaines.

3.1.2 Composants ATM spécifiés

Les premières instanciations de ce modèle ont été faites pour les réseaux ATM par les chercheurs de l'université Columbia[20] avec des propositions d'interfaces **CCM**, **L** et plus récemment **U**. En fait, plusieurs composants de spécifications P1520 sont aujourd'hui disponibles autour d'ATM. Issues de plusieurs laboratoires ou groupes industriels, ces propositions ne sont pas normalisées à ce jour.

Les principales contributions sont issues du projet *X.Bind* qui a depuis donné naissance à une startup. Ces propositions comprennent notamment les spécifications suivantes :

– une abstraction d'un commutateur ATM [ADA 99d] et sur la base de cette abstraction, une spécification des ressources d'un tel commutateur [ADA 99c] ;
– une spécification de l'interface de service d'un commutateur [ADA 99b].

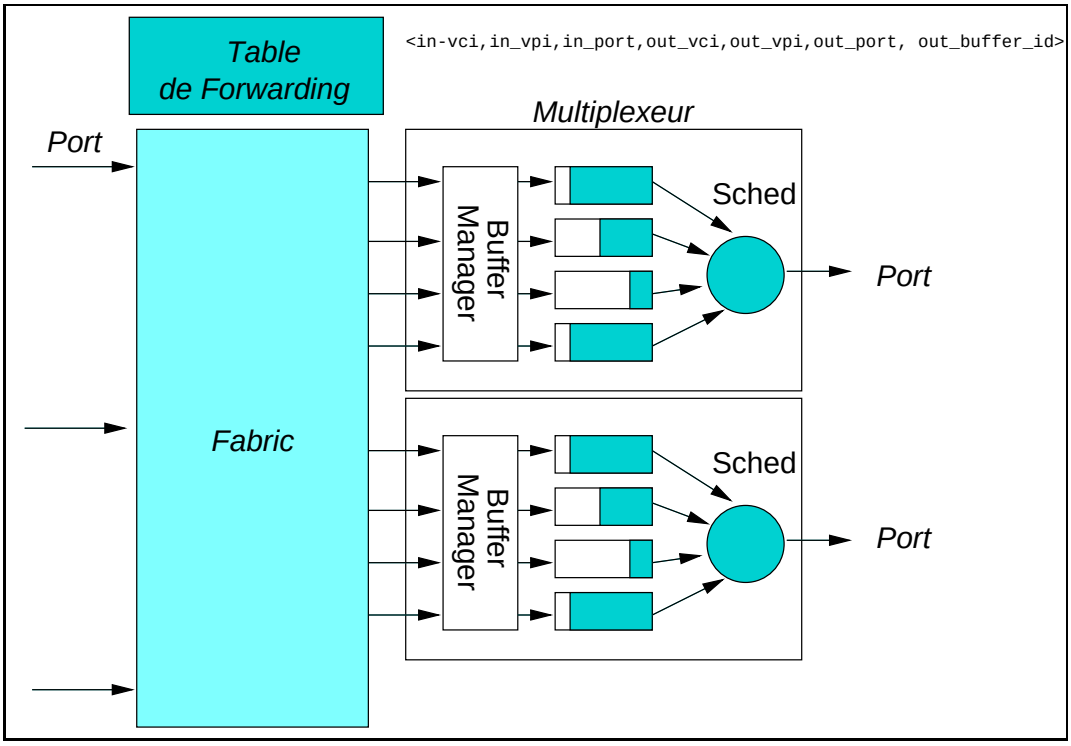

FIG. 3 – *Le modèle abstrait de commutateur ATM*

La figure 3 illustre le modèle abstrait d'un commutateur. Celui-ci est caractérisé par un certain nombre de ports en entrée et en sortie par la matrice de commutation (table de forwarding) et, pour chaque port de sortie,

18. Ingénierie à rebours :-).
19. `http://www.omg.org`
20. `http://comet.ctr.columbia.edu/pin-atm/`

par un multiplexeur composé d'un gestionnaire de buffer, d'un ou plusieurs buffers ainsi que l'ordonnanceur associé au port de sortie. La configuration du commutateur est maintenue dans une table de *forwarding* qui comprend tous les couples `<chemin virtuel, buffer de sortie>` où un chemin virtuel est caractérisé par : les identificateurs de chemin virtuel entrant et sortant, les identificateurs de circuit virtuel entrant et sortant, les ports d'entrée et de sortie ainsi qu'un identificateur du buffer de sortie associé à ce flux.

La caractérisation de la capacité du commutateur est basée sur le concept de région d'ordonnancement (*Schedulable Region*) défini dans [HYM 91]. Ce concept définit la région dans l'ensemble des charges (en paquets par seconde et par classe de trafic) dans laquelle les contraintes de QoS sont respectées par un algorithme d'ordonnancement donné sur un port spécifique.

La spécification des ressources basée sur ce modèle comporte une description en IDL des 3 principaux composants à savoir la table de *forwarding*, le multiplexeur et la région d'ordonnancement. Elle est fournie dans [ADA 99c]. La table de *forwarding* est définie comme une table dans laquelle toute entrée représente un canal (`vci` entrée et sortie, `vpi` entrée et sortie, port entrée et sortie, identificateur du buffer associé). Les opérations offertes à cette interface sont la récupération d'informations générales sur la table (nombre de ports, capacité du commutateur, adresse MAC), l'ajout et la suppression de canaux, l'interrogation sur l'espace de nommage d'un port (intervalles possibles pour `vci` et `vpi` en entrée et sortie) et le retrait en une opération d'un canal multicast. Le multiplexeur est défini par son débit cellulaire, le nombre et la taille des buffers, la politique d'ordonnancement appliquée au sein de chaque buffer, la politique d'ordonnancement (*Scheduling Policy*) qu'il applique à l'ensemble des *buffers*. Ces différents paramètres sont décrits dans l'interface et des opérations d'accès et de modification sont définies pour le multiplexeur. La région d'ordonnancement est composée des classes de trafic, de leur association à des buffers de sortie et de la région elle-même définie comme une matrice de dimension m (nombre de flux) $* n$ (nombre de classes) représentant pour chaque entrée, les débits cellulaires sur ces buffers.

Une classe de trafic est représentée comme un couple description/QoS où la description de trafic comprend le débit crête, moyen, la taille de *burst* et le type (*premium, standard,...*), et la QoS des paramètres de délai maximum, moyen, taux de perte, délai entre pertes, et aire de mesure. Des opérations d'accès et de modification de ces différents paramètres sont fournies dans l'interface.

Le modèle des ressources correspond directement à l'interface avec l'équipement. En général, tout appel au modèle de ressource se traduit par un appel au protocole GSMP ou qGSMP vers le commutateur. La figure 4 illustre l'intégration de ces différentes interfaces qui sont inclues dans l'interface de *Binding* (liaison) de base [ADA 98].

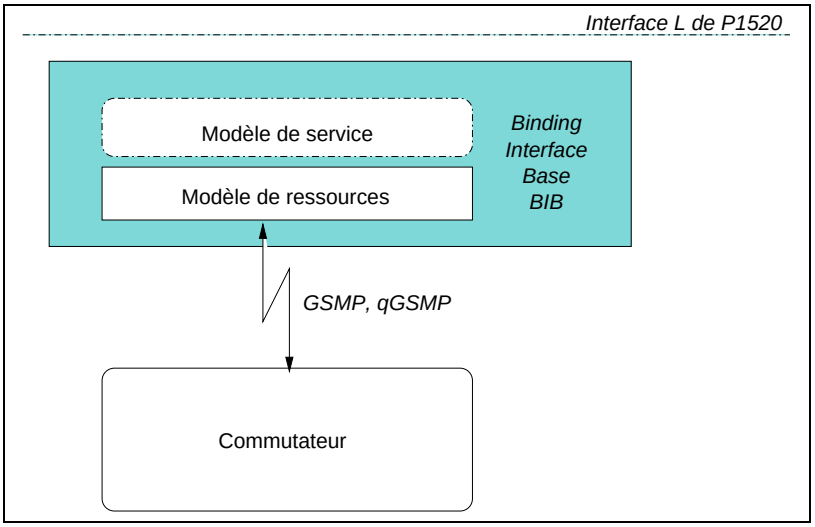

FIG. 4 – *L'intégration des modèles de commutateur ATM*

La spécification de l'interface de service d'un commutateur ATM [ADA 99b] offre une vision service des ressources modélisées précédemment. Ce service se compose de deux objets : une interface nœud ATM et un service de commutateur. Le premier offre des capacités d'acquisition et de libération de points de terminaison (VC et/ou VP). Le second offre une interface d'ajout et de libération de canaux (appelés *Segment* dans la

spécification) sur un commutateur. Ces deux composants semblent[21] issus de l'interface de liaison de base (*Binding Interface Base BIB* [ADA 98]).

Les principales investigations du sous-groupe ATM se concentrent aujourd'hui sur ce niveau d'interface. Cependant, des premières propositions pour des interfaces de plus haut niveau (en l'occurrence **U**) font l'objet de contributions. Récemment une proposition d'une telle interface pour de l'ATM sans fil a été faite dans [RAN 99]. Elle se base sur l'interface BIB pour ATM sans fil définie dans le cadre du projet P1520. L'interface **U** définit principalement deux composants :

1. un gestionnaire d'appel : objet de base d'un sous-réseau. Ce composant maintient les connexions ainsi que la connaissance des terminaux mobiles actuellement dans sa zone de gestion. L'interface offerte par ce gestionnaire permet de :
   - demander (resp. répondre à) la mise en place d'un circuit de QoS donnée ;
   - ajouter (resp. supprimer) un participant dans une communication multipoints ;
   - demander l'attachement au sous-réseau d'un terminal (procédure demandée par le terminal lui-même) soit à la suite d'une migration (*handover*), soit lors d'une première mise en service ;
   - engager une procédure de *handover* ;
2. un gestionnaire de route : objet intégrant les algorithmes de calcul de route et de gestion de la localisation des terminaux. Une route est ici définie comme une séquence de nœuds identifiés chacun par son adresse et définissant la portion de route par le port d'entrée et le port de sortie sur ce nœud. Cet objet est utilisé par le gestionnaire d'appel et offre trois méthodes : le calcul d'une route entre deux points avec des contraintes de QoS, une méthode de modification de la localité d'un terminal et une méthode de consultation de la localité d'un terminal.

3.1.3 Composants pour IP

La définition d'interfaces de programmation de routeurs et/ou commutateurs IP a énormément progressé depuis sa création en 1998. Les premières études ont porté sur l'intérêt du modèle dans le contexte IP et les résultats démontrent que l'intérêt est fort dans un contexte IP avec QoS. C'est sur ce domaine que les travaux se sont donc lancés. Dans un premier temps, un *mapping* générique du modèle P1520 (couches + interfaces) a été défini vers le monde IP [LIN 99b]. Ce *mapping* est illustré dans la figure 5 ci-dessous.

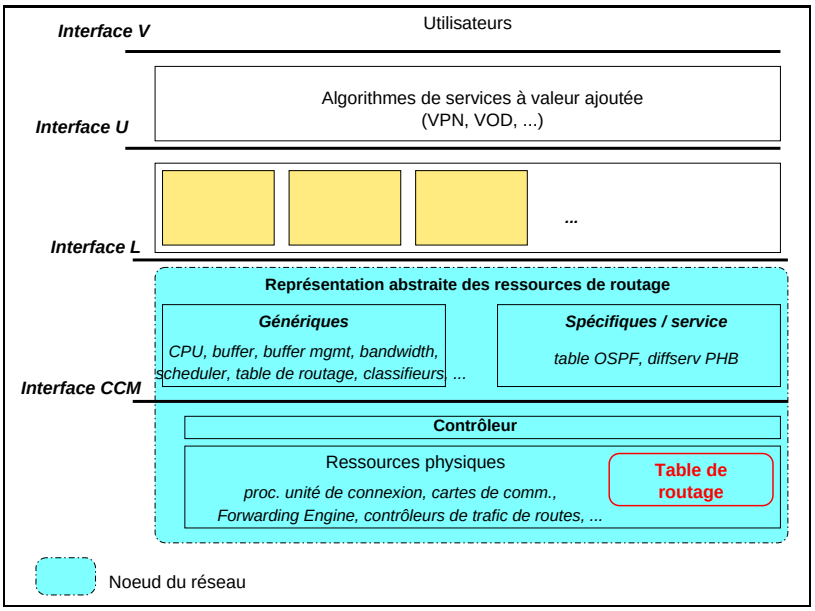

FIG. 5 – *Instanciation du modèle de référence pour IP1*

21. En fait, vu la complétude des documents P1520 et notamment grâce aux références inexistantes entre ces documents, les auteurs ne sont pas en mesure de garantir ce découpage de BIB en deux niveaux.

Les applications visées par ces interfaces de programmation sont principalement des applications de configuration et de modification dynamique des équipements et des ressources (ex. allocation de ressources par classe de trafic).

Parallèlement à ces travaux d'architecture, des premières propositions d'interfaces **L** ont vu le jour autour de la différenciation de services[22]. Les premiers résultats de ce travail proposent une extension du modèle P1520 reposant sur la nature différente des composants définis dans une interface **L** à savoir des composants génériques, indépendants de tout service spécifique et des composants spécifiques à des services donnés (ex. *DiffServ*, MPLS, ...). Pour cela, les auteurs de ces travaux préconisent au sein d'une interface **L** de fournir dans un premier temps des abstractions de ressources génériques et de permettre la spécialisation de ces ressources pour un service particulier.

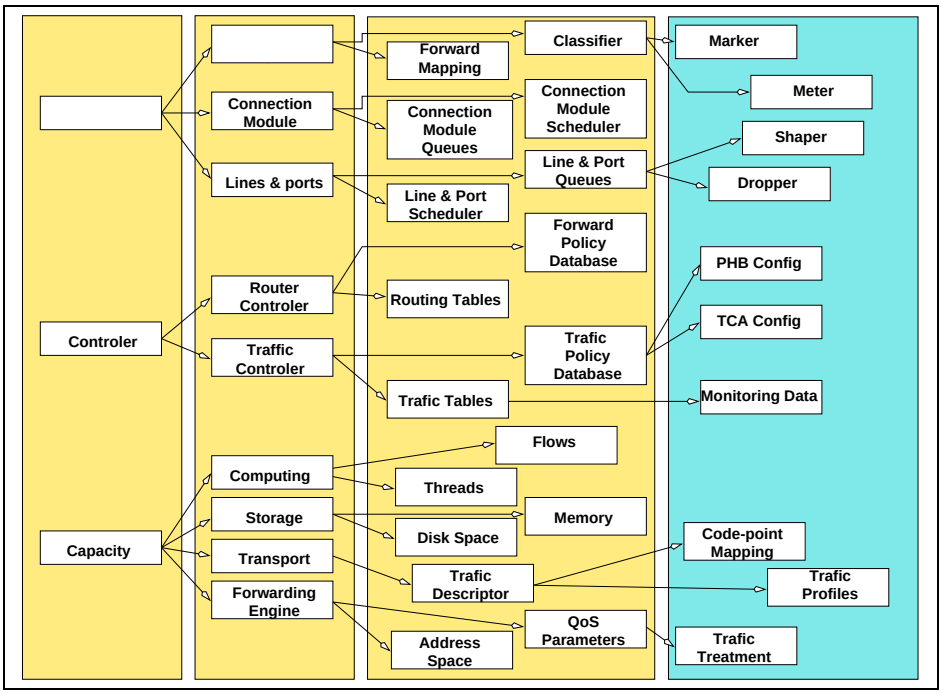

FIG. 6 – *Les classes de l'interface L pour IP et la différenciation de services*

Ceci a notamment été réalisé pour un routeur IP et pour le service *DiffServ*[23] [LIN 99a, DEN 98, DEN 99]. La figure 6 illustre les différents composants qui entrent dans l'interface **L** d'un routeur IP. Ces composants sont d'une part des abstractions génériques de base (transport, caractéristiques du routeur, entité de contrôles) qui regroupent les entités fonctionnelles qui les représentent (contrôle de route, capacité de traitement, mémoire, entité de *forwarding*), elles-même mises en place par des composants (port, tables de routage, classifieurs, paramètres de QoS, ...).

Ce modèle générique peut ensuite être spécialisé pour un service spécifique (ici *Diffserv*). L'ensemble de ces composants forment l'interface abstraite **L** pour un routeur IP avec services de QoS. Plus récemment, une proposition d'interface **U** vient d'être proposée permettant de mettre au sein d'un seul réseau, des réseaux virtuels avec protocoles de QoS et mécanismes de routage spécifiques DiffServ, MPLS, IntServ, ... Si les travaux apportent des abstractions intéressantes, le groupe IP souffre actuellement d'un problème majeur qui est l'absence d'un protocole de contrôle de type GSMP permettant d'accéder réellement aux ressources des routeurs.

3.2 Composants pour le réseau intelligent

Les travaux autour des interfaces programmables issues du modèle de référence P1520 dans le domaine du réseau intelligent et de la signalisation SS7 n'ont pas encore véritablement débuté. En effet, la masse critique

22. À la fois surprenant car au départ le groupe IP de P1520 visait des interfaces pour MPLS, mais également compréhensible car DiffServ est certainement un excellent candidat pour des interfaces de contrôle de haut niveau.

23. Différenciation est vue ici comme la classification, le marquage, le lissage au bord du réseau sur la base d'un contrat et le marquage au cœur du réseau.

en terme de nombre de participants et poids des participants dans le monde industriel ne semble pas atteinte aujourd'hui pour ce groupe. De ce fait, aucun résultat n'a été publié à ce jour. Seules quelques études de positionnement par rapport à TINA-C et d'autres groupements tels quel l'*OMG*[24] et *JAIN*[25](*Java APIs for Integrated Networks*) ont été menées motivant l'intérêt d'une approche P1520 dans ce domaine.

Dès que cette masse critique sera atteinte, des travaux vont pouvoir débuter et aboutir rapidement car le domaine est bien balisé.

3.3 Mobiware

Mobiware [CAM 99d, ANG 98] constitue une extension des travaux autour d'*X.Bind* effectués à l'université Columbia. Cette extension porte sur l'application des paradigmes de réseau programmable aux réseaux mobiles sans fil. *Mobiware* adresse deux objectifs principaux. Le premier est de concevoir et déployer un mécanisme de qualité de service adaptable pour des réseaux sans fils mobiles. Le second objectif est de concevoir et réaliser une architecture ouverte, programmable et active pour l'insertion de nouveaux services de signalisation incluant la signalisation pour la qualité de service adaptative dans des réseaux ATM. *Mobiware* est basé sur l'approche P1520 et développe sur cette approche de nouveaux objets pour la mobilité et pour la QoS adaptative.

L'API de QoS adaptative permet aux applications de spécifier au réseau une fonction de mesure de qualité (évaluation de la perception de la qualité par l'application). Cette fonction de qualité est complétée par un ensemble de politiques d'adaptations à mettre en œuvre sur différentes conditions de modification de la qualité de service sur le lien fixe/radio. Finalement, l'API permet également aux usagers de donner au réseau une classification des flux mobile/fixe. Ces préférences sont utilisées par le réseau pour dégrader les flux les moins prioritaires en cas de chute de la qualité de service.

Deux conditions de variation de la qualité de service sont prises en compte dans l'approche : une variation due aux conditions du lien mobile/fixe (perturbations par exemple) ainsi qu'une variation suite à un changement de base (procédure de handover). L'application spécifie au réseau les conditions de mise en œuvre des procédures d'adaptation pour son flux (modifications instantanées, graduelles *i.e.* après un délai fixe, lors d'une procédure de *handoff* ou jamais).

Pour mettre en place cette architecture programmable, *Mobiware* s'appuie sur le réseau fixe et sur l'architecture définie dans *X.Bind*. Cette architecture est enrichie par des objets pour :

- la gestion des mobiles;
- la gestion de la QoS vers/depuis les mobiles;
- le traitement d'aggrégations de flux pour optimiser les redirections en cas de *handover*.

Le réseau retenu est un réseau IP commuté sur ATM. Le niveau transport assure les fonctions actives entre le réseau et les mobiles. Il implante les fonctions d'adaptativité de QoS. Les fonctions prédéfinies dans l'approche sont une fonction de filtres actifs (adaptation de vidéo/audio multi niveaux) et une fonction de *FEC* (*Forward Error Correction*) (protection par adaptation du codage des paquets au taux d'erreur du support radio). Ces types de fonctions sont réalisées par des objets Java qui peuvent être déployées dynamiquement sur les mobiles. Au niveau réseau, *Mobiware* implémente un mécanisme de gestion d'aggrégation de flux. Celui-ci facilite la redirection de flux lors d'une procédure de changement de point d'accès pour un mobile. La seconde fonction implantée au niveau réseau est une fonction de traitement du *handover* adaptée aux variations de QoS ainsi qu'un mécanisme de maintient d'états pour les mobiles. Le niveau MAC de l'architecture est un niveau programmable fourni par un industriel.

3.3.1 Les objets de Mobiware

Deux types d'objets composent le *middleware* de signalisation et de transport de *Mobiware* : des objets CORBA fixes pour la signalisation et des objets Java déployables dynamiquement sur différents équipements ainsi que sur les mobiles pour le traitement des flux de niveaux transport (plan de données). Les objets de base de l'architecture de signalisation de *Mobiware* sont :

- le MDO (*Mobile Device Object*) situé sur chaque mobile. Cet objet offre aux applications une interface de programmation permettant de se connecter à un point d'accès du réseau, d'établir et de libérer des flux, d'obtenir des informations sur la qualité du lien radio et de négocier avec le point d'accès les algorithmes

24. `http://www.omg.org`
25. `http://java.sun.com/products/jain/index.html`

à déployer sur modifications de QoS (lien radio et *handoff*). C'est au travers de cet objet CORBA que les applications configurent les paramètres de QoS définis dans *Mobiware* ;
- le MAO (*Mobility Agent Object*) : composant principal. Il a pour responsabilité la gestion des flux vers/depuis le mobile associé, l'adaptation aux conditions du réseau et la gestion de la mobilité. Tout mobile dispose d'un tel agent sur le réseau fixe ;
- le SAP (*Service Access Point*) : objet représentant le point d'ancrage d'un mobile sur le réseau à un moment donné. Cet objet gère l'enregistrement d'un mobile sur l'accès au réseau fixe, propage les demandes de connexion ainsi que les informations de rafraichissement. Cet objet sert également d'intermédiaire pour le déploiement de filtres actifs au niveau transport vers le mobile. C'est avec ce dernier qu'il négocie les algorithmes de gestion de QoS mis en place (FEC et adaptative QoS notamment).

La localisation de ces objets dans l'architecture physique est illustrée dans la figure 7. S'ajoutent à ces objets spécifiques, des extensions d'objets définis dans *X.Bind* dont :

- ATM Route : objet permettant, sur une demande contenant une source et une destination, de retourner le chemin le plus court entre les deux. Cet objet est proche de l'objet de gestion de connexion défini dans *Genesis* (voir section suivante) et dans P1520. Il s'appuie sur une connaissance de la topologie du réseau ainsi que sur la disponibilité des serveurs virtuels de commutateurs ;
- le serveur de nœud : représente une abstraction de commutateur et offre une API pour la réservation de ressources (identificateurs de VP/VC, bande passante) pour chaque flux. Il maintient également pour chaque flux des informations d'état.

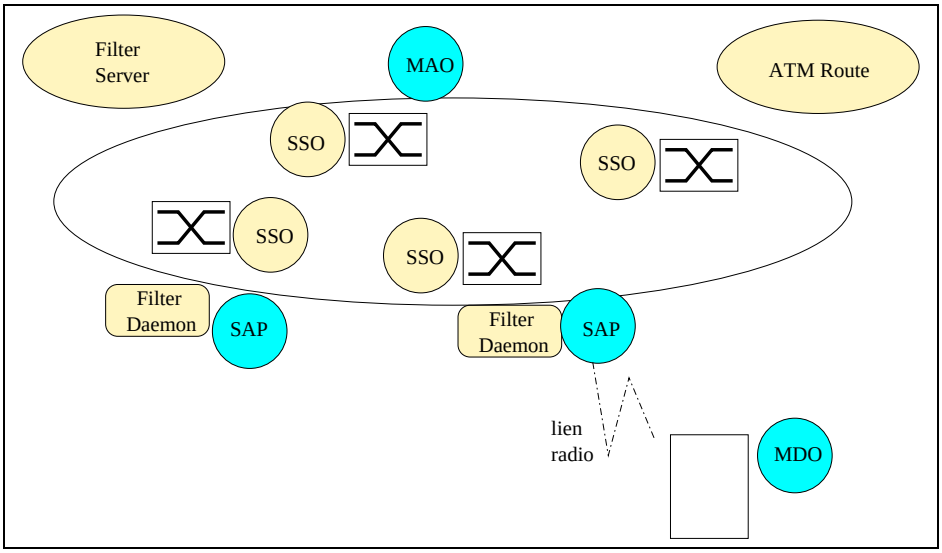

FIG. 7 – *Les objets de base pour la signalisation dnas Mobiware*

Pour la gestion de la QoS, notamment pour l'adaptation sur les flux de transport de données, l'infrastructure de *Mobiware* est enrichie des objets spécifiques additionnels suivants :

- des serveurs de filtres : *repositories* de classes Java implantant chacune un filtre particulier. Ces classes peuvent être déployées sur le chemin d'un flux utilisateur pour appliquer à celui-ci le filtrage qu'elles implantent. Dans la version initiale de l'implantation, seul un filte actif de gestion sélective sur des flux MPEG 1 est fourni ;
- des démons de filtrage qui sont localisés à différents points sur le réseau et qui sont capables d'accueillir des filtres et de les appliquer à des flux qui transitent sur le nœud sous-jacent. En général, ces démons tournent sur des points d'accès entre le réseau fixe et le réseau mobile.

L'ensemble de ces objets permet également de mettre en œuvre la gestion de la QoS adaptative sur des réseaux mobiles. De plus, cet ensemble fournit une infrastructure de base pour la conception, le déploiement et l'expérimentation de nouveaux protocoles de signalisation et d'adaptation pour les réseaux mobiles.

Ces composants ont été implantés sur un réseau expérimental. Les études de performance montrent qu'à l'aide de plusieurs optimisations du code (groupement d'objets CORBA, réduction des appels CORBA depuis

le mobile, pré-binding en mode *promiscuous*), une procédure de *handoff* (mise en place de tous les objets + redirection des flux) prend en moyenne 43ms ce qui est encourageant pour un prototype.

3.3.2 Résumé

Mobiware est une extension de l'approche X.Bind pour les réseaux sans fils mobiles. Cette approche fournit une panoplie d'objets permettant la gestion des mobiles et le traitement sur la base de variation de QoS des flux liés à une interaction entre un mobile et le réseau fixe. L'architecture développée démontre d'une part la faisabilité d'une approche programmable pour les réseaux mobiles et fournit une plate-forme d'expérimentation idéale pour le test de nouveaux algorithmes d'adaptation de QoS. L'ensemble des composants logiciels de ce projet sont disponibles sur Internet[26].

L'apport principal de cette approche dans le contexte de l'ouverture des réseaux est de démontrer que le couplage réseau programmable/réseau actif est utile pour couvrir différents plans d'un réseau et que ces approches sont utiles pour des réseaux mobiles. Les résultats du projet *Mobiware* serviront certainement de base de travail pour le groupe de travail sur les réseaux mobiles programmables au sein du projet P1520 présenté dans une section précédente.

3.4 Le projet Genesis

Genesis [CAM 99a, CAM 99b] est le projet le plus récent de l'université de Columbia autour des réseaux programmables. Il respose sur un constat d'extrême lourdeur et de complexité liées à la planification et au déploiement de réseaux privés virtuels aujourd'hui. Cette lourdeur principalement due à la nature manuelle de cette tâche implique des coûts et des délais importants incompatibles avec les besoins de réactivité des opérateurs et fournisseurs de services. Aussi, *Genesis* se propose de fournir une architecture logicielle permettant une automatisation du cycle de vie d'un *VPN* (*Virtual Private Network*).

L'idée de base pour atteindre cet objectif est de mettre en place un mécanisme de *spawning* de réseaux privés virtuels s'inspirant du *spawning* de processus dans les systèmes d'exploitation mais étendu à l'ensemble du réseau et maintenant des contraintes fortes sur les ressources. Dans ce but, une extension des architectures définies dans *X.Bind* et *Mobiware* est proposée. La cible est la définition d'un noyau réseau dédié à la conception, au déploiement dynamique et assisté ainsi qu'à la supervision des réseaux privés virtuels. Un objectif supplémentaire est de pousser à l'extrême le niveau de programmabilité des composants. Finalement, l'architecture logicielle visée doit pouvoir fonctionner sur plusieurs types de réseaux.

Pour cela, les chercheurs du projet ont formalisé le cycle de vie d'un réseau privé virtuel et conçu un «noyau réseau» capable de générer des réseaux privés virtuels à la volée. La génération d'un VPN se fait toujours depuis un *VPN* père dont les ressources contraignent celles de son/ses fils par héritage. Les réseaux virtuels générés supportent une signalisation, des mécanismes de contrôle de QoS et de supervision propres. Le noyau gérant ces *VPN* est construit sur *X.Bind* et implante toutes les fonctionnalités nécessaires au support du cycle de vie d'un réseau privé virtuel.

3.4.1 Architecture

Dans *Genesis*, un réseau privé virtuel est défini par un plan de données, un plan de contrôle ainsi qu'un plan de management propres. Ces différents plans représentent un noyau programmable construit sur *X.Bind* et intégré dans un environnement logiciel supportant l'ensemble des étapes du cycle de vie d'un VPN (planification, déploiement, opération, supervision et suppression). Logiquement, un *VPN* est représenté par un graphe dont les noeuds sont des *routlets* (sous-ensemble des ressources d'un routeur : ports en entrée, ports en sortie, *forwarding engine*, *classifiers*, *schedulers*, ...) et les arcs des liens virtuels définissant la topologie.

Le paradigme de base dans l'architecture est le *spawning*. Celui-ci comprend les différentes phases de la mise en place d'un VPN. La génération d'un tel réseau se fait suivant une relation père/fils : seul un VPN père peut générer un VPN fils et lui attribuer tout/ou partie de ses ressources. L'impact de cette relation père/fils sur l'architecture d'un noeud est illustré dans la figure 8. Dans cet exemple, le noeud physique est un noeud de transit de trois réseaux privés virtuels (Core, A et B). A et B sont deux fils du réseau Core et se partagent les ressources de ce dernier.

Le processus de *spawning* comprend les étapes de la mise en place de la topologie virtuelle, la réservation de l'espace d'adressage, l'allocation des ressources, l'instancation des objets CORBA liés au VPN à créer ainsi

26. `http://comet.columbia.edu/wireless/software/mobiware`

que la liaison de ces objets aux ressources réelles du réseau. Pour chaque plan, *Genesis* comporte des objets prédéfinis qui peuvent être soit spécialisés soit redéfinis.

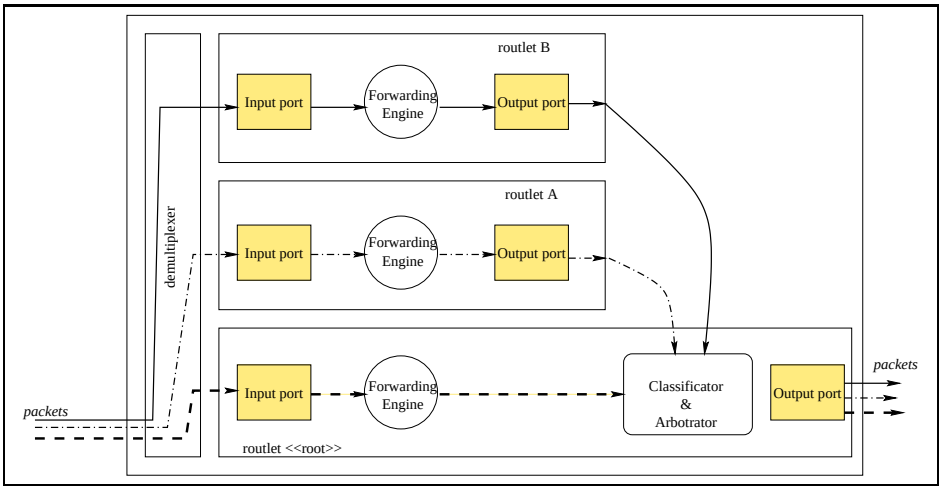

FIG. 8 – *L'organisation hiérarchique des VPNs dans un noeud Genesis*

Chaque VPN dispose de son bus logiciel propre. Celui-ci a toutes les caractéristiques d'un bus logiciel CORBA mais les opérations sont restreintes à un seul bus assurant l'isolation entre les VPNs. Concrètement, cela revient à isoler les ressources CORBA, notamment par le nommage en sous-ensembles isolés appelés *Orblets* dans *Genesis*.

Permettre la création de VPN, requiert la disponibilité d'un environnement logiciel dédié. Dans cette approche, les concepteurs ont défini un mécanisme de profilage et ont développé les outils associés. L'environnement comprend une interface graphique de saisie de configuration. Celle-ci comporte la topologie virtuelle, les besoins en ressources (capacité), le choix des algorithmes de forwarding, la sélection des algorithmes de routage à mettre en œuvre, éventuellemement une sélection de fonctions de tunneling ou de cryptage et finalement un choix parmi trois classes de ressources : *constant, average, best effort*. Derrière cet outil de saisie, la configuration est donnée à un environnement de validation qui, lié à la plate-forme de supervision de l'ensemble du réseau, est capable de confronter ces besoins aux disponibilités du réseau physique existant. Si la configuration est validée, le processus de spawning peut se mettre en œuvre et générer l'ensemble des objets en fonction du profil donné lors de la configuration afin de mettre en place le VPN.

En plus du noyau programmable et de l'environnement de création, *Genesis* comporte également un environnement de supervision des réseaux privés virtuels. Cet environnement comprend un contrôleur de VPN qui gère la facturation basée sur un modèle d'enchères. Ceci implique un effort important sur l'activité de *policing* et de scrutation afin de s'assurer qu'un VPN fils ne consomme pas de ressources du père autre que celles qui ont été négociées. Actuellement, un mécanisme de renégociation de ressources est également intégré dans l'environnement. Concrètement, la plate-forme de supervision est réalisée par un moniteur pour chaque *routlet*. Ce moniteur est piloté par un controleur central appelé *Maestro* dans l'approche.

3.4.2 Résumé

La contribution principale des travaux de *Genesis* est certainement la formalisation du processus de cycle de vie d'un réseau privé virtuel ainsi que la démonstration que l'ouverture d'une architecture de communication peut être poussée à l'extrême en autorisant l'ensemble des plans du réseau à être programmables. Le second point fort de l'approche réside dans le modèle de VPN retenu. Basé sur un modèle objet, celui-ci permet de manière tout à fait élégante de bénéficier du support de l'héritage pour spécialiser des plans et de permettre de ne redéfinir pour un VPN donné, que les traitements spécifiques. Le modèle de confinement (*nesting*) est, quand à lui, idéal pour garantir un contrôle et une allocation stricte des ressources de communication.

L'approche *Genesis* a des ambitions plus larges que celles définies dans l'approche *Tempest* définie plus loin dans ce chapitre, notamment en terme de support du cycle de vie d'un VPN avec un degré de programmabilité supérieur. Ceci a une contre-partie aujourd'hui, l'absence d'une implémentation réelle pour *Génesis*. En fait, il serait parfaitement concevable d'utiliser l'architecture de *Tempest* pour implanter les concepts de Genesis au dessus.

3.5 Tempest

Tempest [MER 98, ROO 98] vise à fournir une architecture logicielle supportant d'une part l'ouverture du plan de contrôle des commutateurs ATM au travers d'une interface de programmation ouverte et indépendante de tout constructeur, d'autre part de fournir au sein de cette architecture des mécanismes permettant de partitionner les ressources physiques d'un commutateur offrant ainsi un support pour des plans de contrôle multiples, standards ou fournis par des entités tierces. L'architecture Tempest repose sur cinq composants principaux :

- une interface de contrôle de commutateur générique ;
- une entité de partitionnement de ressources de commutateur ;
- une interface de supervision indépendante de toute approche standard de supervision ;
- une architecture de *bootstrapping* de l'environnement permettant le déploiement d'un réseau virtuel de base servant à la mise en place de l'architecture distribuée ;
- un service de création de réseaux privés virtuels.

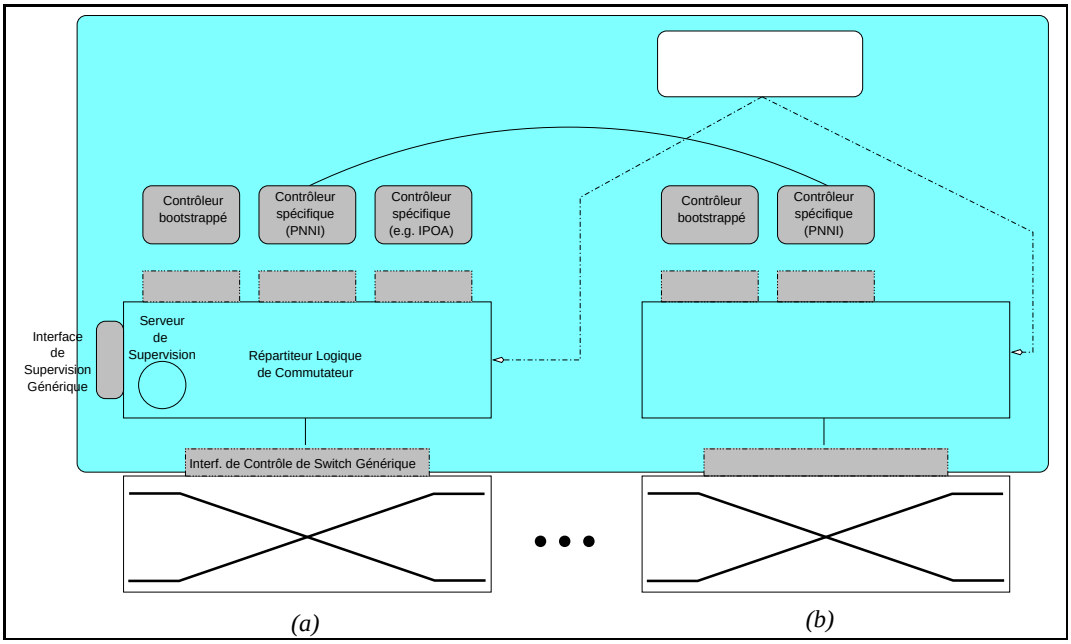

FIG. 9 – *Architecture globale de l'approche Tempest*

La figure 9 illustre les principaux composants de l'architecture globale mise en place dans *Tempest*. L'ensemble des composants sont représentés par des objets CORBA au sein d'un environnement distribué (DPE[27]).

3.5.1 L'interface de contrôle de commutateur

Cette interface implémente une architecture client/serveur et définit les opérations offertes à l'interface d'un commutateur ATM. Cette interface est utilisée entre le répartiteur et le commutateur ainsi qu'entre le répartiteur et les *switchlets* ou entités de contrôle partagé qui voient le répartiteur comme un sous-ensemble de commutateur. Cette interface offre des fonctionnalités sensiblement identiques à celles de GSMP à savoir :

- opérations de configuration : état du commutateur, états des ports, espaces de numérotation des ports, constructeur, ... ;
- opérations de réservation et d'affectation de ressources : définition de contextes de ressources pour un VPN donné. Les paramètres du contexte sont des paramètres de QoS tels que les délais inter-cellules, les délais max de transfert de cellules, les ratios de perte, le débit minimal, le type d'algorithme de contrôle de flux, ... ;
- opérations de connexion qui permettent la création/destruction de chemins et/ou de canaux virtuels;

27. DPE : Distributed Processing Environement

- opérations de collecte d'informations statistiques (nombre de paquets reçus, émis, ...) sur des canaux et/ou des chemins virtuels et/ou des ports ;
- des opérations d'alarmes qui permettent à un client de souscrire à certains types d'alarmes émises par le commutateur.

Cette interface est en cours de standardisation à l'IETF et elle est soutenue par le consortium *MSForum* présenté plus loin dans ce chapitre. De plus, les résultats du projet *Tempest* sont actuellement repris sous forme industrielle dans le cadre de la société CPlane[28].

3.5.2 L'entité de partitionnement de commutateur

Sous la forme d'un serveur, ce composant comporte le cœur de l'architecture *Tempest*. Il a en effet la responsabilité de partitionner les ressources d'un commutateur (espace de nommage, bande passante, buffers) entre tous les réseaux virtuels qui transitent par son nœud (un serveur par commutateur) et d'offrir aux entités de contrôle de chaque VPN, une interface de contrôle générique (ici ARIEL). Ce partitionnement est principalement réalisé par une fonction de filtrage des appels provenant des *switchlets*. Le filtrage se fait sur les numéros de ports, de VC et de VP attribués à une partition.

Les ordres de partitionnement sont donnés via l'interface de supervision implantée dans le répartiteur. Ce répartiteur est implanté dans l'architecture au sein du DPE sous forme d'un serveur d'objets. Le contrôle d'admission de connexions, très prisé dans le monde ATM, est réalisé par les architectures de contrôle de chaque réseau privé virtuel. Le répartiteur fournit à ces architectures, via l'interface de contrôle générique, toutes les information nécessaires à l'exécution de cette fonction.

3.5.3 Le déploiement initial de l'architecture

L'architecture fournit un mécanisme de *bootstrapping* afin de permettre aux composants de base de se mettre en place sur le réseau. Un tel mécanisme est nécessaire car l'architecture étant utilisée pour créer des réseaux virtuels et véhiculer la signalisation, tant que celle-ci n'existe pas, aucune signalisation ne peut être utilisée pour l'instancier.

Pour palier ce problème, un réseau virtuel par défaut est mis en place pour permettre les communications initiales entre composants de la plate-forme. Celui-ci est basé sur IP over ATM et permet la mise en place d'un canal de signalisation entre les composants de base de l'architecture (répartiteurs, environnement de création de réseaux virtuels, ...). En réalité, ce réseau virtuel est utilisé pour mettre en place le DPE CORBA au travers duquel les composants vont communiquer.

3.5.4 L'environnement de création de réseau privé virtuel

L'environnement de création de réseau est un service CORBA de l'architecture dont l'objectif est, sur réception d'une demande provenant soit d'un usager, d'une architecture de contrôle ou d'un opérateur de supervision, d'instancier un réseau privé virtuel et de progager la mise en place jusqu'aux répartiteurs qui transmettrons les ordres aux commutateurs eux-même. Les paramètres d'une demande de création de VPN sont :

- type de contrôle demandé (PNNI, IPOA[29], ...). Ce paramètre peut être nul. Dans ce cas l'entité qui émet la requête fournit elle-même un contrôleur centralisé qui sera lié à toutes les interfaces de *switchlets* des commutateurs traversés par le VPN ;
- paramètres de QoS ;
- adresses source et destination du VPN.

Dans son procesus de décision, l'environnement de création s'appuie sur un courtier qui connait tous les répartiteurs du réseau et qui propose une séquence de répartiteurs à utiliser pour construire la route. Le processus de création invoque ensuite la création de *switchlets* sur les différents répartiteurs sélectionnés.

28. `http://www.cplane.com`
29. IPOA: Ip Over ATM

3.5.5 Études de terrain

Dans le cadre de *Tempest*, une architecture de contrôle spécifique appelée *Hollowan* a été développée au dessus de l'infrastructure. Elle comprend des *switchlets* spécifiques, un gestionnaire de connexions, un agent utilisateur pour chaque station de travail et un support de courtage pour la découverte de services. Les performances obtenues pour la mise en place de connexions sont équivalentes à celles obtenues à l'aide d'implantations standards d'architectures de contrôle. D'autres architectures de contrôle telles que RSVP ou UNITE ont également été prototypées dans l'architecture.

Plusieurs études ont été menées sur la conception et la réalisation d'architectures de contrôle pour des services et des applications spécifiques. Parmi ces études, on note la création d'une architecture de contrôle pour un environnement de vidéo-conférence optimisant les ressources de communication utilisées, ainsi qu'une architecture de contrôle pour de l'ATM mobile.

3.5.6 Résumé

Tempest propose une architecture novatrice pour faciliter la programmabilité des réseaux ATM en permettant la mise en place de réseaux privés virtuels ayant leurs propres architectures de contrôle et se partageant les ressources physiques des commutateurs du réseau. En plus de l'interface de contrôle générique, l'apport principal de l'approche est certainement l'entité de partitionnement de commutateur, élément central de l'ouverture de l'architecture.

De nombreux travaux se poursuivent autour de cette architecture. D'une part, une migration industrielle est en cours sur le cœur de l'architecture via la société *CPlane* et des composants sont en cours de standardisation au sein du consortium *MSForum*[30](*Multi-Service Switching Forum*). Des travaux amont sont également poursuivis autour notamment des agents mobiles utilisés pour de la supervision des composants de contrôles au sein d'un VPN. Des travaux sur la gestion de connexion sont également poursuivis.

3.6 L'API Parlay

Crée en avril 1998, *Parlay*[31] est un consortium d'industriels fondé initialement autour de BT, Microsoft, Nortel Networks, Siemens et Ulticom. Aujourd'hui se sont joint à cette initiative AT&T, Cegetel, Cisco, Ericsson, IBM, et Lucent. L'objectif principal de *Parlay* est de définir une API réseau permettant de séparer dans un réseau les composants génériques de services des services spécifiques en fournissant un ensemble de facilités de programmation dans les niveaux inférieurs (réseau, composants génériques). Cette approche favorise la séparation entre services et opérateurs, ouvrant ainsi la voie au développement rapide et compétitif de nouvelles applications et services sur les réseaux. L'objectif principal est notamment de fournir aux développeurs d'applications nécessitant des accès aux ressources de communication (téléphone, IP, GSM, ...) une API unifiée pour les utiliser.

La figure 10 issue de [PAR 99], illustre cette évolution. La première version de l'API se focalise sur une API pour le contrôle d'appels, la gestion de messageries ainsi que sur la sécurité. L'API en elle même est divisée en deux composantes :

- une API de *framework* indépendante de tout service. Cette API définit les fonctions d'accès et de manipulation de services. Les fonctions comprises dans cette API sont un support de service d'authentification mutuelle, des fonctions de découverte de services, un environnement de gestion de notifications (fonctions du framework + fonctions à implanter par les applications qui utilisent le framework) et finalement des fonctions de gestion et de contrôle génériques ;
- une API de services.

Cette approche se révèle moins catégorique que les approches P1520 ou *Tempest* en ne militant pas pour une ouverture totale des fonctions du réseau en particulier celle du plan de signalisation. Ici, une approche intermédiaire est adoptée en ce sens que l'API n'offre que les fonctions nécessaires aux applications d'entreprise et permet aux opérateurs de garder la maîtrise des autres fonctions. De plus, *Parlay* ne se préoccupe pas de la définition des interfaces de programmation des ressources ce qui est fait dans P1520. Pour l'accès à ces dernières *Parlay* s'appuie sur les APIs existantes.

30. `http://www.msforum.org`
31. `http://www.parlay.org`

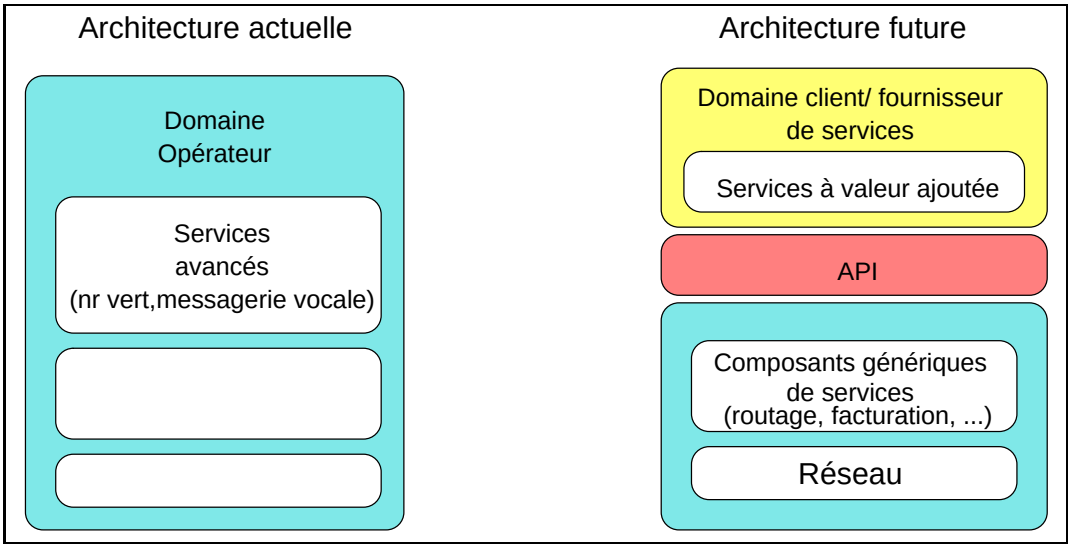

FIG. 10 – *L'architecture PARLAY*

Une extension de l'API *Parlay* est en cours. Son objectif est d'offrir des librairies supplémentaires visant à unifier l'accès aux services relatifs au sans fil (mobile), à IP ainsi que ceux de téléphonie permettant aux applications de combiner ces différentes techniques pour leurs besoins.

Aujourd'hui les spécifications de la phase 1 sont disponibles, une implantation de référence ainsi qu'un kit de développement s'appuyant sur Java, COM et CORBA sont en cours de réalisation.

3.7 Autres projets

Les autres projets autour des réseaux programmables sont :

- *Xbone*[32]
- VNS[33] (*Virtual Network Services*)
- Darwin[34] [CHA 98, TAK 99]
- GeoPlex d'AT&T[35]
- les travaux de standardisation du *MSForum* (*Multiservice Switching Forum*[36])

Ces projets et les résultats associés seront présentés dans une version ultérieure de ce rapport.

3.8 Résumé

Le concept de réseau programmable, apparu en 1996, s'est largement développé ces deux dernières années. Les spécifications ont énormément progressé via la création de deux consortiums industriels. Il est évident que ce concept va, dans les prochaines années se développer fortement dans l'industrie et que des interfaces de programmation pour des équipements de commutation ainsi que pour des routeurs devraient voir le jour dans des offres commerciales au plus tard d'ici deux ans.

4 Les réseaux actifs

Plusieurs publications dressent un état de l'art et énumèrent les différents travaux en cours dans le domaine des réseaux actifs [TEN 97, CAL 98, PSO 99, CAM 99c]. Cette section fournit comme les publications précédentes une vue d'ensemble des approches mais détaille plus chacune d'entre elle et l'illustre au travers d'un exemple simple, souvent fourni dans les approches : le protocole *ping*. En raison de l'activité intense autour

32. `http://www.isi.edu/touch/pubs/gi98/`
33. `http://www.cs.cmu.edu/~hzhang/VNS/`
34. `http://www.cs.cmu.edu/~darwin/`
35. `http://www.geoplex.com/`
36. `http://www.msforum.org`

des réseaux actifs et de la naissance quasi mensuelle de nouvelles propositions d'architectures, cette section est certainement la section du rapport susceptible d'évoluer le plus rapidement au fil des versions de ce document.

4.1 SmartPackets (BBN)

L'approche *SmartPackets* développée chez BBN [SCH 99] est une approche intégrée motivée par la croissance exponentielle de la capacité de traitement disponible sur les nœuds des réseaux, la nécessité de réduire le plus possible le traffic de gestion qui a tendance à exploser avec le nombre de noeuds et finalement le besoin de décentraliser les fonctions de gestion afin de soulager les stations de supervision.

Afin de répondre à ces besoins, *SmartPackets* utilise intelligemment le paradigme de réseau actif pour décentraliser la gestion. Le principe de l'approche est le suivant: un paquet de supervision est un programme véhiculé dans un paquet actif. La principale restriction sur ces paquets est la taille maximale fixée à 1Ko afin que ceux-ci tiennent dans une trame Ethernet. Dans les différents nœuds du réseau, l'approche propose une architecture d'accueil et d'exécution de ces fonctions de gestion.

Le format de paquets SmartPackets est donné dans la figure 11. On y retouve l'entête de trame IP, l'encapsulation dans le standard de paquets actifs ANEP (voir section 5) et finalement le paquet SmartPacket. Celui-ci est composé d'un numéro de version, d'un type, un contexte et des données véhiculées. Les différents types possibles sont:

- *Program*: le paquet contient un programme envoyé vers un agent ;
- *Data*: le paquet contient des données renvoyées vers la station de supervision depuis un agent ;
- *Error*: le paquet véhicule une erreur d'un agent vers un superviseur ;
- *Message*: permet à un agent ou un gestionnaire d'envoyer des données qui ne sont ni un programme, ni une réponse à un envoi et une exécution de programme.

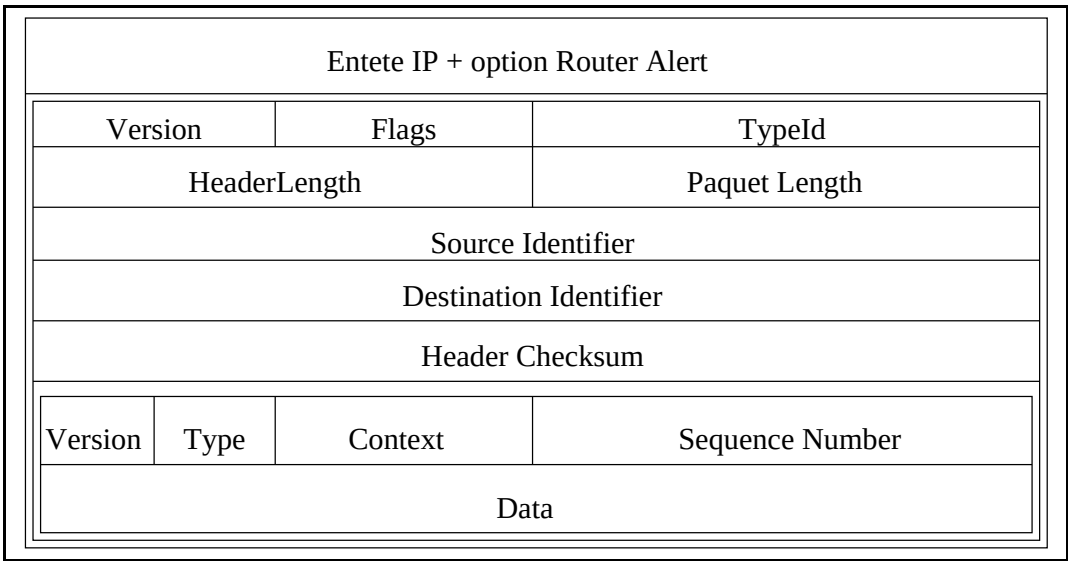

FIG. 11 – *Le paquet SmartPackets*

Permettre la programmation de fonctions de gestion requiert l'utilisation d'un langage de programmation. *SmartPackets* définit son propre langage de programmation de paquets. Appelé *Sprocket*, celui-ci est un sous-ensemble de C++ (simplifié et sans pointeur) étendu par des primitives d'accès et de manipulation d'informations de gestion issues d'un agent SNMP (Get, Set, Get-next) ainsi que par des primitives de manipulation de paquets actifs. La figure 12 illustre l'utilisation de ce langage.

L'exemple comporte un programme chargé de collecter sur un agent toutes les interfaces, les adresses IP associées ainsi que le MTU (Maximum Transfer Unit). Les lignes 1 à 6 comportent les déclarations des variables utiles au traitement. À la ligne 8, un premier appel est fait à la MIB-II au travers de la fonction `num_ifaces()` qui renvoit le nombre d'interfaces déclarées dans la machine. Ce nombre est inséré dans le paquet de réponse au travers de la commande `pkt.add()` à la ligne 9. Il sert également à l'itération (lignes 10 à 16) dans laquelle

```
1 Main()
2 {
3   array of address addr;
4   packet pkt;
5   unsigned8 num_interfaces;
6   unsigned8 index;
7
8   num_interfaces = num_ifaces();
9   pkt.data_append(num_interfaces);
10   for (index=1; index<=num_interfaces; index++)
11    {
12      addr.set_size_of_dimensions(num_addresses(index));
13      get_addresses(addr);
14      pkt.data_append(index);
15      pkt.data_append(addr);
16      pkt.data_append(get_iface_mtu(index));
17    }
18  pkt.send();
19}
```

FIG. 12 – *Un exemple de code Sprocket*

pour chaque interface, l'ensemble des adresses ainsi que le MTU sont récupérés et ajoutés au paquet. Celui-ci est émis via la méthode `send()` (ligne 18) une fois rempli.

Tout code *Sprocket* est compilé en un assembleur spécifique appelé *Spanner*. La principale différence avec les assembleurs classiques est que celui-ci ne fourni pas d'instruction d'accès mémoire direct mais uniquement via des variables déclarées ainsi qu'une pile pour des raisons de portabilité.

Tout nœud capable de traiter des *SmartPaquets* dispose d'une machine virtuelle qui démultiplexe les paquets ANEP et exécute le code *Spanner*. L'authentification et la confidentialité sont assurées pas un cryptage des champs non mutables d'un paquet ANEP couplé à l'utilisation d'un système de clefs publiques/privées. Les fonctions d'autorisation sont implantées par des listes de contrôle d'accès par agent. Ces listes définissent pour chaque client connu, les fonctions de manipulation de MIB possibles ainsi que la liste des objets de celle-ci accessibles par le client et pour chaque objet, les opérations précises autorisées.

L'approche *SmartPackets* est l'une des premières approches actives publiées. Dédiée à la supervision elle est particulièrement intéressante car elle jusitifie totalement l'utilisation de l'actif pour des fonctions de supervision et démontre l'apport d'un langage de programmation de paquets dédié à la fonction visée, concept qui sera repris plus tard dans les réseaux actifs via les langages spécifiques de domaines pour des fonctions autres que la supervision. Cependant, l'approche prototypale développée souffre d'un certain nombre de limites. Il est notamment impossible pour celles-ci de laisser des états dans un nœud pour des fonctions futures interdisant ainsi des chaînes de traitements qui peuvent se révéler intéressantes en gestion. La seconde est la limite forte sur la taille de programmes (bien que cela ait été un pré-requis dans l'approche), limitant les fonctions déportées à des traitements restreints. Finalement, très peu d'expérimentations et aucun résultat chiffré sur l'apport de ce type de délégation n'a été reporté, ce qui est très dommageable et regretable car ce type de résultats aurait peut-être pu démontrer de manière indiscutable l'intérêt de l'approche.

4.2 Switchware

Switchware [SMI 98, ALE 98b] regroupe l'ensemble des activités de l'université de Pennsilvanie autour des réseaux actifs. Les travaux menés dans ce projet visent à définir une architecture homogène incluant différents paradigmes tels que le nœud actif, le paquet actif ainsi que la sécurisation d'un réseau actif. Les aspects langage de programmation de réseau actif sont également au cœur des travaux de *Switchware*.

L'architecture générale d'un nœud *Switchware* est donnée dans la figure 13. Celle-ci est basée sur 3 couches. La première représente un niveau de système d'exploitation et d'environnement d'exécution de base. La seconde offre une facilité d'enrichissement du système d'exploitation par des extensions actives (fonctions offertes aux paquets actifs sur un nœud). La troisième couche représente la fonction de traitement de paquets actif com-

portant chacun du code exécuté sur chaque nœud traversé. L'architecture *Switchware* repose sur 3 principaux composants. Ces composants sont : ALIEN, PLAN et SANE.

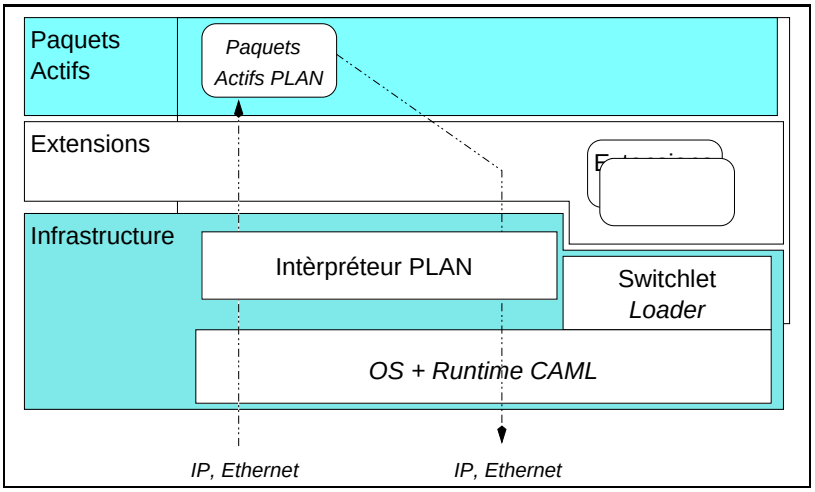

FIG. 13 – *L'architecture Switchware*

4.2.1 ALIEN

ALIEN [ALE 98a, ALE 99b] constitue l'un des fondements du projet *Switchware*. ALIEN explore l'architecture d'un nœud actif comportant à la fois des extensions actives chargées par l'opérateur du réseau et un support pour des paquets actifs comportant chacun le code associé à leur traitement. Le langage de programmation des extensions ainsi que des paquets actifs est OCAML (*Objective CAML*[37]) étendu avec des primitives de manipulation de paquets actifs (accès, envoi, réception). L'architecture d'un nœud repose sur 3 niveaux:

- un système d'exploitation standard (Linux) comportant un interpréteur OCAML ;
- une interface de nœud offrant un accès au réseau et au système d'exploitation ainsi qu'un composant de chargement de code permettant de charger des extensions actives ;
- des librairies de routines utiles.

Cette architecture de nœud permet l'expérimentation d'une approche nœud actif au travers du chargement d'extensions actives. Ces extensions ne sont pas mobiles et offrent des services aux paquets actifs qui transitent sur le réseau. L'architecture d'ALIEN permet également l'expérimentation d'une approche paquet actif, via le support d'envoi et de réception de paquets actifs dans un format ANEP (voir section 5). L'intérêt d'ALIEN est d'offrir dans un environnement standard, toutes les fonctions de base d'un nœud requises pour les fonctions de niveau supplémentaire.

4.2.2 PLAN & PLANet

PLAN (*Packet Language for Active Networks*) [ALE 97c, HIC 98, HIC 99b] constitue une approche complémentaire d'ALIEN du point de vue langage de programmation de réseau actif et s'appuie sur l'architecture d'ALIEN pour la construction d'un nœud actif. En effet, PLAN n'exploite pas un langage de programmation générique mais définit son propre langage inspiré du langage fonctionnel *Scheme*[38] et CAML.

Le langage de programmation de paquets de PLAN est un langage fonctionnel simple basé sur le λ-calcul typé ayant un haut niveau d'abstraction. La communication entre nœuds actifs se traduit dans un programme par une évaluation de fonction à distance et donne lieu à la génération et l'envoi de paquets actifs au travers du réseau. L'intérêt principal de disposer d'un langage restreint réside dans les limites que l'on peut placer sur celui-ci afin de maîtriser les actions réalisables par les paquets dans les nœuds. Cela permet notamment de restreindre la sécurité placée sur le code dans les paquets. De plus, cela permet de fournir un ensemble de primitives dédiées à l'évaluation à distance et au traitement des paquets dans le langage.

37. `http://ocaml.inria.fr`
38. `http://www.cs.indiana.edu/scheme-repository/SRhome.html`

Le format des paquets actifs de PLAN est basé sur la notion de *chunck*[39], littéralement bout de code. Un tel paquet comporte le code à exécuter sur le nœud distant, le nom de la première fonction de ce code à invoquer ainsi que les paramètres effectifs de cette fonction (qui sont en fait les données utiles véhiculées sur le réseau). Ce format est illustré dans la figure 14.

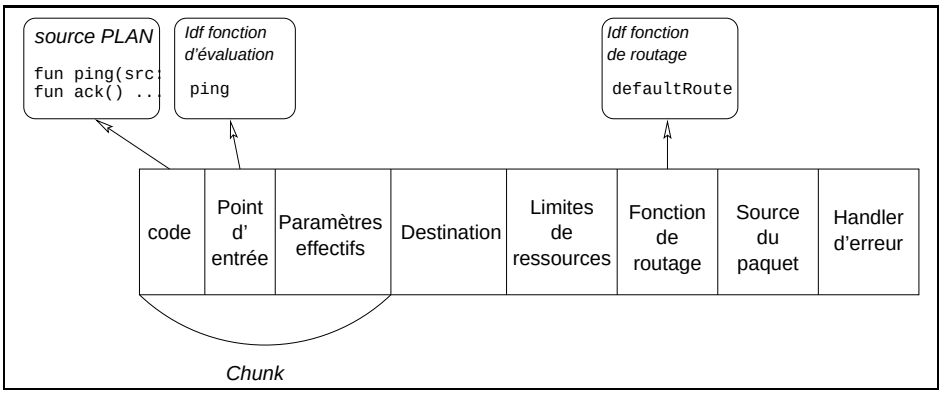

FIG. 14 – *Le paquet PLAN*

Un mécanisme de limitation des ressources basé sur un compteur est également disponible. Le routage est basé sur une évaluation par paquet. En effet, tout paquet actif comporte l'identification de la fonction de routage à appliquer sur celui-ci. En cas d'erreur, un mécanisme de traitement peut être appelé depuis n'importe quel nœud sur la source du paquet. La fonction implantant ce mécanisme constitue l'un des paramètres d'un paquet PLAN.

L'évaluation à distance (envoi de paquet) est réalisée par tout nœud lors de l'évaluation d'une fonction d'appel à distance (`OnRemote`) ou d'appel à un voisin (`OnNeighbor`). L'appel récursif de fonctions autres que celles d'évaluation à distance est interdit dans PLAN afin de garantir la terminaison de l'évaluation dans un nœud. La figure 15 illustre les paramètres des appels d'évaluation à distance.

```
OnRemote(Quoi?,Où?,Combien?,Comment?)

OnNeighbor(Quoi?,Où,Combien?);
```

FIG. 15 – *Les paramètres des méthodes d'évaluation à distance dans PLAN*

La fonction d'évaluation à distance `OnRemote` comporte quatre paramètres qui sont le nom de la fonction à invoquer et les paramètres effectifs (Quoi), l'identificateur du nœud sur lequel on demande l'évaluation (Où), les ressources associées (Combien) ainsi que le mécanisme de chemin retenu pour y aller (Comment). Notons que la fonction n'est pas évaluée dans les nœuds intermédiaires qui sont parcourus pour aboutir à la cible. La fonction `OnNeighbor` ne nécessite pas de paramètre indiquant une fonction de routage, le nœud adressé se retrouvant obligatoirement sur le même brin que le nœud source. Une illustration de ces appels est donnée dans l'exemple du `ping` détaillé dans la section suivante.

Ce langage est parfaitement adapté à la spécification de petits programmes définis comme des micro-protocoles insérés dans des paquets. Deux implantations de PLAN sont aujourd'hui disponibles. L'une est réalisée en OCAML, la seconde est fournie en Java.

PLANet [HIC 99b] est un réseau d'expérimentation couplant les apports d'ALIEN et de PLAN. Les routeurs PLANet offrent un environnement d'exécution pour des paquets PLAN et les services offerts par cet environnent d'exécution sont extensibles grâce au concept de *Switchlet* ou d'extensions actives définies dans ALIEN. Les paquets PLAN sont étendus avec deux champs supplémentaires dans PLANet. Ces champs sont : un identificateur de session et un identificateur de flux. Ils permettent d'associer, d'une part une application à un paquet(champ session) et, d'autre part un flux. L'adressage dans PLANet repose sur des adresses de 48 bits. La configuration de routage est maintenue par un protocole de routage applicatif centralisé organisant les nœuds en une hiérarchie comprenant un maître et `n` esclaves [MOO 97a].

39. `http://www.cis.upenn.edu/~switchware/papers/planchuncks`

4.2.3 Le Ping

La figure 16 comporte l'exemple du *Ping* codé dans le langage PLAN. Celle-ci est définie par deux fonctions : une fonction `ping` et une fonction `ack`. La fonction `ping` définit le corps du traitement. On y retrouve la demande d'évaluation à distance `OnRemote` utilisée pour exécuter `ping` sur la cible et `ack` sur la source depuis la cible.

```
fun ping (src:host, dst:host) : unit =
if (not(thisHostIs(dst))) then
  OnRemote (|ping|(src,dst), dst, getRB(), defaultRoute )
else
  OnRemote (|ack|(), src, getRB(), defaultRoute )

fun ack() : unit = print("Success")
```

FIG. 16 – *Le ping en PLAN [ALE 98b]*

La spécification complète du langage PLAN est donnée dans [KAK 97]. Sa spécification formelle sous forme de machine abstraite et de règles de réécriture des types est donnée dans [KAK 99b, KAK 99a]. Avec l'environnement de programmation, différents tutoriels et guides de programmation sont fournis. Ces guides sont : un tutoriel sur l'environnement et le langage [KAK 97], le manuel de programmation [MOO 97b], le manuel de programmation de services actifs déployés sur les nœuds [HIC 97a] et le manuel d'utilisation du routeur PLAN [HIC 97b] (ici dans sa version Java). Les mêmes documents existent pour l'implantation OCAML pour laquelle les versions de PLAN sont plus récentes.

4.2.4 SANE

La sécurisation d'un nœud ainsi que des paquets et extensions actives est le résultat des travaux menés dans SANE (*Secure Active Network Environment* [ALE 98c, HIC 99a, ALE 99a]). Ici, un mécanisme en couches est défini. Le niveau le plus bas inclut la couche physique du routeur actif ainsi que la couche de système d'exploitation. Ce niveau a pour objectif de garantir que le système est chargé dans un état correct. Le mécanisme de *bootstrap* développé dans SANE s'appelle AEGIS et garantit que le système est correctement lancé en ne permettant le chargement dans la machine physique que du code (en fait l'OS) signé d'un tiers auquel on fait une totale confiance. Ceci permet de charger l'OS ainsi que l'environnement d'exécution de base. Pour cela, une extension du BIOS d'un PC a été réalisée afin que l'initialisation passe par l'environnement AEGIS.

Dans une seconde étape, pour le chargement des extensions actives, cette architecture offre des mécanismes d'authentification de ces extensions. L'interface qu'offre ces extensions aux paquets actifs étant clairement définie, ceux-ci ne peuvent la contourner. Les paquets actifs eux-mêmes ne nécessitent pas, en raison des limitations faites sur les programmes contenus, de sécurisation particulière. Ce niveau est classifié comme niveau **public**. Sur ces paquets actifs, PLAN fournit des garanties importantes pour la sécurité notamment celle que tout paquet se termine (pas de boucles infinies) et qu'un paquet ne peut aller fouiner dans des données d'un autre paquet et ce grâce au typage fort. La limite de ressources garantit cette terminaison à l'échelle du réseau. Ces fonctions sont offertes dans ALIEN [ALE 99a].

Finalement un mécanisme de contrôle d'accès à des extensions actives privilégiées avec authentification des paquets entrants permet de restreindre le champ d'accès de chaque paquet [HIC 99a].

4.2.5 Intérêts et limites

L'approche *Switchware* est certainement l'approche la plus complète dans le domaine des réseaux actifs aujourd'hui. Elle propose un langage, un format de paquets, une architecture de sécurité et atteint des performances intéressantes (60 Mbps sur un lien de 100Mbps dans une approche de nœud actif [ALE 98a]). Cependant, certains points de l'approche sont discutables, notamment l'impossibilité pour un paquet actif de déposer des états dans un nœud pour des paquets futurs et l'absence de cache de code nécessitant l'envoi dans tout paquet actif du code de traitement associé et, lors de chaque réception dans un nœud, du rechargement de ce code. Ceci entraîne un coût énorme sur le traitement des paquets. Un rapprochement avec les concepts d'ANTS pour ces aspects serait souhaitable mais entraînerait une modification des formats des paquets PLAN.

4.3 PLAN-P

4.3.1 Principes

PLAN-P[40] (*Programming Language for Active Networks and Protocols*) [THI 98a, THI 99] est une approche proposée à l'IRISA dans le projet COMPOSE. PLAN-P est principalement une extension du langage PLAN assortie d'un environnement d'exécution optimisé. PLAN-P est plus orienté vers la réalisation de protocoles dans un nœud que PLAN qui est plus orienté vers la réalisation de code de capsule actives.

L'extension principale de PLAN-P sur PLAN est la notion de protocole [THI 98b]. Un protocole est un programme qui est téléchargé sur un équipement et qui n'est pas mobile. Un protocole PLAN-P est défini par un programme et comprend un ou plusieurs canaux. Un canal est caractérisé par un état ainsi qu'une fonction de traitement appliquée sur chaque paquet dans le canal. Les paquets arrivant dans un canal sont sélectionnés par le nœud suivant le type déclaré par un canal. Les canaux ont des états ainsi que les protocoles. Pour ces composants, les extensions au langage PLAN sont restreintes offrant des primitives de déclaration de protocoles, canaux, paquets et traitements.

Tout routeur PLAN-P possède un interpréteur. Celui-ci est généralement dans le noyau au même niveau qu'IP. Les composants de cette approche offrent, grâce à l'utilisation d'un compilateur JIT[41], d'excellentes performances dans le traitement des paquets par des protocoles téléchargés (proches de celles d'un programme C). L'implantation du nœud a été réalisée directement sur IP permettant aux protocoles d'en influencer les algorithmes de routage par défaut.

4.3.2 Avantages et limites

PLAN-P complète parfaitement l'approche PLAN de l'université de Pennsylvanie en fournissant un langage pour la conception de protocoles dans les nœuds ainsi qu'un environnement d'exécution performant au niveau IP. Il serait intéressant de coupler plus fortement ces deux approches en utilisant PLAN pour des paquets actifs et PLAN-P pour les protocoles de base sur les nœuds. L'expérimentation pourrait alors adresser des aspects tels que la composition de services ce qui n'est pas fait pour le moment. De plus, l'architecture PLAN-P ne dispose pas aujourd'hui, ou du moins celui-ci n'est pas documenté, d'un mécanisme de déploiement des programmes dans les nœuds. Une étude sur ce point devrait être menée afin de fournir une plate-forme complète.

4.4 Active IP

4.4.1 Principes

Active IP est un prototype de réseau actif développé en 1996 au MIT [WET 96]. Ce prototype est basé sur l'utilisation des extensions IP pour le transport du code à appliquer à chaque paquet actif. Tout routeur supportant la fonction de routeur actif déclenche sur réception d'un paquet actif le traitement de l'option IP qui contient un code TCL[42] (*Tool Command Language*) étendu avec des primitives de manipulation de paquets IP ainsi que des fonctions de base d'accès aux ressources du nœud (adresse, horloge, ...). Ce code peut modifier le contenu des données du paquet IP, reconstruire un ou plusieurs paquets et les émettre. Le transport des routines actives se fait dans une option d'IP dont le contenu est donné dans la figure 17.

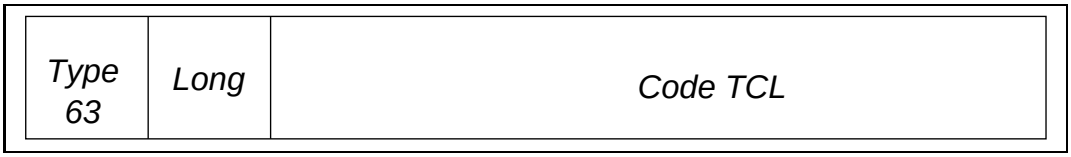

FIG. 17 – *Le paquet Active IP*

L'option comporte un type particulier (63), sa longueur ainsi que le code TCL à exécuter sur les nœuds. Une seconde option est définie. Son utilisation permet la création d'un paquet de demande à un routeur pour l'interroger sur les langages actifs supportés. Afin que tous les routeurs tentent de traiter la partie active du paquet IPv4, l'option *Router Alert* et également positionnée. Si un routeur ne supporte pas l'option active, il

40. `http://www.irisa.fr/compose/plan-p`
41. JIT: Just In Time compiler.
42. `http://www.tclconsortium.org/`

transmet le paquet suivant son routage par défaut. Les primitives offertes dans le langage de programmation de paquets sont de trois types :

- fonctions d'accès à l'environnement du nœud : demande de l'adresse IP, date, mtu, domaine, ...
- fonctions de manipulation de paquets : accès à et modification des données d'un paquet, adresse source, adresse destination, longueur, protocole encapsulé et ttl ;
- fonctions de contrôle : envoi de paquets IP et UDP, abandon de paquet, réponse à un paquet et évaluation d'un paquet.

De plus, toutes les primitives de contrôle de Safe-TCL sont supportées.

4.4.2 Le ping

La figure 18 comporte le code d'un paquet qui implante un ping. Ce code est d'une simplicité extrême. Si le paquet a atteint la destination, une réponse est retournée à la source. Sinon, le paquet est routé de manière standard.

```
  if
[node]==[destination] then { reply_ip [data 0] discard }
```

FIG. 18 – *Le ping en Active IP [WET 96]*

4.4.3 Avantages et limites

L'avantage principal de cette approche a été son caractère novateur fournissant une plate-forme de base pour la preuve du concept. Si l'approche est intéressante, plusieurs limites ont été identifiées. Parmi elles on retrouve principalement, l'absence de partage de ressources dans un nœud et le fait que la nature paquet actif pure ne permet pas de maintenir des services dans des nœuds ce qui est dommageable pour des flux de données uniformes (chaque paquet va contenir le même code et le transférer à chaque fois).

4.5 ANTS

4.5.1 Les principes

Dans la suite des travaux sur *Active IP*, les chercheurs du MIT ont conçu et réalisé une nouvelle architecture de réseau actif appelée ANTS (*Active Node Transfer System*) [WET 98b, WET 99c, WET 99b, WET 98a]. Le principe de cette architecture repose sur la capacité offerte aux applications de déployer dynamiquement dans le réseau les services et protocoles qu'elles utilisent et ceci sur tous les routeurs actifs que traversent les flux associés. Pour cela, ANTS offre une architecture de nœud permettant un support de protocoles multiples ainsi qu'un mécanisme de déploiement dynamique de ces protocoles au sein du flot de données de l'application. Les composants principaux de l'architecture ANTS sont :

- le protocole : définit de manière conceptuelle le traitement à effectuer sur un flux donné. Un protocole est défini par un identificateur (signature MD5 [43] du code qui l'implante), un ensemble de classes qui en définissent le comportement (groupes de code) ainsi que les unités de distribution de ce code (capsules) ;
- la capsule : unité de base de la programmation du réseau. La capsule est utilisée d'une part pour véhiculer les données d'une application entre les nœuds actifs et, d'autre part pour transporter le code d'un protocole à déployer sur les nœuds du réseau.

  Une capsule de données comporte une adresse source et destination ainsi qu'un champ TTL identiques à IP, un champ de version d'ANTS, un identificateur de méthode déterminant de manière unique la primitive d'évaluation et le protocole associé, un champ comportant l'adresse du dernier nœud traversé ainsi que les données sur lesquelles le protocole référencé dans l'entête par la signature MD5 de sa méthode d'évaluation doit être appliqué. Ces composants sont illustrés dans la figure 20.

  Dans une capsule de transfert de code les données transmises représentent le *byte code* de l'implantation du protocole. Celle-ci peut être composée de plusieurs capsules ;

43. Message Digest

– le nœud actif: environnement d'exécution. Le nœud actif permet la réception, l'exécution et l'envoi de capsules suivant un modèle de traitement prédéfini (protocole sélectionné par l'application ayant émis la capsule). Tout nœud actif dispose en plus d'un routage par défaut.

Le nœud, dont l'architecture est illustrée dans la figure 19, offre des services de cache de protocoles actifs, des fonctions d'accès aux ressources de communication (canal, table de routage) au travers d'une API de contrôle et d'accès à l'environnement, ainsi qu'un service de stockage d'informations via son API de cache. Ce service est offert aux capsules leur permettant de laisser des données dans un nœud traversé pour les capsules suivantes de même type (protocole). Les services (API) offerts par un nœud aux capsules qui le traversent représentent une vingtaine de fonctions.

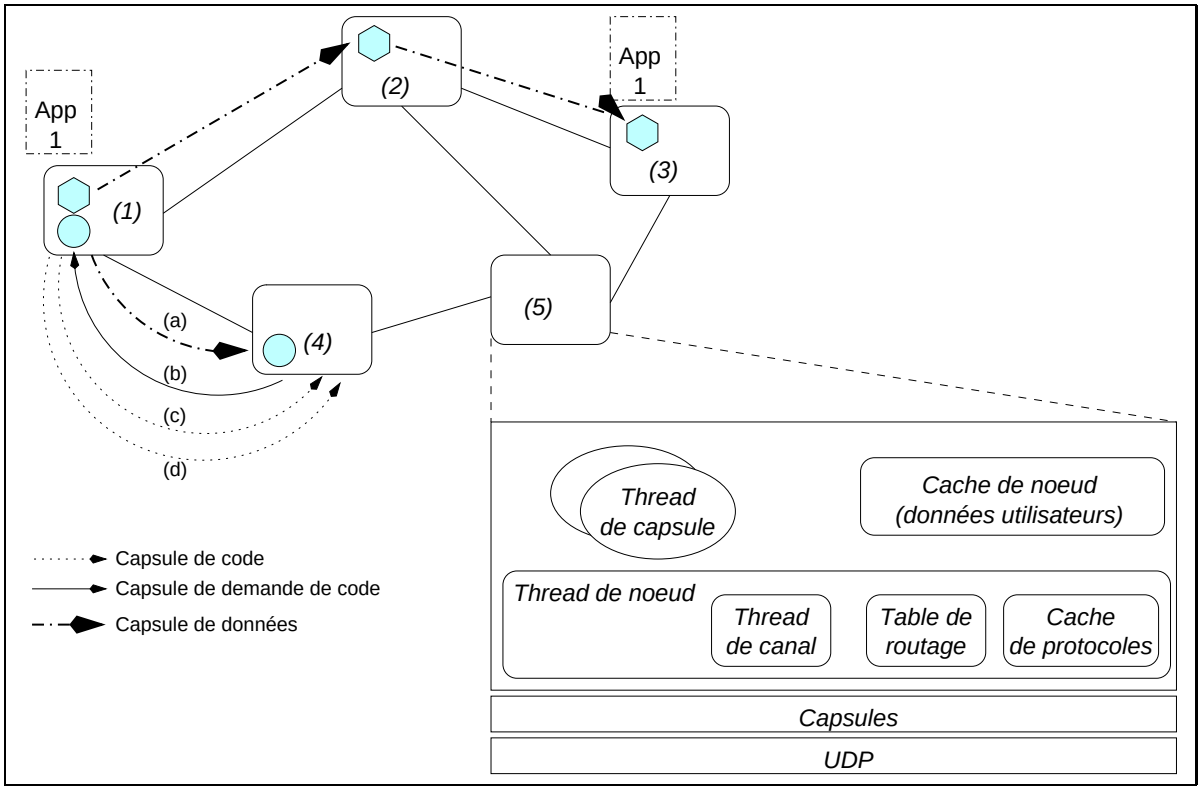

FIG. 19 – *L'architecture ANTS*

Le protocole définit l'unité d'isolation des traitements dans un nœud. Cela signifie que les traitements au sein d'un protocole partagent le même espace de nommage et le même espace mémoire et ne peuvent en aucun cas avoir accès à l'espace des autres protocoles ni générer des capsules dans d'autres protocoles. Ceci permet à des capsules d'un même protocole de s'échanger des données lors d'un passage dans un nœud mais isole les protocoles entre eux.

Le groupe de code définit l'unité de transfert, *i.e.* lors du chargement dynamique d'un protocole dans un nœud, seul le groupe de code (ensemble de capsules) auquel appartient la capsule actuellement traitée est demandé au nœud précédent. Ceci permet de s'assurer que tous les codes nécessaires sont disponibles afin de ne pas arrêter un traitement lié à un groupe, faute de disponibilité. La taille limite pour le transfert d'un groupe de code est de 16K dans l'implantation actuelle d'ANTS[44]. La capsule elle, est bien l'unité de traitement. C'est elle qui est évaluée à chaque passage dans un nœud.

Les trois principaux apports d'ANTS sont certainement son mécanisme de déploiement dynamique de code, sa gestion des protocoles au sein des nœuds et le support de cache de données offert aux capsules dans un nœud. Le premier rend l'architecture tolérante aux pannes et robuste vis-à-vis du passage à l'échelle en mettant en place un mécanisme de chargement à la demande depuis le nœud précédent. Lorsqu'une capsule arrive sur un nœud et demande l'exécution d'un protocole qui n'est pas présent, le nœud actif demande à l'émetteur de cette capsule de lui transmettre le code associé. Ceci garantit par récurrence, le déploiement effectif.

44. Ce qui permet de limiter l'impact du transfert de code sur le trafic du réseau, mais qui restreint la complexité et le type de services déployables.

Ce système est tolérant aux pannes car si un routeur n'est plus opérationnel, le routage dynamique va, si possible, rediriger les flux. Tous les routeurs sur le nouveau chemin vont pouvoir charger au fur et à mesure le code des protocoles associés à ces flux. Il est clair qu'une capsule ne peut pas changer de protocole en cours de route sinon ce mécanisme n'est plus valide. Le passage à l'échelle est également dû au mécanisme de chargement de protocole à la demande. D'une part seuls les nœuds qui ont besoin d'un protocole, le chargent. D'autre part, le chargement se fait systématiquement au plus près, *i.e.* depuis le nœud actif précédent.

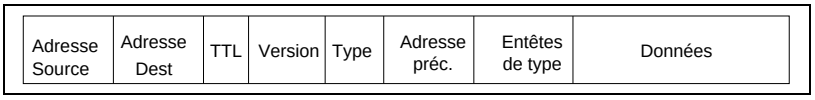

FIG. 20 – *La capsule ANTS*

Ce mécanisme de déploiement de code à la demande est illustré dans la figure 19 sur un réseau composé de cinq nœuds actifs. Le schéma comporte un protocole qui est déjà déployé sur le chemin d'un flux (*1,2,3*) ainsi qu'un protocole en cours de déploiement sur un autre chemin (*1,4,5,3*). Sur ce dernier cas, une capsule de données pour un nouveau protocole étant arrivée au nœud *4*, celui-ci demande au nœud précédent (*1*) de lui transmettre les capsules de code pour ce protocole. Une fois ce protocole déployé sur le nœud, la capsule de données ainsi que toutes les capsules suivantes pourront être traitées et transmises au nœud suivant qui demandera le chargement du protocole associé et ainsi de suite jusqu'à la cible.

Le second apport d'ANTS est son fonctionnement de nœud qui mélange l'approche paquet actif (les paquets transportent le code qui doit leur être appliqué) et l'approche nœud actif (le nœud peut accueillir des codes, les exécuter et les maintenir au delà de la durée de vie d'un paquet). Si conceptuellement, les données et le code transitent dans le même flux, ANTS ne nécessite pas que toute capsule de données inclut le code qui va permettre son traitement dans un nœud. Le mécanisme de cache de code fourni dans le nœud permet d'éviter cet excès en maintenant des codes sur des durées adaptées aux flux (au moins la durée d'une session moyenne pour un protocole donné). Ceci est particulièrement intéressant pour des flux importants sur lesquels toutes les capsules doivent subir le même traitement dans les nœuds (ex. des flux MPEG).

Le dernier intérêt de l'approche ANTS concerne la possibilité offerte à des capsules de déposer des états dans tout nœud traversé, états accessibles par les capsules de même type qui transiteront ultérieurement par le nœud.

Le principal inconvénient d'ANTS réside, comme pour de nombreuses approches de réseau actif, dans l'absence d'un modèle global de gestion et d'attribution de ressources pour des protocoles au niveau du réseau, mais également au niveau d'un nœud. Une limite également présente à ce jour concerne la sécurité du code et des usagers qui repose uniquement sur la signature de la méthode d'évaluation d'un protocole. Cependant, l'intégration des résultats issus des travaux sur les systèmes d'exploitation de nœud tels que Scout permettront de remédier à cette limitation.

Notons qu'il existe un mécanisme d'extension de nœud permettant à un administrateur d'étendre les fonctionnalités de celui-ci pour offrir des services avancés aux usagers sans que ceux-ci aient à déployer du code dans les différents nœuds du réseau. Ce mécanisme est statique et les extensions sont chargées lors du lancement du nœud dans la version actuelle.

Le prototype ANTS a été entièrement réalisé en Java. Il fonctionne aujourd'hui au dessus d'UDP, implante un mécanisme de routage dynamique et de nombreuses applications et protocoles ont été expérimentés sur cet environnement.

Une réalisation a également porté sur l'intégration d'IPv6 et d'ANTS. Ce travail [MUR 97] propose une réalisation d'un nœud actif en Java offrant un support pour le traitement de paquets IPv6 standards ainsi qu'un support pour ANTS, les capsules étant encapsulées dans des trames IPv6 standards via l'utilisation d'options. Visant à démontrer l'intérêt d'une approche active pour tester et déployer de nouvelles fonctionnalités d'IPv6, ce nœud ne consiste pas en la réalisation d'un environnement d'exécution directement sur une souche native d'IPv6 mais plutôt en une réalisation intégrée dans l'espace utilisateur.

4.5.2 Le ping

La figure 21 contient un sous ensemble du code de la capsule ping telle que fournie dans la distribution ANTS. La principale méthode de la capsule est la méthode `evaluate` qui est appelée à chaque nœud traversé. Dans le cas du ping, chaque évaluation dans un nœud commence par tester si la capsule est arrivée à destination. Si cela est le cas, la valeur `ping` est positionnée à vrai et dans le second test, la capsule est renvoyée vers la source. Si ce n'est pas le cas, alors le paquet continue à se propager vers la destination (méthode `routeForNode`).

Si un ping est en chemin vers la source, alors dès que celle-ci est atteinte, le ping est délivré à l'application (méthode `deliverToApp`). Cette capsule est intégrée dans un protocole, lui-même enregistré dans le réseau par toute application ping qui s'y déploie.

```
public class PingCapsule extends DataCapsule
{
  ...
  // devient vrai des qu'on a atteint la cible du ping
  public boolean ping = false;
  ...
    public Xdr encode()
      {
         Xdr xdr = super.encode();
         xdr.PUT(ping);
         return xdr;
      }

     public Xdr decode()
      {
         Xdr xdr = super.decode();
         ping = xdr.BOOLEAN();
         return xdr;
      }

     public boolean evaluate(Node n)
      {
        // Je suis à destination
        if ( n.getAddress() == getDst()) { ping = true;}
        else
        // Je suis en chemin vers la destination
        if (ping!=true) { return n.routeForNode(this, getDst());}

        //  Je suis arrivé à la source
        if (n.getAddress() == getSrc()) {return n.deliverToApp(this, dpt);}
        else
        // Je suis en chemin vers la source
        if (ping) { return n.routeForNode(this, getSrc());}
        return false;
      }
...
}
```

FIG. 21 – *La capsule ping en ANTS*

Pour programmer ANTS, la thèse de WETHERALL [WET 99c] (chapitre 3 : *The ANTS Toolkik*) fournit une bonne base. Les deux articles fournis dans la distribution [WET 97, WET 98b] donnent des exemples mais ne sont pas suffisants. Cependant, pour une utilisation extensive du prototype, une étude plus complète du code source qui est assez propre est parfois nécessaire. D'autant plus que certaines méthodes du prototype ne sont pas documentées et que certains points du routage (notamment dynamique) sont à modifier pour le faire tourner efficacement. De manière plus générale, programmer dans une approche capsule est assez perturbant au départ [45].

45. *NDLR: À force de toujours se demander où il est, le paquet finit par ne plus le savoir. et comme le souligne bien la citation d'Otto de Habsbourg: "Celui qui ne sait pas d'où il vient, ne sait pas où il va, car il ne sait pas où il est."*

4.5.3 Avantages et limites

ANTS offre une approche simple et compréhensible du concept de réseau actif. Son principal avantage réside dans son modèle de déploiement de code à la demande n'installant que des protocoles sur les nœuds qui en ont vraiment besoin à un moment donné. Basé sur les mécanismes de *forwarding* IP, ce modèle est tolérant aux pannes, redéployant automatiquement les protocoles suivant les routes empruntées par les flux associés. La disponibilité d'un cache de code est également l'un des intérêts de l'approche, lui conférant le statut d'approche mixte (paquet + nœud actif). Cette disponibilité évite en effet à chaque capsule de transporter son code. La disponibilité d'un cache de données permettant aux capsules de déposer des informations pour les suivantes est également utile pour de nombreux protocoles.

La principale limite de l'approche repose sur la non distinction entre usagers (anonymes dans ANTS) et sur l'absence de gestion de ressources dans les nœuds. Ceci est en grande partie dû aux fonctionnalités restreintes offertes pour cela dans la machine virtuelle Java (notamment 1.1 sur laquelle ANTS a été initialement développé). La seconde limite est certainement le manque de performances de son implantation et sa disponibilité unique en mode tunnel sur UDP. Une implantation réelle sur IP serait plus appropriée.

Des extensions sont à mener sur le déploiement d'extensions à distance ainsi que sur la fiabilisation du chargement de code de protocoles. Des améliorations peuvent également être apportées au support d'applications multiples sur un même nœud d'une part et sur le support de transparence aux applications (*i.e.* applications qui ne sont pas sur un nœud actif et qui pourraient profiter d'une passerelle vers le monde actif).

Une synthèse complète réalisée par les concepteurs d'ANTS est donnée dans [WET 99a]. Les auteurs insistent sur la viabilité de l'approche malgré les performances relativement faibles de l'implantation Java. Cependant, ils défendent l'idée que dans le futur relativement peu de services actifs seront déployés, mais que les réseaux actifs sont les seuls à permettre le test à grande échelle de nouveaux protocoles à des coûts faibles.

4.6 ASP EE

ASP EE(*Active Signaling Protocols Execution Environment*) [PHI 99a] est un environnement d'exécution développé dans le cadre du projet ARP (*Active Reservation Protocol*) à l'Université de South California [BRA 99a]. Cet environnement est le plus récent proposé dans la communauté.

4.6.1 Principes

ASP fournit un environnement d'exécution de nœud (EE) capable de supporter des Applications Actives[46] (AA). Dans ce projet, les applications actives visées sont des services qui implantent des protocoles de signalisation (ex. un AA pour RSVP). L'architecture ASP respecte la nomenclature définie dans l'architecture standard des nœuds actifs (voir section standardisation). Cet environnement supporte simultanément plusieurs AAs. Une application active supporte plusieurs activités en parallèle (par exemple plusieurs versions opérationnelles d'un protocole de signalisation)[47].

Du point de vue des interfaces de communication, ASP permet d'une part l'envoi et la réception de paquets actifs, mais également l'envoi et la réception de paquets standards. Pour cela, l'environnement offre aux AAs un mécanisme de filtre pour intercepter des paquets non actifs. Sur réception d'un paquet, l'EE détermine l'AA qui doit le traiter. Ce dernier détermine finalement l'activité cible. L'EE garantit l'isolation entre AAs mais permet à ces derniers de partager du code.

Le chargement dynamique du code des AA est assuré par un mécanisme de chargement à distance. ASP n'offre pas de support pour les paquets actifs ce qui est logique vu les types d'applications cibles : les protocoles de signalisation. En effet, pour ce type de protocoles, le code est en général assez volumineux pour ne pas pouvoir être transporté dans chaque paquet. Dans ASP, le chargement de code d'un AA se fait sur la réception dans le nœud d'un paquet actif qui comporte un champ spécifique définissant la spécification de l'AA cible. Si l'AA est présent, le paquet lui est envoyé, sinon, l'EE cherche à charger l'AA depuis des serveurs de codes distants. La spécification d'un AA est une chaîne de caratères comportant 3 éléments:

- un nom globalement unique pour l'AA visé ;
- une liste d'URL de serveurs sur lesquels les classes de code de l'AA peuvent être téléchargées. La liste de ces classes est également fournie dans la spécification ;

46. Le concept d'AA est proche de celui de protocole dans ANTS.

47. Ceci est une extension par rapport à l'architecture standard d'un nœud actif qui ne spécifie rien sur la manière dont un EE traite les AAs.

– le nom de la classe représentant l'AA. Cette classe est la classe instanciée par l'EE comme le point d'entrée de l'AA [48].

ASP définit en plus le concept d'UA (*User Application*). Celui-ci définit une application utilisateur qui a la responsabilité d'initier une AA sur le réseau [49].

Le code d'un AA est, de manière similaire à ANTS, gardé dans un cache de code de l'EE. Tout code non invoqué au bout d'une période d'expiration est supprimé du cache. Tout AA dispose d'un repository de données soft et/ou hard. Le modèle est proche de celui d'ANTS avec cependant la possibilité de sauvegarder tout type d'objet. Dans le repository soft, des mécanismes de rafraichissement et de *timeout* maintiennent l'état de la base de code.

Pour traiter des paquets non actifs, l'EE offre aux applications actives la possibilité de définir des canaux de communication filtrants. Ce concept est appelé *Network Channel*. Les canaux sont gérés par l'EE pour le compte des AAs. Ce mécanisme de canal permet d'une part d'offrir aux AAs deux types d'interfaces réseau : une interface VNET (*Virtual Network* : un réseau virtuel composé de liens point à point au dessus d'UDP) et IP (adresses V4 et V6 supportées). Un canal est un concept orthogonal aux véritables interfaces réseaux disponibles sur un nœud.

La principale innovation concerne le support et la gestion de version de code de protocoles (d'AA) dans la machine virtuelle via un mécanisme de mapping de noms étendu. En effet, l'EE ASP implante un support de mapping dynamique de noms de codes (classes) qui pour une classe donnée, transforme ce nom en celui de classe étendu avec un numéro de version majeure ainsi qu'un numéro de version mineure. Ce nom étendu est utilisé par le class loader pour charger le code. Par défaut, la version la plus récente est chargée. Une AA peut cependant demander une version précédente en fournissant cette information dans sa spécification. L'architecture globale d'un EE ASP est donnée dans la figure 22.

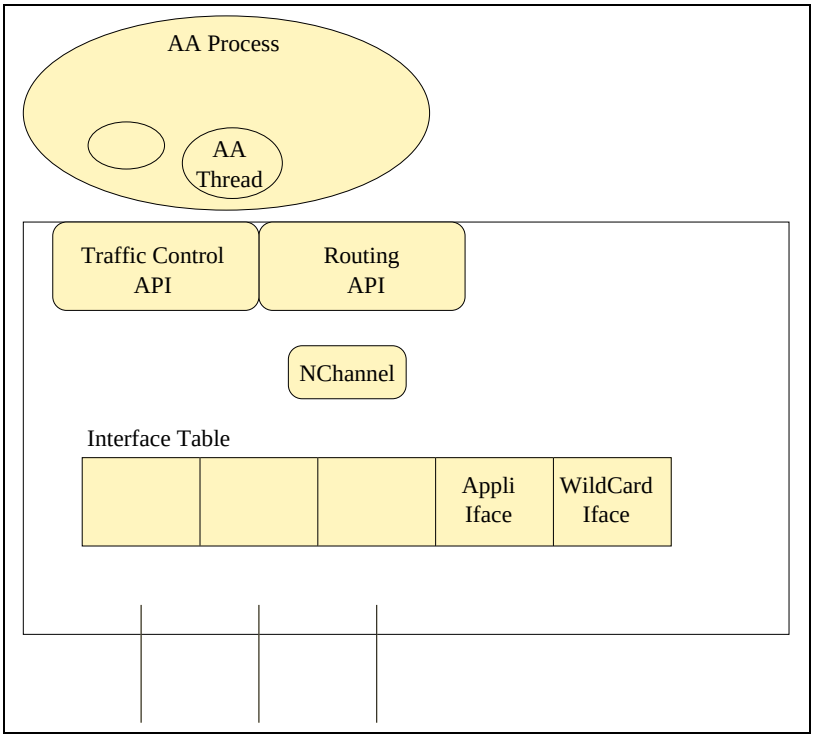

FIG. 22 – *Les principaux composants de l'EE ASP*

L'environnement d'exécution offre une interface spécifique pour les besoins de signalisation aux applications actives. Cette interface, appelée PPI (*Protocol Programming Interface*) [PHI 99b] fournit les services suivants :

– un modèle de processus d'application active (AA) ;
– un modèle de threads pour AA ;

48. Ceci est assez proche du paramètre de fonction à invoquer dans un chunk PLAN.
49. Ce concept est assez proche de la notion d'application dans ANTS.

- un *scheduler* propre (*round-robin*) entre les AA ;
- une API de gestion de store au sein d'un processus applicatif ;
- une API de gestion de container de softstate (*get, put, size, isempty*) et des fonctions de gestion du *refresh* et du *timeout*. Pour ces deux fonctions, l'AA fournit des *callbacks* à invoquer lorsque l'une des conditions est vérifiée ;
- les interfaces d'entrée/sortie (vers le réseau et/ou vers une application usager proposent une API d'envoi et de réception de messages ainsi qu'une liste de fonctions d'interrogation de ces interfaces (état, MTU, nom, adresse associée, préfixe CIDR, ...) ;
- une interface d'accès aux informations de routage. Cette interface offre principalement une fonction de demande de route (`RouteQuery`) qui renvoie la liste des interfaces utilisables pour renvoyer le paquet. Cette interface est extensible via la possibilité de définir sa propre fonction de recherche d'interfaces ;
- une interface de manipulation des fonctions de réservation de bande passante (RSVP). Cette interface comprend des fonctions d'initialisation de flux, d'ajout, modification, supression de spécification de flux, d'ajout et de modification de filtres et d'annonce tels que définis dans RSVP.

La nature de ces APIs est fortement orientée vers le contrôle du réseau et représente bien les AAs cibles d'ASP, à savoir les protocoles de signalisation et de routage sur IP et VNet.

4.6.2 Implantation

Le nœud actif ASP est implanté en Java. Un nœud opère sur une machine virtuelle standard. Le chargement dynamique de code est assuré par un *Class Loader* unique partagé entre les AAs. Les processus applicatifs (AA) au sein d'un EE sont implantés par des threads Java spécialisés. Tout AA doit être une sous-classe d'une classe appelée `AAContext` définie dans l'API PPI. Cette classe implante une fonction de réception de paquets ainsi que le support d'extraction de la specification de l'AA (`AASpec`) dans tout paquet qui le concerne. Les APIs d'entrée/sortie sont des interfaces (réseau ou interface de type *pipe*) vers les applications usagers dans certains nœuds d'extrémité. Les canaux de communication sont implantés par des threads (un ou plusieurs threads par canal).

4.6.3 Avantages et limites

L'approche ASP offre plusieurs mécanismes intéressants. Le premier est le support de traitement de paquets non actifs via le mécanisme de canaux réseaux. Ceci permet la conception de protocoles actifs directement sur IP servant de passerelle entre le monde *legacy* et le monde actif. Le second intérêt de l'approche réside dans le modèle d'applications actives. Celles-ci sont isolées entre elles et l'EE implante son propre mécanisme d'ordonnancement. ASP est assez proche des concepts développés dans ANTS mais en représente sans aucun doute une évolution.

La contre-partie de cette évolution est une programmation plus complexe des entrées-sorties et une architecture plus lourde pour le déploiement de code. Les aspects sécurité ne sont pas abordés dans les spécifications initiales (notamment les code non fiables) et il reste des précisions à apporter sur les accès direct à IP notamment. Les mécanismes qui implantent ce support ne sont en effet pas détaillés.

4.7 Protocol Boosters

4.7.1 Principes

Protocol Boosters [DAV 98, MAR 98] est une approche totalement orientée nœud actif. Un *Booster*, littéralement accélérateur, est un composant logiciel implantant une logique de protocole qui est capable d'améliorer la performance d'un réseau et le service du support sous-jacent. Un *Booster* s'installe et opère de manière transparente sur un protocole existant pour en améliorer le fonctionnement dans des conditions précises. Ce n'est pas, comme dans d'autres approches sur les réseaux actifs, un protocole complet de bout-en-bout qui assure toute la logique de service. Ceci est illustré dans la figure 23.

Les accélérateurs peuvent se composer entre eux et plusieurs peuvent opérer sur un même protocole. Concrètement, pour améliorer un protocole sur un réseau spécifique, il suffit de déployer à l'entrée et à la sortie de celui-ci ou sur un sous-ensemble du chemin, les accélérateurs qui optimisent le fonctionnement du protocole

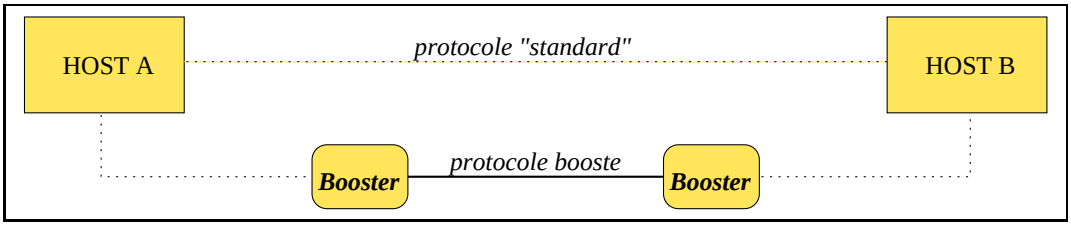

FIG. 23 – *Le concept de Booster*

à *booster*. Les *Boosters* modifient les paquets en entrée de la portion boostée et les restituent en sortie. Trois familles de *boosters* ont été développées dans l'approche :

- des *boosters* de supervision : composants permettant le traçage des traitements ainsi que de la journalisation ;
- des composants de débogage : ajout, suppression de paquets, insertion de latence, dupplication de paquets permettant de tester un protocole ;
- des composants de contrôle d'erreur (ARQ-Z,-A, FEC, FZC, PMOD, ACK-COMP, ...).

Du point de vue de l'architecture, l'approche profite des capacités de chargement dynamique de modules dans le noyau LINUX pour charger des extensions de protocoles et offre un mécanisme de séquencement de *Boosters* sur un même flux. Cette architecture est illustrée dans la figure 24.

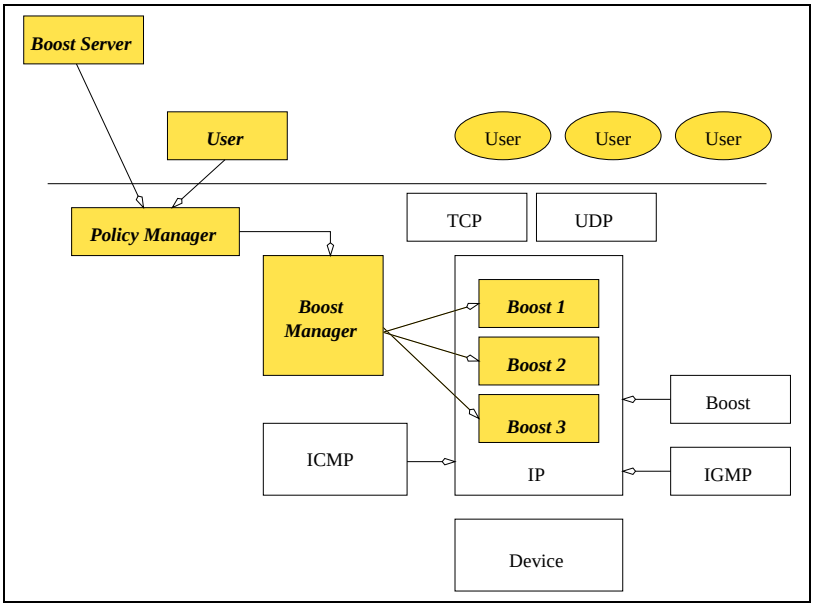

FIG. 24 – *L'architecture interne des nœuds de Boosters*

Chaque booster dispose de six interfaces : une interface de supervision, des interfaces d'entrée/sortie et de *forwarding* de paquets, des canaux de traitement *out-of-band* et une interface `ioctl`. Le gestionnaire de *boosters* est responsable du chargement/déchargement de boosters dans le noyau. Il assure également le monitoring et la comptabilité pour les différents *boosters* déployés. Le gestionnaire de politiques (*Policy Manager*) assure la redirection des flux vers les modules (*Boosters*) et gère la configuration du manager. Il lui offre également l'interface vers un opérateur sur une console et l'interface vers des serveurs de code distants. Le noyau offre un mécanisme performant de *pipelining* d'accélérateurs.

4.7.2 Avantages et limites

L'intérêt principal de cette approche est de fournir des motivations claires sur le besoin de telles architectures et de fournir une preuve de concepts au travers de l'implantation. Le second intérêt de cette approche réside

dans la nature discrète des *Boosters* qui sont totalement transparents à l'application certes, mais surtout au protocole *boosté*.

Le concept est intéressant et l'architecture bien maîtrisée. La seule limite actuelle est la restriction à des OS Linux. Une seconde extension non adressée aujourd'hui est le déploiement dynamique de boosters sur des portions de réseau sans intervention d'opérateur. Ce problème représente un sujet de recherche intéressant non abordé à ce jour.

4.8 Router Pluggins

4.8.1 Principes

Router Pluggins [DES 98a] est une proposition d'architecture logicielle de routeur modulaire extensible et performant. L'objectif est de concevoir une architecture de routeur capable de charger dynamiquement de nouveaux services et fonctions de traitement des paquets. Dans ce but, l'architecture proposée intègre la notion de flux de données et de composants de traitement. Cette architecture est illustrée dans la figure 25.

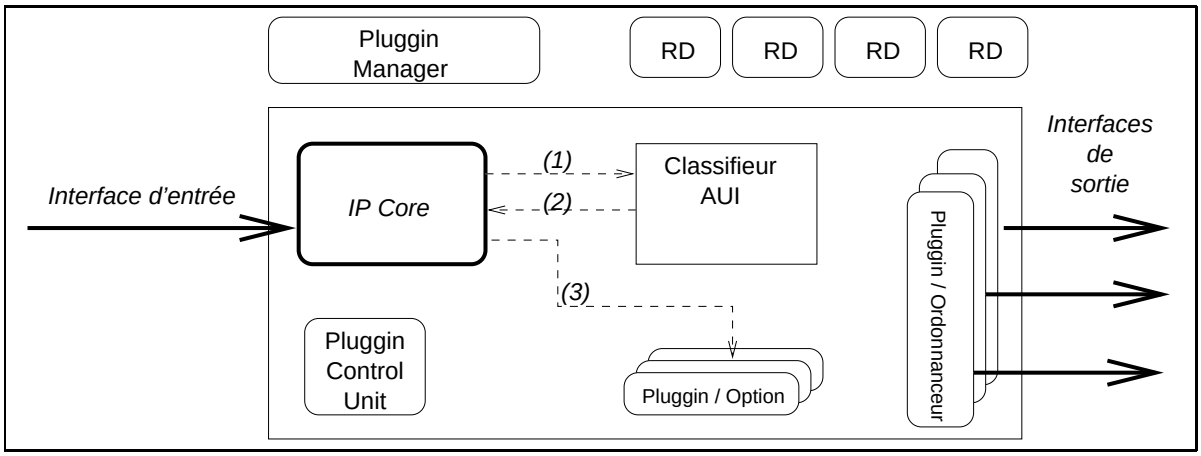

FIG. 25 – *L'architecture d'un routeur supportant les pluggins*

Un tel routeur est constitué d'un noyau IP dont le code est étendu avec des *slots* de traitement ouverts (appelés *gate* dans l'approche). Dès qu'un paquet est reçu et que dans le traitement le noyau rencontre une porte, il fait appel au classifieur AIU (*Association Indentification Unit*). Celui-ci identifie sur la base de filtres de paquets sous la forme `<@IP source, @IP dest, #port source, #port dest, protocole, plugginId>` [50] le *pluggin* à appliquer sur le paquet et renvoie un pointeur vers celui-ci au *core*. Une fois cette association établie, le traitement du *pluggin* est appliqué au paquet avant de passer au traitement suivant dans le *core* (éventuellement vers un autre pluggin d'ordonnancement en sortie par exemple. Le composant de contrôle des *pluggins* est appelé PCU (*Pluggin Control Unit*). Ce composant permet l'instanciation et la libération de modules dans le noyau ainsi que leur attachement (resp. détachement) à un filtre de paquets dans l'AIU.

L'architecture permet à plusieurs démons de routage de tourner sur le routeur. Un démon de supervision de modules est également proposé mais ses composants ne sont pas détaillés dans la proposition. Seule sa tâche de configuration à l'initialisation, *i.e.* lancement du routeur est proposée à ce jour.

Cette architecture, intitulée *Crossbow* a été implantée dans l'OS NetBSD [51] et offre d'excellentes performances. Elle a été reprise pour la construction d'un OS spécialisé pour les réseaux actifs appelé DAN et présentée dans la section dédiée aux systèmes d'exploitation pour les réseaux actifs.

4.8.2 Avantages limites

L'intérêt principal de cette approche est de fournir au niveau du noyau une architecture logicielle extensible et performante implantant de nombreuses optimisations. Une telle architecture est parfaitement adaptée pour accueillir des réalisations telles que les *Protocol Boosters*, présentés précédemment.

Du point de vue des réseaux actifs, cette architecture ne possède pas pour le moment d'environnement de création et de déploiement de *pluggins* à l'échelle du réseau. Un couplage avec des mécanismes de déploiement

50. Ce filtre représente dans cette approche l'identificateur de flux.
51. `http://www.netbsd.org`

issus des réseaux actifs serait intéressant. De même, une étude sur la gestion des *pluggins* serait à mener. Par exemple, l'agent de gestion attaché à un tel routeur pourrait lui-même, en plus de contenir une MIB générique de pluggins, être conçu sur un tel modèle et charger dynamiquement des MIB et l'instrumentation associée pour les *pluggins* déployés dans le nœud.

4.9 Netscript

NetScript [YEM 96], propose une architecture de réseau programmable (nœud actif pur). NetScript fournit une architecture pour programmer des réseaux, une architecture de nœud programmable dynamiquement dans le réseau, un langage appelé *NetScript* pour développer des logiciels de réseau sur cette architecture programmable ainsi qu'un ensemble de classes Java (*toolkit NetScript*). *NetScript* est développé pour supporter la notion de Réseaux Actifs Virtuels en tant qu'abstraction programmable. Les abstractions de Réseau Actif Virtuel peuvent être composées et gérées de manière systématique. De plus, *NetScript* automatise la gestion du nœud au travers d'extensions du langage qui génèrent des MIBs.

Les tâches principales d'un nœud *NetScript* sont le traitement des flux de paquets et l'allocation des ressources du nœud pour permettre ces traitements. La programmation des fonctions de traitement (services) est réalisée sous forme de programmes, appelés *boîtes*, interconnectés par des flux de paquets. Les boîtes forment un système réactif dans lequel les données sont transmises d'une boîte à l'autre [DAS 98a]. Une boîte peut être décrite en utilisant le langage *NetScript*] [DAS 98b], un langage fortement typé qui crée des abstractions permettant de programmer les fonctions des nœuds du réseau. Cette description est ensuite compilée afin de générer le code Java correspondant. Un protocole est donc décrit sous forme de composition de boîtes élémentaires.

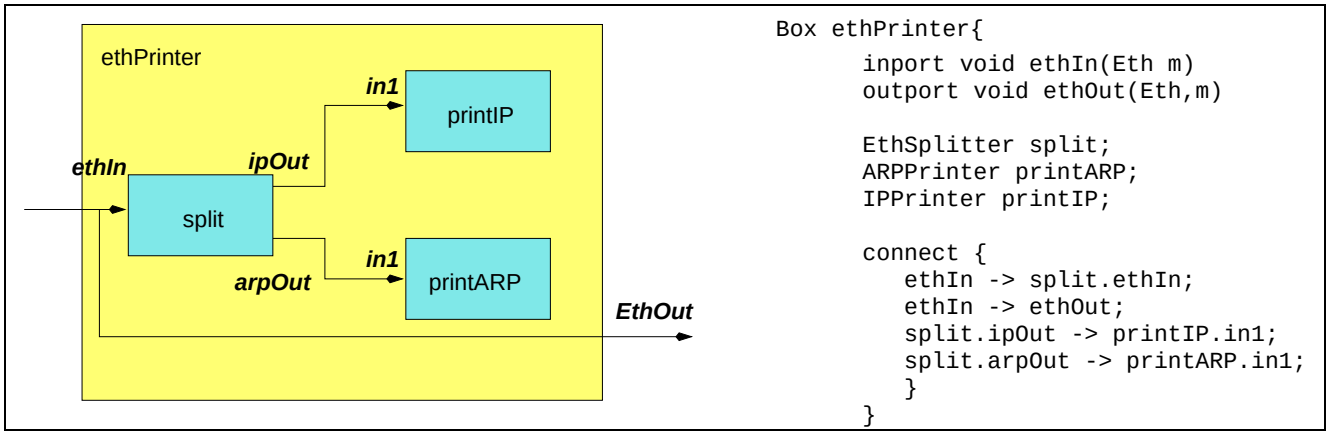

FIG. 26 – *Les blocs NetSript*

La composition de bout en bout permet la construction d'éléments actifs qui, une fois installés dans un réseau actif, réalisent des fonctions de bout-en-bout. Pour permettre la composition de bout-en-bout, NetScript utilise le concept de VAN (*Virtual Active Network*). Un VAN est constitué d'un ensemble de moteurs NetScript (NetEngine) dans un réseau virtuel, permettant le déploiement de code, la gestion de la configuration, la sécurité et la gestion des ressources de bout en bout. Un VAN est créé en reliant des moteurs NetScript au travers de liens virtuels. Un lien virtuel correspond à un ensemble de liens et de nœuds physiques et peut interconnecter n'importe quel nombre de moteurs NetScript pour gérer des liens de multi-diffusion. Le VAN est géré en tant qu'entité, ce qui ouvre des possibilités nouvelles pour la gestion de la sécurité à l'intérieur du réseau.

NetScript est une approche qui se classe à la fois dans la famille des réseaux programmables et dans celle des réseaux actifs (sous-classe nœud actif).

4.9.1 Avantages et limites

Le principal apport de *NetScript* est la programmabilité offerte à tous les niveaux du réseaux (Ethernet, IP, UDP, TCP). L'approche apporte une seconde innovation qui est son utilisation en supervision de réseau. En effet, ce type de nœud est parfaitement adapté pour créer des moniteurs spécifiques de type RMON mais bien plus puissants. Cependant, ce n'est pas dans ce but que *NetScript* a été initialement créé. Du point de vue des composants, la principale lacune est l'absence à ce jour du langage complet pour la spécification des composants

et, dans la version disponible sur Internet, la partie concernant la génération automatique de MIBs qui fut en partie à l'origine de l'approche.

4.10 ALAN

4.10.1 Principes

ALAN [52] (*Application Level Active Networking*) [FRY 99] est un projet en démarrage à l'université de Sydney. Dans ce projet, les chercheurs visent à démontrer l'intérêt d'une approche active pour des services applicatifs. Leur postulat de base est que les technologies actives sont utiles au réseau mais ne se retrouverons pas à des niveaux autres qu'applicatifs avant de nombreuses années.

Toute l'architecture vise donc à fournir la possibilité de développer et déployer dynamiquement des services actifs mais au niveau applicatif. Ces services sont transparents pour les applications finales (ex. serveur et client Web). Pour cela, une architecture à base de *proxy* extensible est proposée.

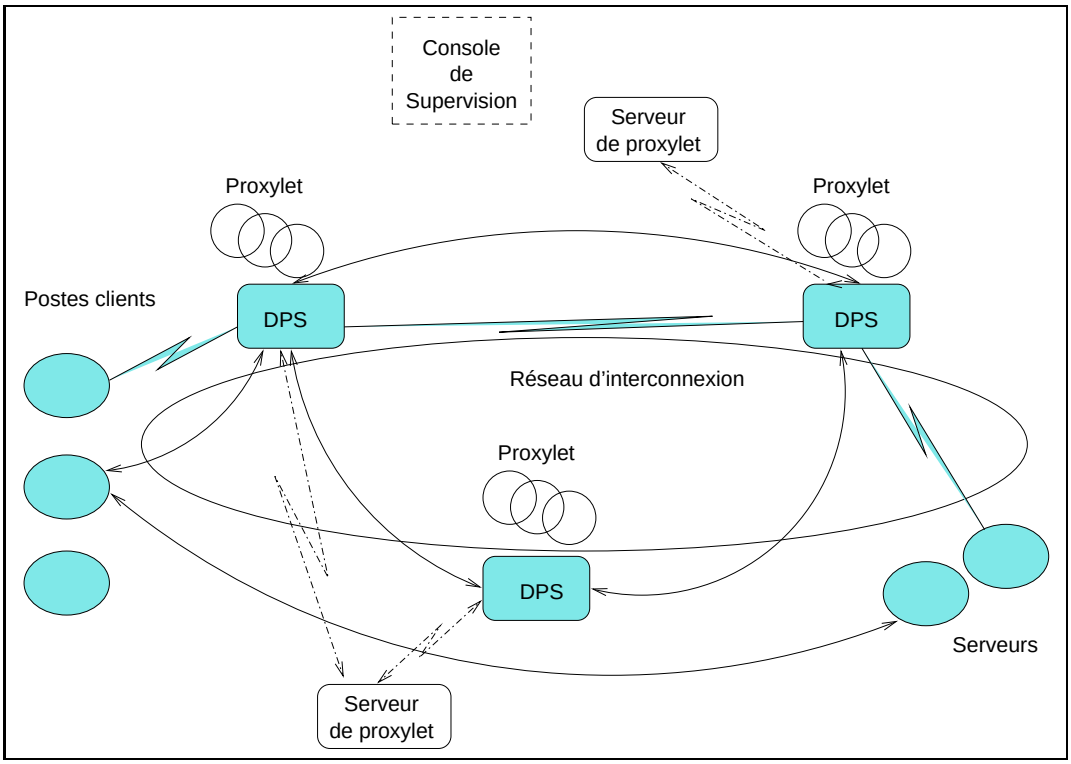

FIG. 27 – *Une architecture ALAN*

La figure 27 illustre les composants de l'architecture *ALAN*. Le composant de base est le proxy d'interposition entre applications appelé DPS (*Dynamic Proxy Server*). Les DPS sont placés par l'opérateur dans des endroits stratégiques du réseau.

Ce composant applicatif est un serveur invocable par une interface de contrôle permettant de charger des passerelles applicatives (*proxylet*, ensemble de classes Java) de les lancer, d'en modifier l'état courant et de les arrêter. Ces passerelles applicatives se trouvent elles même sur des serveurs de code gérés par l'opérateur. Chaque *proxylet* offre une interface de contrôle au DPS et aux usagers. Une console de supervision permet de suivre l'exécution et l'état des serveurs et des proxylets sur l'ensemble du réseau.

En résumé, cette architecture permet de configurer des proxys à l'aide de programmes Java pour des protocoles applicatifs. La réalisation de cette architecture a été faite en Java et utilisée principalement pour coder des transcodeurs de protocoles applicatifs (audio) et des compresseurs de pages Web.

52. `http://dmir.socs.uts.edu.au/projects/alan/`

4.10.2 Intérêts et limites

L'intérêt de cette approche est d'annoncer clairement son objectif de ne pas adresser des protocoles de niveau réseau et de fournir un environnement de proxys programmable. Cependant en limitant l'approche aux protocoles applicatifs, on laisse de côté de nombreux protocoles intéressants tels que les protocoles multicast qui trouvent naturellement leur place au niveau réseau.

Les limites principales de cette approche résident dans son implantation actuelle fortement restreinte. Une étude complémentaire est également à mener sur la spécification de la composition des proxylets distants pour un service donné. Ce point n'est pas abordé dans la proposition initiale, mais une contribution récente [MAR 99] porte sur l'utilisation de la technologie XML[53] pour véhiculer les informations de chargement de proxylet, les séquences d'enchaînement d'appels ainsi que les paramètres formels de ces derniers. Ce mécanisme est notamment utilisé pour fournir à un proxylet de routage des politiques spécifiques demandées par des usagers.

4.11 Services actifs

4.11.1 Principes

Avant *ALAN*, une architecture pour les services applicatifs actifs a été proposée dans [AMI 98]. Cette architecture, proposée comme une alternative aux réseaux actifs, prône le déploiement de code utilisateur dans le réseau mais de manière restreinte, en limitant ce déploiement à un niveau applicatif sans modifier le réseau sous-jacent (ici IP) et en restreignant les sites de déploiement à des endroits stratégiques dans le réseau (serveurs dédiés). Le principal argument en faveur de cette approche est qu'elle est nettement plus facile à déployer et maîtriser dans des réseaux actuels que des solutions opérant à des niveaux inférieurs.

Les composants de base de l'architecture (voir figure 28) sont les serveurs d'agents, les agents de service ainsi que les protocoles de contrôle et de sollicitation.

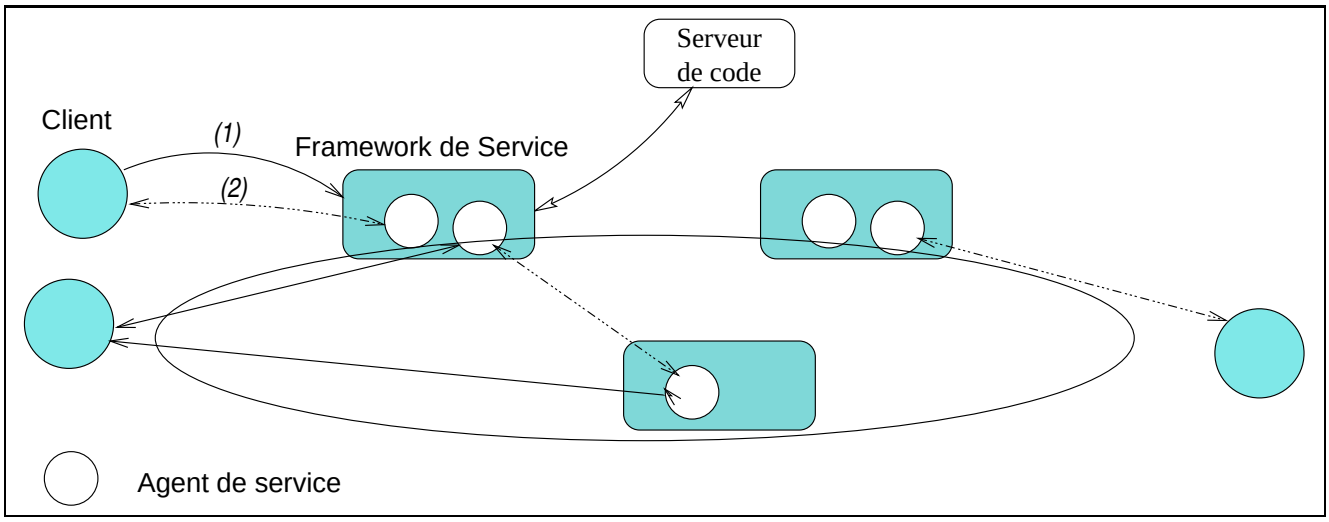

FIG. 28 – *L'architecture Active Services*

Les serveurs d'agents son appelés *Active Framework*. Ils offrent l'environnement d'exécution ainsi que l'API d'accès aux ressources pour les agents de services. Un agent de service implante la logique de service. Celle-ci peut être implantée par plusieurs agents distribués sur le réseau.

Un client demande, via un canal multicast et en utilisant un protocole spécifique appelé ASCP (*Active Service Control Protocol*), à l'architecture de service l'instanciation d'un agent de services (*1*). Cette architecture instancie un agent en fonction de ses disponibilités actuelles (partage de charge) et du type de service demandé (via un identificateur de code) sur l'un de ses serveurs. Une fois l'agent installé, il met en place le service (flux, connexion, traitement) et offre une interface de contrôle au client pour lequel il travaille (*2*). L'implantation de cette architecture a été réalisée en TCL.

53. `http://www.w3c.org`

4.11.2 Intérêts et limites

Le principal intérêt de cette approche est bien sûr de fournir une architecture propre pour le déploiement de services applicatifs. L'un des problèmes non traités ici reste celui de la sécurité qui, d'après les auteurs, est en cours d'étude.

La principale limite d'une telle approche est la restriction au niveau applicatif. Celle-ci implique que de nombreux services ne sont pas traitables (multicast, contrôle de congestion, ...) en raison de leur localisation nécessaire dans les couches inférieures (notamment 3). Cette restriction est identique dans l'architecture ALAN.

4.12 Agents mobiles

Dans le domaine des agents mobiles, plusieurs projets examinent également l'utilisation de ce paradigme pour le déploiement de code actif dans le réseau. Cette approche est assez naturelle car un réseau actif peut être vu comme une plate-forme à agents mobiles spécialisée dans les couches basses du réseau. Les paquets actifs eux, peuvent alors être définis comme des agents mobiles très compacts et opérants sur les nœuds du réseau. Dans le contexte de réseau programmables, les agents mobiles sont souvent proposés comme infrastructure de déploiement des services. Ceci est notamment le cas dans le cadre du projet européen MARINE[54] où de tels agents sont utilisés pour le contrôle de réseaux intelligents.

4.13 Autres approches

Il existe de nombreuses autres approches proposées pour les réseaux actifs mais dont les travaux sont extrêmement récents et qui ne sont pas aujourd'hui aussi aboutis que les propositions présentées précédemment. Cette section a pour objectif de les présenter brièvement. Celles dont les évolutions aboutiront à des architectures et réalisations complètes, trouveront dans les versions ultérieures de ce rapport, une section qui leur sera dédiée.

4.13.1 Une approche langage

La première approche présentée ici est celle développée à l'université de Sussex. Dans un premier temp, cette approche s'est focalisée sur la définition d'un langage de programmation de programmes actifs intégrés dans des capsules suivant un certain nombre de contraintes identifiées et disposant d'une sémantique formelle [JEF 97, WAK 98]. La sémantique du langage est basée sur CCS et sur la théorie des types pour exprimer des propriétés de *safety*. Le modèle de communication choisi est celui de l'Internet (routage dynamique et asymétrique, paquets actifs transparents aux routeurs non actifs, réseau ne garantissant pas la délivrance de paquets ni leur arrivée dans l'ordre). De nombreuses contraintes ont été identifiées sur l'environnement d'exécution (limitation de l'utilisation des ressources, limite de threads, de mémoire et de bande passante en sortie par programme actif, ...), et le langage de programmation (pas de boucle infinie, accès uniquement au nœud suivant pour une destination donnée, tracabilité et certification de l'origine d'un paquet actif, ...). Le langage défini est un langage objet proche de Java et intégrant les primitives de communication de paquets actifs. À ce jour, un compilateur Java a été défini pour ce langage et un environnement de simulation est en cours de réalisation.

4.13.2 NFL

La seconde approché résumée dans cette section est une approche basée sur un concept de nœud actif. Elle se situe dans le prolongement des travaux sur *NetScript* et propose un langage de description de paquets basé sur XML. Cette approche appelée NFL[55] (*Network Flow Language*) [YEM 99] propose un langage de description de paquets intitulé FTD (*Flow Tag Definition*) proche du concept de DTD (*Document Type Descrition*) dans XML. Ce langage basé sur XML permet de définir la structure d'un paquet actif ainsi que les balises propres au protocole défini. FTD inclut également le support naturel d'un ensemble de types de base (entiers, booléens, caractères, ...). La sémantique des paquets est définie à l'aide d'une notation appelée XFL (*eXtensible Flow Language*) qui est un langage de règles inspiré d'XSL (*eXtensible Stylesheet Language*). Dans ce langage, le concepteur d'un protocole peut définir l'association entre un ensemble de tags et un code capable de traiter leur contenu dans les différents nœuds traversés par le paquet. Ce code est appelé *Flow Processor* et peut être implanté à l'aide de n'importe quel langage de programmation suivant une API définie dans l'approche (en fait

54. `http://www.italtel.it/drsc/marine/marine.htm`
55. `http://www.cs.columbia.edu/dcc`

une API proche de *NetScript*). À ce langage, est associé un environnement d'exécution dont l'architecture est illustrée dans la figure 29.

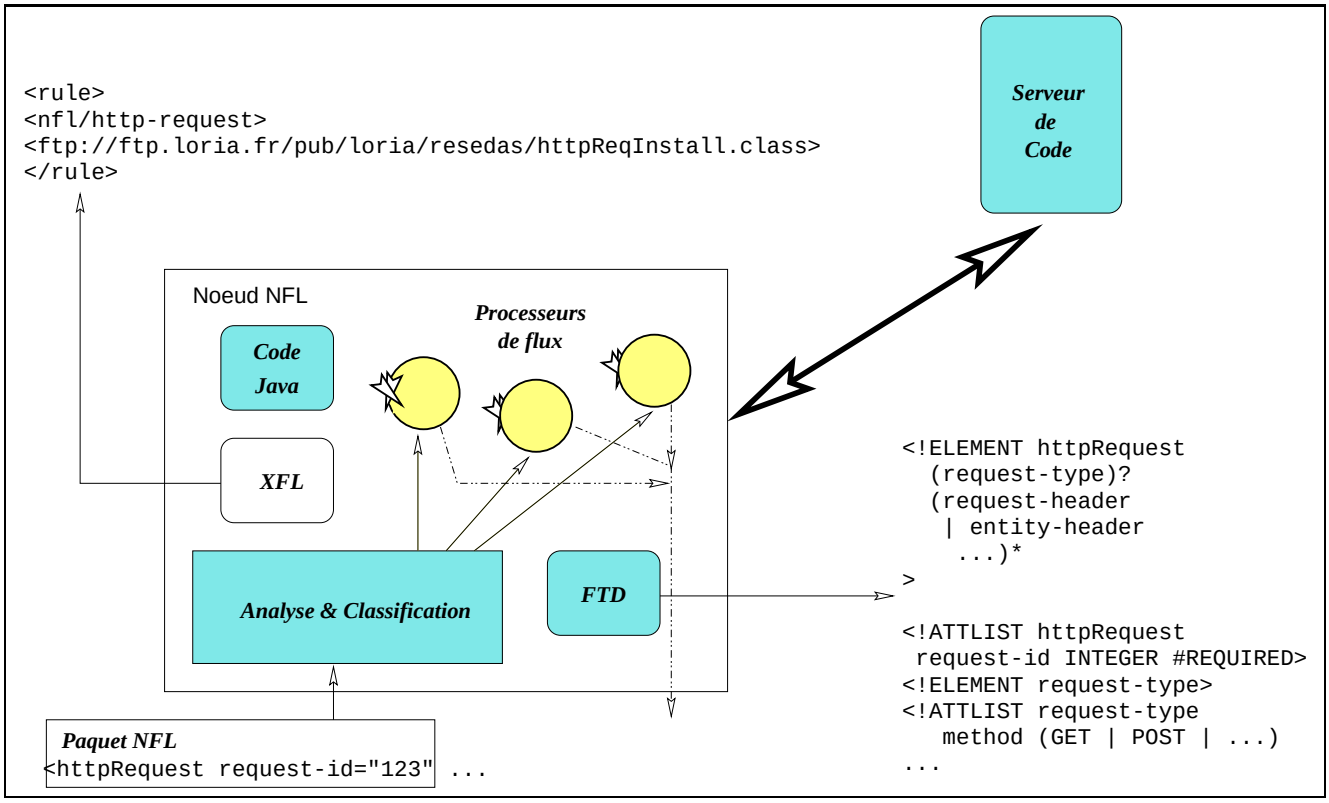

FIG. 29 – *L'architecture de nœud dans NFL*

Le mode de programmation des traitements s'inspire du modèle de flux de *NetScript* (boîte, flux, canal, composition). Sur arrivée d'un paquet, le nœud NFL analyse le contenu (analyse syntaxique suivant la spécification FTD) et construit l'arbre abstrait associé. En fonction des balises, le nœud invoque via une fonction de classification le ou les traitements associés définis dans la spécification XFL associée au protocole. Si, le nœud ne dispose pas de la spécification FTD, ni du code associé, il peut récupérer ces derniers sur un serveur auquel il accorde une totale confiance. Le code ainsi que les feuilles de style et les spécifications de protocole sont téléchargés à la demande des nœuds sur arrivée de paquet.

Pour un paquet donné, le réseau est vu comme une suite de traitement qui sont appliqués sur le paquet suivant les balises qu'il porte. Cette séquence de traitement est appelée «canal» dans l'approche.

Cet environnement d'exécution a été appliqué à la spécification d'un protocole de transport pour informations du Web proposé en replacement du couple HTTP/TCP et l'environnement d'exécution à été réalisé sur la base de l'approche *NetScript* en Java. En effet le codage de fonctions de traitement se fait en Java suivant le même modèle que *NetScript* (composition de boîtes via des connecteurs définissant des flux). L'extension de *NetScript* a cependant été nécessaire par l'introduction de boîtes spéciales appelées classifieurs dont l'objectif est de router les paquets au sein d'un nœud vers le bon code en fonction des balises trouvées dans le code, fournissant ainsi un graphe de traitement adapté. Cette approche est toute récente et très intéressante. Il manque cependant encore un certain recul concernant son utilisation pour des protocoles plus complexes que l'exemple cité et la disponibilité d'un environnement logiciel stable serait un plus. Cela arrivera probablement bientôt.

4.13.3 AMNet

AMNet[56] (*Active Multicasting Network*) [WIT 98c] fournit une architecture pour le support de multicast dans un contexte de récepteurs hétérogènes (principalement une hétérogénéité de QoS). L'objectif de cette architecture est de mettre en place au travers de l'actif dans les nœuds internes du réseau des filtres de QoS, une signalisation étendue pour la gestion de ces filtres ainsi qu'un mécanisme de contrôle d'erreur. L'architecture

56. `http://www.ibr.cs.tu-bs.de/~wittmann/AMnet.html`

repose sur des démons de gestion des filtres dans tous les nœuds internes. L'originalité de l'approche réside dans l'utilisation de RSVP et de ses extensions pour mettre en place la signalisation de configuration entre les nœuds. Ce qui est aujourd'hui discutable dans cette approche, par rapport aux réseaux actifs, est que la proposition ne fait pas apparaître l'intérêt de l'actif dans l'architecture visée. En effet, tous les filtres sont codés en dur dans les démons et la signalisation est figée. Le chargement dynamique de filtres dans les nœuds n'est pas du tout abordé. Celui-ci fournirait certainement une motivation suffisante pour l'utilisation des capacités offertes par les réseaux actifs.

4.13.4 L'approche de Georgia Tech

L'architecture de nœud actif définie à Georgia Tech [BHA 97b, LEG 98b, BHA 96] est une approche typiquement nœud actif qui nécessite l'implantation de services actifs dans le noyau d'un système d'exploitation, ici Solaris. Elle permet à des utilisateurs, à l'aide d'options IP spécifiques à leur approche, de choisir parmi un ensemble de services pour les traitements sur ces derniers dans chaque nœud traversé. La fonction à appliquer sur un paquet est définie dans l'option par un identificateur unique. Cette fonction est paramétrable sur le nœud et les paramètres effectifs sont fournis dans l'entête de chaque paquet.

Le nœud est caratérisé par une table de fonctions avec pointeurs sur le code, permettant de dispatcher tout paquet entrant sur la bonne fonction de traitement. Le nœud permet également l'accès à certains états du nœud.

Cette architecture n'est pas facilement extensible. Cela est dû à l'objectif des recherches qui ont été menées sur ce réseau, visant plus les aspects performances de services que des aspects liés à leur déploiement dynamique. Plus récemment, les chercheurs de Georgia Tech ont défini un simulateur de réseau actif basé sur *ns*. Cet environnement semble très intéressant mais n'est pas distribué à ce jour.

4.13.5 RCANE

RCANE [MEN 99] est une proposition d'extension de l'architecture *Switchware* incluant un support de contrôle strict des ressources au sein d'un nœud actif. Afin de fournir cette fonctionnalité, RCANE introduit le concept de **session**. Celle-ci représente une entité pour laquelle des ressources sont réservées sur un nœud. Le nœud fournit également un mécanisme de comptabilité qui gére la répartition des ressources entre les sessions.

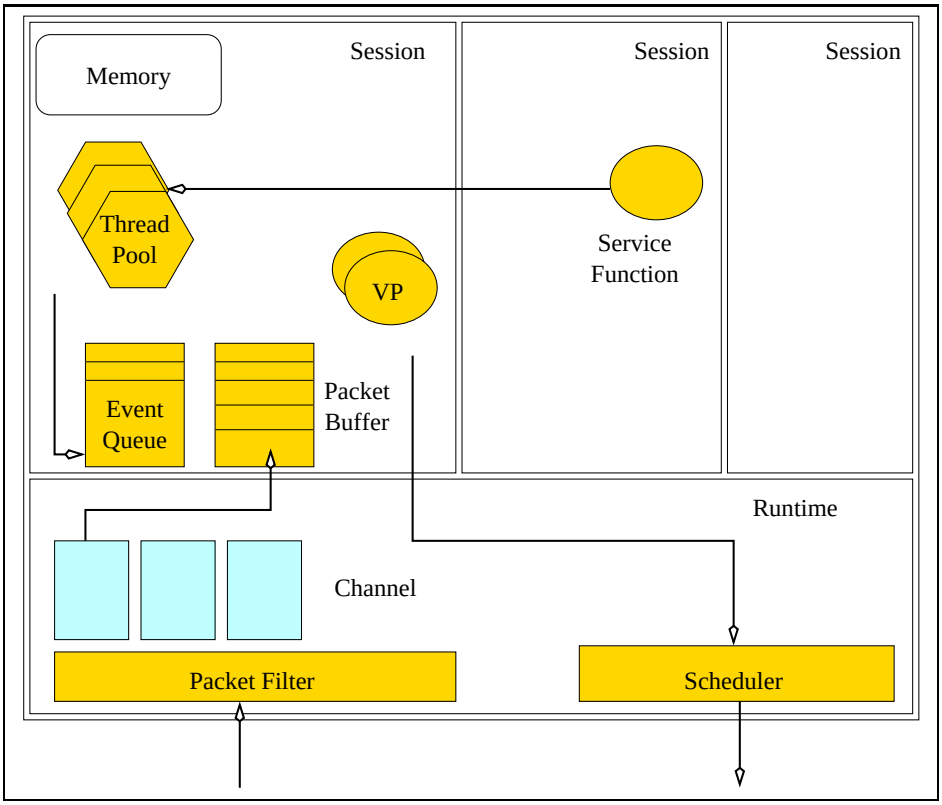

FIG. 30 – *L'architecture de nœud dans RCANE*

La figure 30 illustre les composants d'une session ainsi que l'architecture d'un nœud. L'architecture d'un nœud comprend en plus des composants de *Switchware* :

- un *runtime* qui fournit l'accès et l'ordonnancement des ressources du nœud
- un *core* étendu qui gère les droits d'allocation de ressources
- un mécanisme de chargement de code à distance proche de celui d'ANTS.

À chaque session sont alloués des processeurs virtuels dont les politiques d'allocation sont gérées par la session elle même. Les unités de traitement sont allouées à ces processeurs par le *runtime*. Une session demande l'allocation de CPU en définissant des fenêtres d'exécution sur des périodes de temps (ex. 40 ms toutes les 5 secondes). Lors de la création de session, les paramètres de mémoire et de CPU sont positionnés. Une session particulière appelée session système gère les autorisations des demandes et vérifie la compatibilité des demandes avec la charge actuelle du nœud.

Il existe un mécanisme supplémentaire permettant à une session d'exporter des services vers une autre session. La session cliente est facturée (ressources utilisées) pour l'utilisation de tels services. Une API de création, destruction de session est fournie aux applications. L'ensemble de ces composants a été réalisé en OCAML et implanté au dessus du système d'exploitation Nemesys au sein duquel les canaux de communication sont directement mappés sur les canaux RCANE. La classification des paquets se fait sur des filtres définis dans *Nemesys*.

4.13.6 MAGICIAN

MAGICIAN est un prototype de réseau actif développé à l'université du Kansas. L'approche est une approche paquets actifs (les paquets sont intitulés *SmartPackets*) reprenant la plupart des concepts de l'approche ANTS. Les principales différences résident dans le support de la sérialisation Java 1.1, dans l'encodage des paquets actifs, le support d'un démultiplexage ANEP pour de multiples environnements d'exécutions au sein d'un nœud, une gestion plus fine des ressources dans un nœud (limitation de CPU et mémoire / thread / capsule). Ce dernier point, gestion et allocation des ressources est certainement la contribution la plus intéressante de cette approche [WIJ 97]. En effet, les auteurs ont mis en place un mécanisme de scheduling des threads usagers basé sur un mécanisme de priorité des files d'attente (forte, standard, faible) ainsi que sur une classification des threads en fonction de leur activité sur un nœud (supervision: prioritaire mais peu de CPU, services déployés sur nœud : durée de vie longue mais activité sur des périodes courtes, paquets actifs : durée de vie courte et activité intense en CPU sur la durée de vie). L'allocation de mémoire est contrôlée par le nombre d'appels au constructeur `new` de Java[57].

Finalement, le mécanisme de déploiement ne semble pas être au niveau de celui d'ANTS (il n'est pas clairement défini comment les codes sont transmis aux nœuds). En tout cas cela ne semble pas être via des *SmartPackets*). À noter cependant un mécanisme d'optimisation réalisé sur sérialisation qui peut être partielle, de paquets actifs.

4.14 Les projets périphériques

Il existe aujourd'hui de nombreux projets liés aux réseaux et services dont l'objectif principal ne constitue pas la réalisation d'un environnement d'exécution actif, mais qui s'appuient ou peuvent potentiellement s'appuyer sur de la technologie active pour atteindre leurs objectifs. Ces projets sont présentés dans cette section.

DETOUR[58] [SAV 99] est un projet mené à l'université de Washington. Il vise à fournir des solutions à l'inefficacité des mécanismes de routage et de transport dans l'Internet actuel. Une première phase de ce projet a permis de dégager différentes raisons d'inefficacité de ces mécanisme dans le réseau actuel. L'inefficacité du routage est définie comme l'envoi d'un paquet sur une route donnée alors qu'il existe une alternative plus performante. Les raisons d'inefficacité relevées par les auteurs pour le routage sont : l'extrême pauvreté des métriques de routage échangées entre routeurs (principalement limités à une information de connectivité ou de nombre de systèmes autonomes traversés, rien sur les performances) ; des politiques de routage trop restrictives, e.g. routage vers le réseau externe le plus proche ; sélection d'un chemin unique pour le routage. Pour le transport (TCP), les auteurs montrent comment les mécanismes de TCP peuvent réduire de manière dramatique les performances de l'application en fonction de la latence et de la perte de paquets.

57. *NDLR: À notre avis, cela peut être bypassé par un paquet malveillant, mais bon... à confirmer*
58. `http://www.cs.washington.edu/research/networking/detour/`

Concrètement, l'architecture *DETOUR* se compose actuellement d'un ensemble de nœuds routeurs interconnectés entre eux sur l'internet par des tunnels IP[59]. Ces routeurs *DETOUR* sont placés stratégiquement aux frontières du réseau et effectuent de l'aggrégation de trafic usager. Ces routeurs s'échangent des informations de QoS sur leurs liens permettant une détermination de routage plus optimale. De plus l'étude de routage multi-chemins est également en cours de réalisation dans ces routeurs ainsi que du routage par classes de trafic (ex. TCP par d'autres chemins que UDP en fonctions de certains paramètres de QoS).

Le côté actif de l'approche *DETOUR* réside dans la possibilité d'informer les protocoles de transport sur les véritables capacités du réseau en termes d'informations détaillées sur la disponibilité de ressources (ex. RTT disponible dès l'ouverture d'une connexion TCP).

4.15 Autres projets

Il existe de nombreux autres projets d'architectures autour des réseuax actifs. Ceux-ci ne sont pas détaillés dans cette version initiale du rapport. Ces projets sont :

- NINJA (U. Berkeley)
- Messenger (U. Zurich)
- ANAIS (LORIA- INRIA Lorraine)
- LIANE (Integration de services WG)
- PANDA (U. Berkeley)

4.16 Réseaux d'expérimentation

Il existe plusieurs projets autour du déploiement à grande échelle de réseaux actifs.

L'ABone[60] [BRA 99b], un réseau mondial d'interconnexion de nœuds actifs en est un exemple. Ce réseau, soutenu par de nombreux projets DARPA, a pour objectif de fournir une infrastructure de grande échelle, *i.e.* atteindre rapidemment plus de 1000 nœuds pour de nombreux systèmes d'exploitation de nœuds, environnements d'exécution et applications actives, tout ceci de manière concurrente et sûre. L'opération de ce réseau se fait via une administration distribuée dans les sites participants. La coordination est assurée par une entité appelée ABOCC[61] actuellement assurée par des chercheurs de l'ISI et du SRI. Celui-ci est présenté dans la section dédiée à la standardisation.

Il existe aujourd'hui une alternative au *ABone*. Celle-ci repose sur l' architecture ANON[62] (*Active Network Overlay Network*). Son déploiement expérimental entre plusieurs sites à travers le monde utilise un format de paquets propre, fournit une autre plate-forme internationale d'expérimentation pour les réseaux actifs et les agents mobiles.

Tous ces réseaux sont aujourd'hui des réseaux d'*overlay* et interconnectent l'ensemble de leurs nœuds au travers de tunnels UDP ou TCP. Cette situation est en train d'évoluer, notamment dans le ABone ou des implantations commencent à supporter du multiplexage/démultiplexage de paquets actifs au niveau réseau et même liaison de données pour certaines implantations.

5 Standardisation

L'architecture d'un réseau comporte la définition des couches de communication, des protocoles utilisés, des schémas d'adressage et autres conventions nécessaires à sa mise en oeuvre. Dans le domaine des réseaux actifs, la standardisation de l'architecture n'en est qu'à ses débuts. Celle-ci s'est à ce jour focalisée sur la définition d'un protocole d'encapsulation des données actives et sur l'architecture interne d'un noeud. Plus récemment des travaux se sont développés autour de la composition de services dans les réseaux actifs. Ces trois points ainsi que l'architecture de sécurité sont détaillés ci-après. Une vision d'ensemble de la standardisation dans les architectures de réseaux actifs peut également être consultée dans [SMI 99].

59. Paquets IP inter-routeurs encapsulés dans des paquets IP qui transitent sur le réseau standard.
60. `http://www.csl.sri.com/ancors/abone/`
61. ABone Coordination Center
62. `http://ryyppy.ifi.unizh.ch/~anon/`

5.1 Les protocoles d'encapsulation

5.1.1 ANEP

Premier composant d'un réseau actif en cours de standardisation et aujourd'hui largement reconnu et utilisé par la communauté, ANEP (*Active Network Encapsulation Protocol*) [ALE 97a] définit le format d'un paquet actif. Ce format est indépendant du protocole dans lequel ce paquet actif est véhiculé (IP, TCP, UDP, trames Ethernet, ...) et s'adapte à tous les types de paquets actifs tels qu'ils sont définis dans les différents projets (ANTS, PLAN, SmartPackets, ...). Le format des paquets est donné dans la figure 31.

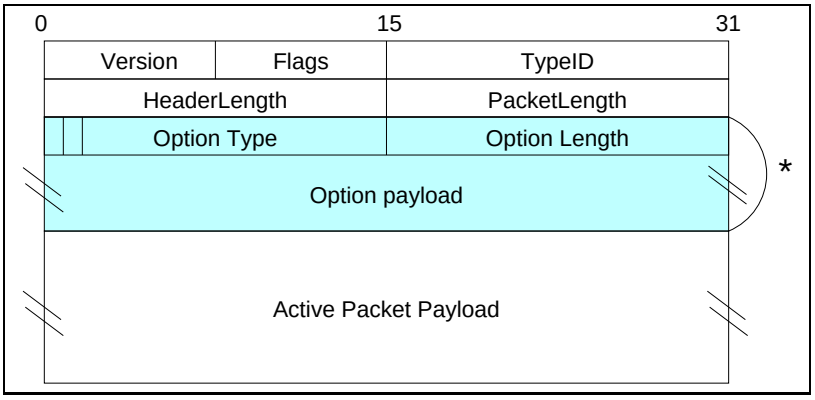

FIG. 31 – *Le format des paquets ANEP*

Il est composé d'un entête et d'un *payload*. Les composants de l'entête sont :

- un identificateur de version ;
- des drapeaux indiquant au routeur actif le comportement à adopter en cas de non connaissance du type (1 : jeter le paquet ; 0 : tenter de le transmettre suivant un routage par défaut). Seul 1 bit est actuellement utilisé ;
- un identificateur de type permettant d'indiquer l'environnement d'exécution associé au paquet. Les valeurs de types sont gérées par l'ANANA (*Active Network Assigned Number Authorithy*) et affectées à des environnements d'exécution. Par exemple, le projet ANAIS du LORIA dispose des identificateurs 71 à 75 ;
- un certain nombre d'options.

Toute option est une représentation Type/Longueur/Valeur. Actuellement quatre options sont définies :

- une option d'adressage de la source (schéma d'adressage, e.g. IPv6 + adresse) ;
- une option d'adressage de la cible (schéma d'adressage, e.g. MAC + adresse) ;
- une option de checksum sur un paquet ANEP complet ;
- une option de certification non négociée permettant à un émetteur de s'authentifier auprès des nœuds du réseau.

Plusieurs schémas d'adressage sont possibles. La taille maximale d'un paquet ANEP est fixée par celle des données du protocole utilisé pour véhiculer les paquets ANEP.

La nature générique de ce protocole dénote l'incertitude actuelle sur la localisation exacte d'un environnement d'exécution dans une pile de protocoles (niveau 2,3,4 applicatif ....). Ceci permet notamment à l'environnement d'exécution qui s'appuie sur le format ANEP de s'abstraire partiellement des détails d'adressage du réseau sous-jacent et en cas de modification de celui-ci d'être rapidement adapté. Les schémas d'adressage supportés sont IPv4, IPv6 et MAC. Les adresses source et destination d'un paquet actif sont définies dans une option ANEP qui comporte l'identification du schéma et l'adresse correspondante.

De nombreuses implémentations de codec ANEP dans différents langages de programmation sont aujourd'hui disponibles sur Internet et de nombreux projets ont retenu ce format de paquet pour l'échange de données actives entre les nœuds du réseau.

5.1.2 SAPF

SAPF[63] (*Simple Active Packet Format*) définit un format alternatif à ANEP pour l'encapsulation de paquets actifs. Ce format est utilisé dans le réseau d'expérimentation ANON

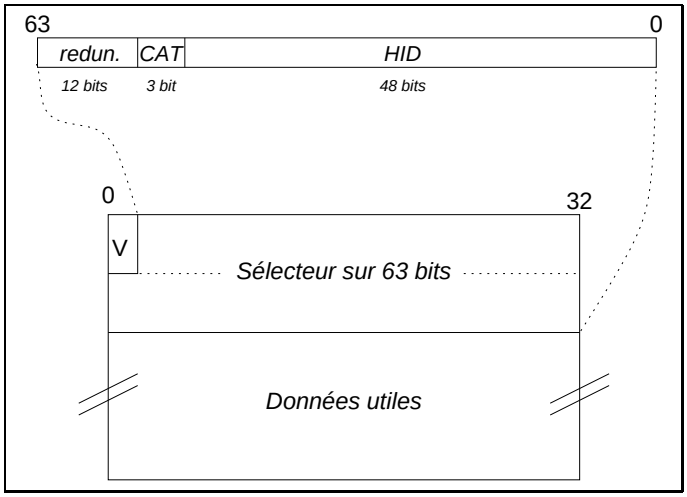

FIG. 32 – *La trame SAPF*

Le format de tels paquets est donné dans la figure 32. Son contenu est relativement simple : un champ de version (actuellement 0), un entête de 63 bits représentant un label d'identification (sélecteur) et un champ de données comportant l'information active utile. Le sélecteur est utilisé pour faire du tag switching dans les routeurs. Trois mécanismes d'allocation de sélecteurs sont permis :

- allocation statique : des valeurs de sélecteurs sont associées à chaque type d'environnement d'exécution[64] (*e.g.* ANTS, Switchware, Messenger). Tous les nœuds actifs ont une connaissance statique de ces sélecteurs ;
- allocation dynamique par le récepteur : similaire à l'approche d'*IP switching*, l'allocation d'un sélecteur se fait par le destinataire sur demande de l'émetteur ;
- allocation dynamique par l'émetteur : l'émetteur choisit un sélecteur pour son flux actif sans coordination avec le récepteur.

Le sélecteur est composé d'une valeur sur 48 bits qui définit le sélecteur lui-même (HID : *Handler Identifier*), 3 bits qui en définissent la classe (0 : statique, 1 : récepteur, 2 : émetteur) et un code correcteur d'erreurs sur 12 bits permettant de corriger jusqu'à 2 bits d'erreurs dans le sélecteur. L'adressage des machines sources et destination est réalisé dans les données utiles. La taille maximale d'un paquet SAPF est déterminée par la taille maximale des données utiles du protocole sous-jacent. Ce réseau est aujourd'hui extrêmement peu utilisé.

5.1.3 Une représentation ASN.1

5.2 ASN.1

ANEP et SAPF ne définissent que les entêtes des paquets actifs. Le codage et la représentation du corps des paquets actifs est laissée à la discrétion de chaque approche. Ce constat a conduit les auteurs de [WIL 99] à proposer l'utilisation d'ASN.1 pour représenter des programmes actifs. Cette proposition se base sur la notation de valeurs ASN.1 et propose que tout environnement actif fournisse un traducteur de ses programmes actifs vers une notation de valeur ASN.1, permettant ainsi au noeud actif d'assurer le codage/décodage du corps des paquets actifs (voir figure 33 ci joint).

L'intérêt principal d'une telle configuration est la possibilité de séparer l'encodage/décodage des environnement d'exécution et de réaliser cette fonction directement dans le noeud actif supportant de multiples environnements, ceci dans un but d'efficacité et d'interopérabilité entre environnements d'exécution. Cet argument est parfaitement discutable. En effet, d'une part les indications d'efficacité ne sont appuyées d'aucune mesure et, si il existe aujourd'hui des codec ASN.1 effectivement efficaces, le processus de mapping de tout programme actif

63. `http://ryyppy.ifi.unizh.ch/~sapf/sapf-0.7.html`
64. Le type ID assigné par l'ANANA pourrait être utilisé dans ce but.

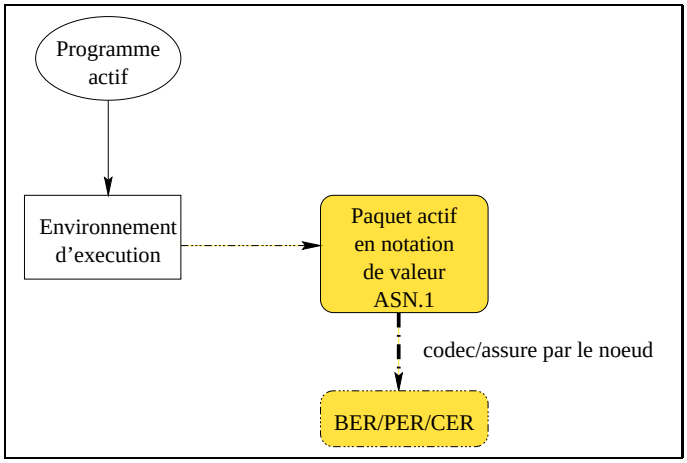

FIG. 33 – *ASN.1 et les réseaux actifs*

vers une notation de valeur ASN.1 prend certainement autant de temps que le mapping vers une représentation interne propre à l'approche, d'où une perte de temps probable. Le second point est encore plus hasardeux car fournir un format de représentation commun entre environnements d'exécution est certes un pas vers l'interopérabilité, mais les APIs offertes pas différentes EE aux programmes actifs n'étant pas les mêmes, cette interopérabilité visée ne peut être réalisée.

Cependant, cette proposition a, selon nous un intérêt majeur différent [65] qui est celui de pouvoir, sans changer les traducteurs des environnements d'exécution, choisir entre plusieurs règles d'encodage (BER [66],PER [67],CER [68]) en fonction du contexte et des besoins.

5.2.1 Autres représentations

Les trois représentations présentées ci-dessus se veulent génériques et indépendantes de toute approche de réseau actif. Actuellement, chaque approche de réseau actif dispose de sa propre représentation des paquets. Chaque représentation est détaillée dans les présentations des approches. Cependant certains projets ont également adoptés, ou des adaptations ont été faites pour supporter l'un des formats génériques présentés précédemment, notamment ANEP.

5.3 Architecture de noeud

Dans la continuité de la définition d'un protocole d'encapsulation, les travaux de standardisation ont porté sur l'architecture d'un nœud actif. [WG. 99] définit une architecture standard sur laquelle se basent plusieurs réseaux d'expérimentation tels que le ABone.

Les principaux composants d'un nœud actif sont illustrés dans la figure 34. Le composant de base est l'environnement d'exécution (EE). Un environnement d'exécution est une machine virtuelle capable d'interpréter un type de paquets actifs, ex. ANTS, *Switchware*, *Messenger* mais aussi IPv6 ou IPv4 qui sont vus comme des EEs dans cette architecture. Cette machine offre des APIs spécifiques aux codes entrants via des paquets actifs ainsi qu'un moteur d'exécution pour ces codes. Un EE peut être considéré comme un ensemble de librairies et de services qui peuvent être invoqués par une ou plusieurs applications actives.

L'ensemble des EEs reposent sur un système d'exploitation de nœud (*NodeOS*). Son unique objectif est de permettre à plusieurs EEs de tourner en parallèle et de gérer l'accès aux ressources du nœud. Le système d'exploitation du nœud réalise l'isolation et la sécurisation des ressources (disque, mémoire vive, CPU, bande passante). Il les alloue et les répartit entre les EEs sur la base de requêtes émises par les EEs. Pour l'allocation et la sécurisation du nœud, le système d'exploitation s'appuie sur un composant de gestion de politique d'allocation et de sécurité, piloté par un EE particulier qu'est l'EE de supervision. La dernière version de la spécification de l'architecture de nœud intègre également le concept d'application active. Ces applications se situent au dessus des environnements d'exécution.

65. Cet intérêt est relevé par les auteurs de la proposition mais considéré comme un intérêt secondaire.
66. Basic Encoding Rules
67. Packet Encoding Rules
68. Certified Encoding Rules

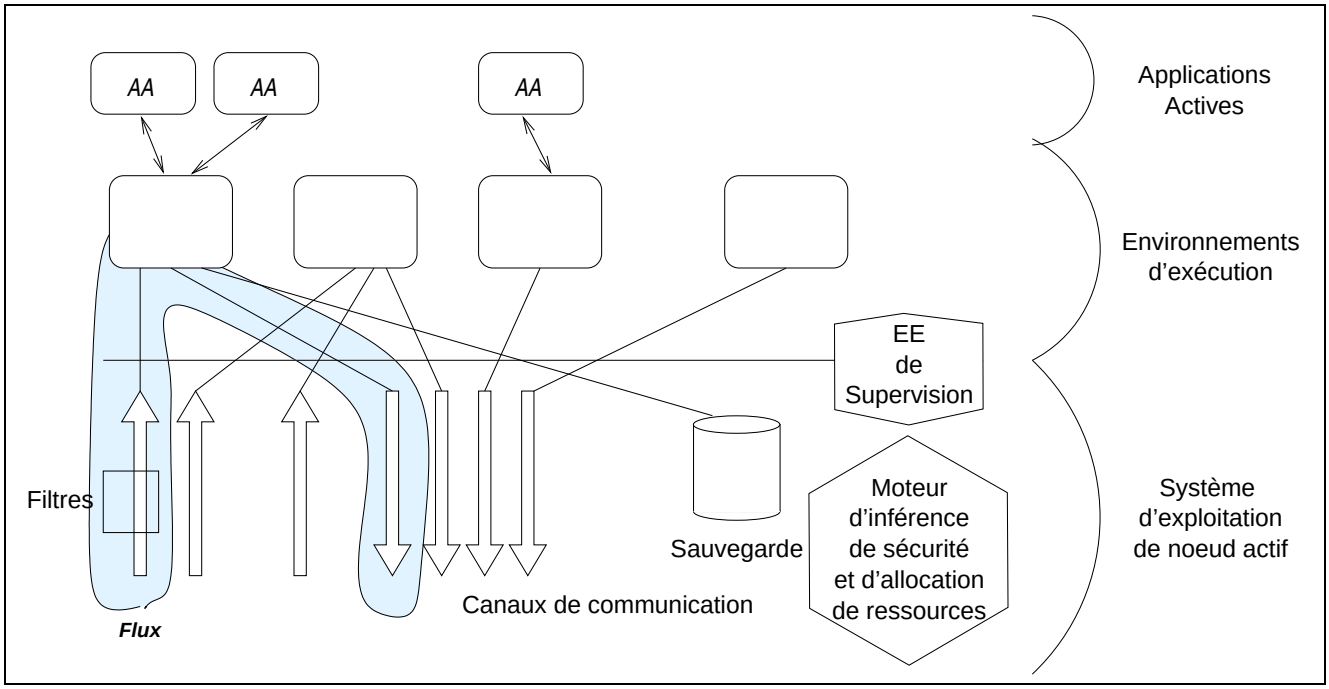

FIG. 34 – *L'architecture standard d'un nœud actif*

Actuellement des efforts importants sont portés sur l'interface de programmation que doit offrir le système d'exploitation de nœud. Une spécification préliminaire[69] est disponible. Elle définit les différentes abstractions d'un nœud actif et spécifie les structures et fonctions pour la manipulation de flux et de ressources par un EE dans le système d'exploitation de nœud.

L'abstraction de base pour l'allocation et la comptabilité des ressources dans le système d'exploitation de nœud est le **flux**. Celui-ci est composé de **canaux de communication en entrée et en sortie** auxquels sont rattachées des ressources (CPU, threads, mémoire) et dont la nature des paquets transmis est spécifiée par un mécanisme de filtre portant sur l'encapsulation des paquets (ex. ANEP sur IP sur ATM en entrée et ANEP sur UDP sur IP sur Ethernet en sortie). Il existe des canaux uni et bi-directionnels ainsi que des canaux de *pass-through* qui ne font que retransmettre en sortie des paquets en entrée.

La figure 35 résume les différents composants du système d'exploitation de nœud. Les ressources du nœud (CPU, mémoire) sont allouées aux flux qui sont manipulés directement par l'environnement d'exécution qui les a demandés. En général un seul flux est attribué par EE mais celui-ci peut contenir des sous-flux pour déléguer à l'OS le démultiplexage des paquets associés.

Des interfaces de programmation pour des différentes ressources sont en cours d'établissement. Pour les flux, des fonctions de création et de destruction sont offertes. Pour les canaux, il en est de même, plus une fonction d'envoi de paquets. Il existe également des fonctions pour la manipulation de threads et l'accès à la mémoire.

Des premiers prototypes de nœud actif *hardware* dont l'architecture interne est proche de cette spécification sont actuellement en cours de développement [CAR 99, LEE 99]. Une réalisation d'un tel nœud, dans le projet CANEs de Georgia Tech est également en cours et la plupart de systèmes d'exploitation de nœuds actifs s'y conforment.

5.4 Architecture de sécurité

Un groupe de travail pour définir une architecture de sécurité a également été créé de manière similaire et complémentaire au groupe de travail sur l'architecture de nœud. Le principal résultat de ce groupe à ce jour est une proposition initiale d'architecture de sécurité dans un nœud. Celle-ci est définie dans [SEC 98]. Ce standard énonce les principes de base que doivent respecter un nœud actif et un réseau actif dans le contexte d'une architecture de sécurité. Elle définit la terminologie utilisée (*credential*, privilège, composition de droits,

69. `http://www.cs.princeton.edu/nsg/papers/nodeos99.ps`

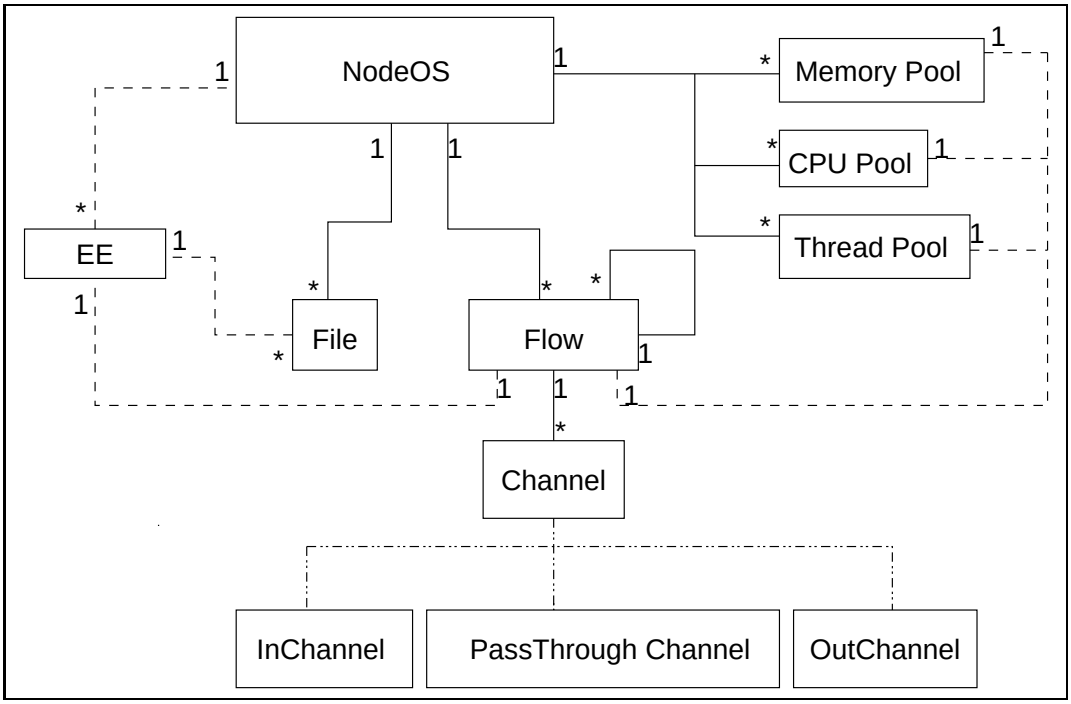

FIG. 35 – *Les composants de l'OS de nœud*

délégation, révocation). Le standard identifie également de manière précise les opérations effectuées par un code actif qui requièrent une attention particulière du point de vue de la sécurité. Ces opérations sont :

- l'accès depuis un paquet actif aux fonctions de traitement et de *forwarding* du nœud ou de l'EE;
- accès depuis un paquet actif à du code résidant sur un nœud ou téléchargeable à distance (un service);
- accès à un paquet depuis un code actif;
- accès depuis un code actif à un autre code actif;
- accès aux ressources du nœud pour modification, destruction, utilisation ;
- enregistrement d'un code actif comme service général pour tous les codes actifs ;
- accès à un paquet ou à un code actif depuis un EE ou l'OS du nœud;
- accès aux ressources du nœud et/ou établissement d'un canal de communication depuis un EE.

L'ensemble de ces opérations doivent être soumises à des contraintes de sécurité fortes. Le document fournit également une définition des différents intervenants, leurs rôles et leurs droits dans l'administration d'un nœud. Ces intervevants sont l'administrateur système, l'administrateur réseau, l'administrateur d'EE, le développeur de services et l'administrateur de sécurité. À chaque rôle sont attribuées des fonctions de sécurité.

La figure 36 illustre les différents composants de l'architecture de sécurité proposée dans le standard. Celle-ci est assez générique pour pouvoir s'adapter à l'ensemble des types d'EE. Chaque nœud actif intègre une base de données de politiques (*policy database*) découplée en trois entités. Une entité est dédiée au nœud dans son ensemble, une est instanciée par EE et une entité spécifique existe par AA. Les identifications des participants sont également stockées dans une base (*credential*). Celle-ci est invoquée lors de la phase de vérification d'autorisation pour un code actif. Toute demande d'accès aux ressources (librairies, interfaces, ...) fait l'objet d'une interrogation de la base de politiques à tous les niveaux (OS, EE, AA).

En plus de l'architecture, le standard propose une spécification d'interfaces de fonctions d'accès relatives à la sécurité. Ces fonctions comprennent :

- des interfaces de profilage de canaux et de flux pour un nœud et un EE ;
- le positionnement de politiques d'allocation de ressources ;
- des interfaces d'autorisation (vérification, calcul de signature, révocation) ;
- des services de cryptographie (signatures DSA, RSA, vérification de signature, hachage MD5, SHA-1, cryptage/décryptrage RSA, DES) ;

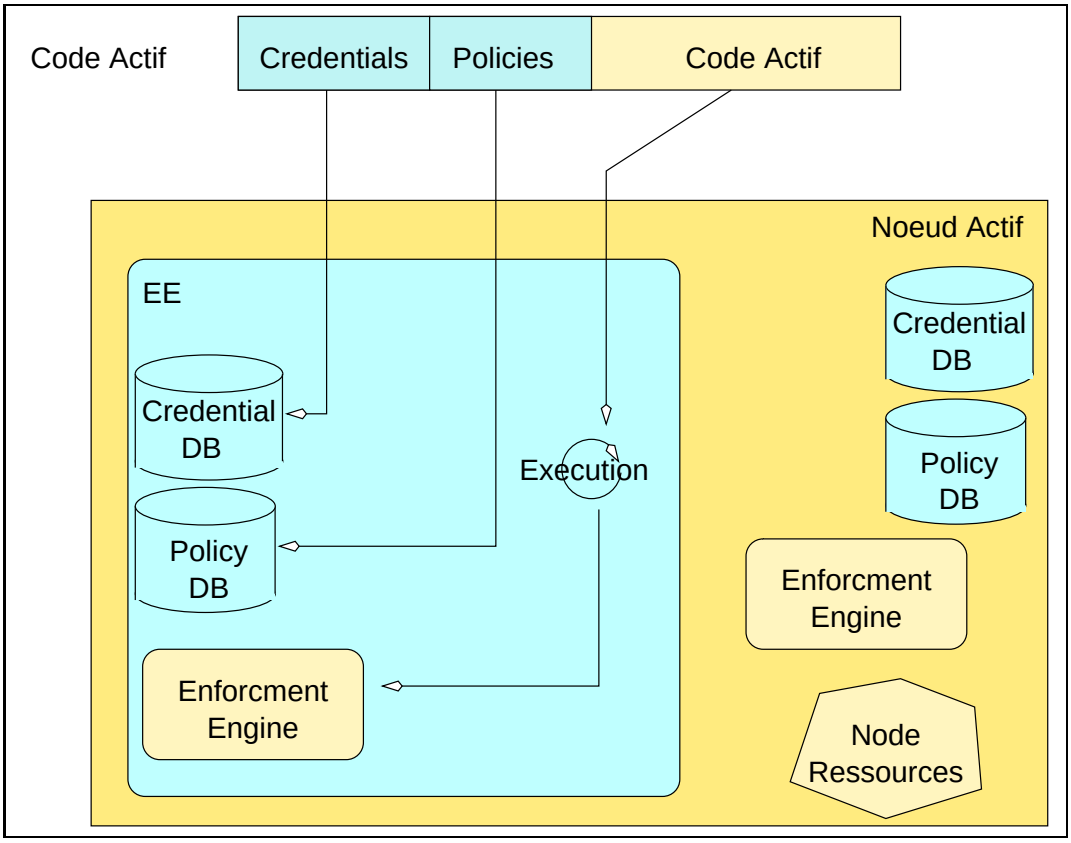

FIG. 36 – *Les composants de l'architecture de sécurité*

– des services de manipulation de clefs (création, enregistrement, conversion de format, validation, révocation, ...).

5.5 Composition de services

Comme les réseaux actifs permettent de déployer dynamiquement de nouveaux services dans le réseau, il est naturel que ceux-ci se heurtent aux mêmes problèmes que ceux rencontrés dans le réseau intelligent. La composition des services en est la parfaite illustration. Dans cet objectif, un groupe de travail s'est créé dans la communauté réseaux actifs et a proposé une première contribution [GRO 98]. Ce document fixe principalement la terminologie (méthode de composition, service, composant de base), et restreint l'étude à la composition au sein d'un unique environnement d'exécution. Quelques exemples de services composites (forwarding conditionnel, filtrage, passerelles de transcodage) sont fournis et les caractéristiques attendues d'une méthode de composition sont définies.

Cinq attributs sont identifiés comme importants dans toute méthode de composition (une méthode est vue ici comme un langage ayant une syntaxe et une sémantique bien définies). Ces attributs sont **le contrôle de séquence** (capacité de décrire la hiérarchie et l'enchaînement des services de base pour un service composé), **le contrôle du partage d'informations** entre services (partage par paramètres, objets partagés, héritage, ...), **le contrôle de la liaison dynamique entre services** (instanciation des différents composants à la création, instanciation à l'exécution dans les nœuds), **les méthodes d'invocation** (lors de l'arrivée d'un paquet, spontanée, à la demande d'un usager) et **la répartition des fonctionnalités** (répartition de la logique de service entre le code dans des paquets actifs et les services de base sur les nœuds par exemple).

D'autres problèmes ouverts tels que le déploiement de services sur les nœuds (*offline, online*), la découverte de services et composants disponibles, le traitement des erreurs, la sécurité et la supervision sont posés dans le document de travail sans fournir de solution actuelle.

Dans le domaine de la composition de services actifs, un projet spécifique est en cours à Georgia Tech. Ce projet est intitulé *LIANE* (*Langage Independent Active Network Environment*). Le principe de base de cette approche est la disponibilité de protocoles ou services à trous. Ces services sont des services génériques qui

peuvent être adaptés à différents points précis de leur exécution par association de *slots* à d'autres programmes fournis par l'usager. Par exemple, on peut définir un service générique de filtrage de paquets et mettre un *slot* sur le traitement de paquets filtrés ainsi qu'un *slot* sur le traitement de paquets qui n'ont pas passés le filtre. Ainsi un usager peut associer par exemple un service d'enregistrement au *slot* de traitement des paquets filtrés et un service de *dropping* des paquets qui n'ont pas passé le filtre.

Le service générique définit le séquencement des services supplémentaires (affectés à des *slots*). La sélection des *slots* se fait par le paquet entrant, soit en fournissant le code de ces *slots*, soit en donnant des références aux services à invoquer lors du déclenchement d'un *slot*. Un modèle formel de ce mécanisme de composition par allocation de *slots* est défini en UNITY dans [BHA 98a].

5.6 Résumé

La standardisation autour des réseaux actifs a débuté en juin 1997. Aujourd'hui elle a abouti à la définition d'un format de paquets commun respecté par la plupart des concepteurs d'EE et forme la base de l'*ABone* actuel. L'architecture de nœud sert également de référence aux développeurs de systèmes d'exploitation dédiés aux réseaux actifs. De plus, elle a permis de clarifier la terminologie des composants dans un réseau actif. Les travaux sur la sécurité et la composition de services progressent lentement mais devraient aboutir à terme à une définition normalisée comprenant, pour la partie sécurité, les interfaces d'un OS de nœud.

6 Les applications des réseaux actifs

6.1 Introduction

Les nouvelles applications qui émergent à l'heure actuelle nécessitent l'exploitation des caractéristiques de nouveaux services qui ne sont pas présents en tant que tels dans les réseaux actuels. Comme il a déjà été mentionné dans les sections précédentes, le contrôle dynamique rendu possible par l'introduction des réseaux actifs permet de confectionner et d'adapter des services par rapport à l'état courant du réseau. Ces services « sur mesure » ont le potentiel d'améliorer les performances vues par une application, ce qui n'est pas le cas dans un réseau classique n'offrant qu'une solution de bout-en-bout. Le but de cette section est de présenter différents exemples mettant en avant l'avantage et l'efficacité des réseaux actifs. Les efforts engagés pour améliorer les performances au moyen des réseaux actifs couvrent un large spectre d'applications. La majorité des exemples d'applications ou de protocoles que nous présentons et qui tirent partie des infrastructures programmables et de la mobilité de codes proviennent du domaine des applications dites « multimédia » ou, tout du moins, peuvent être employés pour améliorer ces dernières.

La question qui se pose et qui reste d'actualité est de savoir si les changements induits par l'approche réseau actif vont permettre de réellement accroître les performances des applications qui s'exécutent au sein d'un réseau [CAL 98, PSO 99]. Il faut bien admettre qu'il n'existe pas encore de réponse définitive à cette interrogation et que les avis restent partagés, voire tranchés pour certains. Un des principaux arguments avancés contre le concept de réseau actif est que la réussite de l'Internet tel que nous le connaissons à l'heure actuelle est en grande partie due à sa simplicité.

Avant de présenter les points clefs et les principales caractéristiques des applications traitées, nous allons détailler les commentaires de divers chercheurs sur leur vision de la relation entre les réseaux actifs et ce qu'il convient d'appeler l'argument de bout-en-bout. Nous étudierons ensuite les solutions proposées et les technologies expérimentales mises en œuvre à l'heure actuelle et évaluerons l'utilité et l'apport de la technologie réseau actif.

6.2 Le principe de bout-en-bout.

Ce principe, aussi nommé « argument de bout-en-bout » [SAL 84], stipule que les décisions finales doivent toujours être du ressort de l'utilisateur et que tenter de le suppléer par de l'intelligence au sein du réseau devient redondant. Ceci implique que les fonctions de contrôle doivent, dans la mesure du possible, toujours être placées à l'extérieur du réseau [HUI 94].

Selon ce principe, un protocole ne doit être déployé au sein d'un réseau que s'il est réellement nécessaire de le mettre en œuvre à ce niveau. Un protocole de routage se conforme bien évidemment à ce principe. À l'inverse, un protocole de transmission fiabilisée ne rentre pas dans le cadre de cette définition. En effet, la mise en œuvre d'un tel protocole implique que des applications, n'ayant pas besoin de la fiabilité, vont se voir pénalisées notamment en terme de latence. De plus, les applications nécessitant la fiabilité mettent le plus

souvent en œuvre un protocole spécifique et donc, l'effet du réseau devient par là-même redondant. Si certains auteurs pensent que l'approche active remet en cause le principe de bout-en-bout, d'autres défendent la thèse inverse et soutiennent que les réseaux actifs abondent dans le sens du principe de bout-en-bout. Dans le numéro spécial *Active and Programmable Networks* paru dans *IEEE Network* [CHE 98], plusieurs points de vues sont confrontés.

Le premier commentaire est donné par BHATTACHARJEE, CALVERT et ZEGURA qui tirent la conclusion que les réseaux actifs n'entrent pas en conflit avec le principe de bout en bout. Ils défendent leur position déjà exposée dans [BHA 97a] selon laquelle, l'approche active est une conséquence « naturelle » du principe de bout-en-bout car différents services ne peuvent être efficacement déployés et améliorés que s'ils disposent d'informations uniquement disponibles et accessibles au sein du réseau. Parmi ces informations, on peut citer l'instant et le lieu où survient une congestion, l'emplacement des points de concentration de trafic (*i.e.*, point du réseau où des requêtes sont fortement corrélées dans le temps et l'espace), les endroits où surviennent des pertes de paquets dans un arbre de multicast... À leur tour, les applications peuvent détenir des informations se révélant fort utiles au réseau pour optimiser les performances. Parmi ces informations, on peut citer l'existence de certaines dépendances entre les données de l'application, les différentes hiérarchies d'importance entre données, la possibilité de répondre à une requête en utilisant des données cachées...

Le second commentaire est donné par PARTRIDGE, STRAYER, SCHWARTZ, et JACKSON qui évaluent l'utilité des réseaux actifs dans le contexte de bout-en-bout pour chaque niveau des couches du réseau Internet. Selon leur étude détaillée couche par couche, il ressort que le principe de bout-en-bout va à l'encontre d'une mise en œuvre de l'actif dans la couche Internet. Le but de l'Internet est d'offrir une connectivité universelle et un moyen de communication entre un très grand nombre d'équipements hétérogènes. Les auteurs de cet article pensent que les réseaux actifs ne peuvent pas améliorer ce service alors qu'il est facile de trouver des cas pour lesquels les réseaux actifs vont mettre à mal ce service : puisque le chemin d'un paquet est affecté par le code qu'il véhicule (voire par le code d'un autre paquet) les chances qu'un paquet arrive à destination est réduite du fait qu'il peut se trouver sur un environnement d'exécution endommagé, non à jour, que les programmes véhiculés soient erronés et que le débogage soit rendu extrêmement délicat. Leur analyse reste en outre très favorable pour toutes les autres couches avec un léger bémol pour la couche transport qui doit garantir un comportement sous certaines conditions. En particulier, la réaction du protocole de transport en cas de congestion doit respecter certaines contraintes. Le problème réside dans le fait qu'imposer un comportement à un programme reste une question ouverte.

Le dernier commentaire revient à REED, SALTZER et CLARK qui réexaminent leur argument de bout-en-bout à la lumière du nouveau contexte introduit par les réseaux actifs. Ils remarquent que pour conserver un niveau de transparence élevé au sein des réseaux, le principe de bout-en-bout requiert que la sémantique de toute fonctionnalité active doit être définie et confinée avec beaucoup de précautions de telle sorte que les interactions entre différents utilisateurs partageant une même couche basse doivent pouvoir être prédites par le concepteur qui utilise les services et les fonctionnalités de la couche active. Un manque de prédiction peut en effet entraîner un surcoût pour tous les autres utilisateurs, y compris ceux ne faisant pas appel aux fonctionnalités actives. Caractériser de façon exacte la sémantique des améliorations apportées par l'actif, devient alors le défit à relever étant donné qu'une sémantique approximative d'une amélioration peut se révéler pire que de n'avoir aucune amélioration. Même si l'idée de l'actif n'est pas complètement mise à mal par le principe de bout-en-bout, les auteurs n'ont pas trouvé d'exemples mettant à jour l'impact indéniable de l'actif tout en garantissant une sémantique simple, flexible et transparente nécessaire pour une utilisation dans les couche basses des réseaux.

Dans [DEL 99] les auteurs étudient les interactions qui peuvent survenir entre plusieurs applications s'exécutant au sein d'un réseau actif. Ils proposent une classification des interactions qui doit permettre d'en exploiter certaines, car elle peuvent être bénéfiques, et de se protéger des interactions indésirables. Ils proposent trois niveaux d'interférences entre applications : (1) *aucune interférence*, une application requiert un niveau de sécurité élevé et n'accepte aucune interférence sur ses paquets ; (2) *interférence intra-application*, ce qui permet de se prémunir contre les interférences provenant des autres applications ; (3) *interférence inter-applications*, une application accepte que des entités du réseau puissent modifier ses paquets. Ils distinguent des interférences la notion de communication entre les applications et proposent aussi trois niveaux de communication : (1) *aucune communication* ; (2) *communication intra-application*, c'est le cas d'une application multi-tâches ; (3) *communication inter-application* lorsque deux applications se mettent d'accord sur l'échange de données entre elles. Bien que les communications puissent être vues comme une interaction mutuelle, la notion d'interférence induit un comportement passif de la part d'une des entités en question.

6.3 Supervision et contrôle

La supervision et le contrôle des réseaux sont certainement les domaines de prédilection des réseaux actifs et programmables. Le nombre important de projets et d'architectures dans ce domaine ne nous permet pas de les détailler tous dans ce rapport. Ils sont présentés dans un rapport de recherche spécifique en cours de rédaction dans le projet RESEDAS.

6.4 Communications de groupe (Multicast)

La notion de groupe est omniprésente dans la société et l'emploi d'un nom collectif pour désigner un sous-groupe de personnes permet de référencer ou d'adresser une partie de la population comme s'il s'agissait d'une même entité. De façon quasi similaire, cette notion de groupe peut être employée dans les systèmes distribués afin d'aider à la compréhension de larges applications [POW 96]. Un besoin émergeant et devenant primordial en matière de communication réside dans la mise en œuvre de protocoles de communications de groupe appelé aussi communication multi-points. Par communication de groupe, on entend une communication qui fait intervenir plus de deux participants désirant échanger des informations. Nous allons voir dans les sections suivantes (6.5 et 6.6) que beaucoup d'applications peuvent bénéficier des communications de groupe. Il apparaît donc primordial de pouvoir satisfaire aux besoins des communications de groupe par le déploiement dans le réseau de nouveaux services qui soient performants, fiables et extensibles. Il est vrai que ces derniers peuvent être développés au niveau des systèmes terminaux, mais il est admis que les implanter au sein même de la couche réseau, engendre généralement de meilleurs résultats.

Divers travaux ont été entrepris sur l'étude de protocoles de multicast dans les réseaux actifs. WETHERALL a présenté dans sa thèse [WET 99c] la mise en œuvre d'un protocole inspiré de PIM (*Protocol Independant Multicast*) [EST 97] afin de montrer que les réseaux actifs pouvaient être facilement employés pour développer et déployer de réels protocoles et ce malgré leur complexité. Ces travaux ont aussi permis de montrer qu'introduire plusieurs modifications aux protocoles originels est une chose facilement réalisable par l'ajout de quelques capsules et que les modifications apportées restent simples d'utilisation.

Les réseaux actifs sont aussi utilisés pour fiabiliser les protocoles de multicast [LEG 98b, LEH 98] et permettent de résoudre de façon élégante les problèmes dûs à l'implosion d'acquittements négatifs (`NACK`), aux retransmissions inutiles, à la concentration de trafic due aux retransmissions, aux duplications de paquets et au fait que le groupe de membres est fortement dynamique. Dans la solution nommée « Active Reliable Multicast » (ARM) proposée dans [LEH 98], les noeuds vont jouer un rôle actif dans le mécanisme de fiabilité :

- ils vont cacher les données transmises afin de pouvoir les réémettre au plus tôt en cas de requête de retransmission ;
- ils vont traiter les NACKS afin d'optimiser les demandes de retransmission et de collecter les informations concernant l'auteur de la demande de retransmission.

Il semble que les réseaux actifs offrent des améliorations prometteuses dans le domaine des protocoles de multicast fiable car ils font appel à un point clé de ce genre de protocole qui est la possibilité de prendre une décision au sein même du réseau.

Une autre expérimentation a été réalisée dans le projet AMnet [WIT 98a, WIT 98b, WIT 98d]. WITTMANN, KRASNODEMBSKI, et ZITTERBART proposent un support pour les communications multi-points (multicast) qui intègre un système de filtres de QoS (*Quality of Services)* permettant de supprimer de l'information des flux multimédia afin de réduire le débit pour les récepteurs à faible bande passante sans pour autant affecter les récepteurs ayant une large bande passante. L'architecture de leur nœud est active dans le sens où elle comprend des filtres de QoS (MPEG-1) et de la signalisation pour mettre en œuvre la QoS (signalisation basée sur RSVP).

Un des problèmes d'implantation d'un protocole multicast est le choix de la structure de diffusion de l'information de groupe. Comme un groupe de multicast est hautement dynamique, la structure, généralement un arbre de recouvrement des membres utilisé pour diffuser l'information, doit être à géométrie variable. Il apparaît donc nécessaire d'avoir un mécanisme assurant sa gestion. La gestion de cet arbre est une tâche difficile et les critères d'optimalité sont nombreux voire incompatibles entre eux. Il faut prendre en compte les besoins de l'application, la disponibilité des ressources de transmission ainsi que la charge de trafic au sein du réseau puisque l'optimisation de la diffusion au groupe revient à optimiser le routage multicast dans le réseau. De plus, il faut prendre en compte l'individualité, c'est à dire la capacité à tenir compte des besoins différents des utilisateurs du fait de leur non homogénéité. Dans [KOU 99], un protocole de gestion d'un arbre de multicast de type PIM est présenté. Il permet de faire migrer l'arbre de multicast (*i.e.*, de changer le *core* aussi nommé point de *rendez-vous*, et de reconfigurer l'arbre) quand ce dernier n'est plus adapté à l'état courant du réseau. Le concept d'actif permet à l'application de déployer elle-même sa propre fonction de coût qui va servir à évaluer

les performances de l'arbre multicast. Une application peut par exemple chercher à minimiser le maximum des délais entre les membres, le délai moyen ou les ressources utilisées dans le réseau... Ce travail a permis de montrer une fois de plus que l'on pouvait facilement et rapidement développer et déployer un réel protocole de multicast complexe (PIM + migration) et que ce protocole peut réellement tenir compte des besoins des applications qui fournissent elles-mêmes la fonction de coût à appliquer à l'arbre de multicast ; fait qui est totalement spécifique aux services offerts par les réseaux actifs.

6.5 Applications multimédia audio/vidéo

La distribution de flux multimédia, que cela soit audio ou vidéo, à tous les membres d'un groupe de multicast est un problème ardu du fait de l'hétérogénéité des chemins au sein du réseau entre l'émetteur et les récepteurs mais aussi à cause du caractère extensible, puisque l'on doit pouvoir supporter un grand nombre de récepteurs. Pour résoudre le problème de l'hétérogénéité et de l'extensibilité, l'approche classique consiste à employer une méthode de régulation orientée récepteur. Dans cette approche, la source émet plusieurs flux qui sont appropriés pour des classes de chemins du réseau dont les caractéristiques et les débits appartiennent à un certain éventail de valeurs. Chaque récepteur s'abonne au flux qui peut être supporté par le chemin le reliant à la source. L'approche réseau actif offre une autre alternative sur l'endroit où l'adaptation et la régulation du flux peuvent être effectuées, *i.e.*, au sein du réseau.

L'idée générale est que la source émet un seul flux à destination d'un seul groupe de multicast auquel vont s'abonner tous les récepteurs. Les routeurs de l'arbre de multicast devront détenir des informations leur permettant de réduire le flux en cas de congestion d'un lien aval de façon « intelligente ». Les nœuds internes de l'arbre de multicast qui effectuent des duplications, ne font la régulation de flux qu'après avoir dupliqué le paquet et ce, afin de pouvoir traiter chaque lien aval de façon indépendante. Ceci permet de garantir que chaque récepteur reçoive un flux en accord avec les caractéristiques du chemin qui le relie à la source. La régulation est directement imposée par les conditions d'encombrement dans le réseau, permettant ainsi de répercuter immédiatement toute augmentation ou baisse de la bande passante. La mise en œuvre de deux protocoles de multicast de flux multimédia a été réalisée par Branchs, Effelsberg, Tschudin et Turau dans [BRA 98] sur les plates-formes Messenger et ANTS. Le premier protocole, « *Robust Multicast Audio* » consiste en un protocole robuste de diffusion de flux audio permettant d'accroître la qualité de l'audio transporté dans un réseau de type *best-effort*. Le codage du flux est basé sur des ondelettes (*Wavelet Audio Radio*). Les paquets audio sont regroupés en séquences composées de $N + R_i$ paquets audio où N est le nombre originel de paquets audio et R_i est le nombre de paquets redondants spécifiques au lien i. Un nœud actif va attendre jusqu'à ce que tous les paquets d'une séquence donnée soient reçus. Si L paquets sont perdus, il va reconstruire le plus de paquets possible des N originels. En fonction du lien sur lequel il va retransmettre les paquets de la séquence qu'il vient de reconstruire, il va ajouter un certain nombre de paquets redondants. Le second protocole « *Layered Multicast Video* » permet d'optimiser la bande passante au sein de l'arbre de diffusion multicast en encodant les données suivant différents niveaux (le niveau 0 correspondant à une qualité minimale et le niveau $i+1$ ajoutant plus de qualité au niveau i). Chaque paquet transporte son niveau facilitant la réalisation du filtre au sein des nœuds de l'arbre de multicast puisqu'il suffit de ne pas retransmettre un paquet sur un lien aval si le niveau de qualité du paquet est supérieur à la qualité maximale que peut supporter le chemin en aval. La mise en œuvre de ces protocoles a démontré la facilité à adapter et améliorer des protocoles déjà existants et à en déployer de nouveaux. Les performances furent suffisantes pour transmettre un flux multimédia et montrer que les réseaux actifs ne sont pas uniquement utiles pour mettre en œuvre des protocoles de signalisation ou de gestion, mais peuvent servir au sein d'un protocole expérimental tout au long du chemin des données tant que le traitement sur chaque paquet reste « simple ».

Le problème plus général qui risque de perdurer dans le futur, et auquel la transmission de flux multimédia doit faire face, est la congestion que l'on rencontre dans les réseaux. Il est donc crucial de trouver des remèdes algorithmiques pour pouvoir y remédier. Les réseaux actifs apparaissent comme de bons candidats pour résoudre ce problème puisque la congestion est un phénomène qui, par nature, se déroule au sein même des réseaux souvent loin du lieu où s'exécute l'application. Il est bien souvent difficile de faire remonter des informations auprès de l'usager afin que le trafic se régule de lui-même et que la congestion disparaisse. Sans être exhaustif, on peut énoncer divers exemples d'utilisation des réseaux actifs :

- un nœud actif peut monitorer la bande passante disponible et contrôler en conséquence le flux de données, ce qui implique bien évidemment de devoir bufferiser des données non plus au niveau des commutateurs (*switchs*) mais dans les nœuds actifs ;
- dans le cas où coexistent plusieurs flux avec diverses exigences, un nœud actif peut contrôler le taux de transmission pour chacun des flux en fonction du flux total disponible ;

- si les applications peuvent dans certains cas s'adapter aux conditions du réseau et donc modifier les données transmises en conséquence, il est alors possible de délocaliser cette modification à l'endroit même où apparaît la congestion ;
- il devient possible d'effectuer un tri sélectif parmi les paquets d'un même flux. C'est le cas typique d'un flux vidéo compressé MPEG pour lequel, si l'on perd une trame de type `I`, il ne sert à rien de transmettre les trames `P` et `B` qui sont corrélées à la trame `I` perdue ;
- pour finir, il est possible de tenir compte de l'interaction entre plusieurs flux. Si un utilisateur reçoit un flux vidéo et un flux audio et s'il y a des pertes dans le flux vidéo, il semble préférable de donner toute la priorité au flux audio afin de continuer à acheminer de l'information utile à l'utilisateur.

Les travaux réalisés au Georgia Institute of Technology [BHA 97b, BHA 99] reprennent certains des points listés ci-dessus et montrent l'attrait des réseaux actifs pour résoudre les problèmes liés à la congestion dans les réseaux.

6.6 Application au Web

Dans leur article [WET 98a], WETHERALL, LEGEDZA, et GUTTAG présentent plusieurs exemples simples d'applications qui bénéficient des nouveaux services pouvant être introduits par l'actif. Ces exemples possèdent le même domaine d'application, à savoir Internet.

Quotations en directe. Le premier exemple concerne l'obtention de quotations boursières via le Web. Obtenir un accès rapide à des informations à jour est un point crucial surtout durant les périodes où le serveur est chargé puisque cela signifie le plus souvent une forte activité des marchés. Les techniques actuelles de cache s'avèrent mal appropriées dans ce cas. Les caches Web ne stockent pas ce genre d'information car ce sont des données dynamiques et la granularité des objets stockés dans les caches Web (*i.e.*, une page Web entière) est inappropriée car chaque boursicoteur génère une requête avec quelques titres d'actions qui lui sont propres, et donc sur plusieurs centaines de titres disponibles, le nombre de combinaisons de pages Web uniques est énorme. Dans une approche réseau actif, on peut imaginer un système de caches mettant en œuvre une stratégie dédiée aux besoins de l'application. Le protocole actif va tout d'abord cacher les titres dans les nœuds du réseau et ce, avec une granularité fine, ce qui permet de répondre à toutes les requêtes concernant des titres en vogue (et donc présents dans les caches) sans tenir compte de l'ordre des titres dans lequel la requête globale a été émise. Le protocole peut ensuite permettre d'associer à toute requête concernant un titre une périodicité minimale de mise à jour. L'idée générale est donc de cacher les quotations dans les nœuds du réseau. Les requêtes d'autres clients seront ensuite interceptées dans les nœuds pour chercher si elles peuvent être satisfaites localement avec les données du cache.

Vente aux enchères. Le second exemple est basé sur les sites Internet permettant d'acheter divers produits aux enchères (voyages, matériel informatique...). Un serveur proposant ce genre de service doit collecter toutes les offres des clients et il doit aussi informer le client du prix actuel de l'objet convoité. Du fait des délais dans le réseau, un client peut soumettre une offre qui est en dessous du prix actuel de l'objet. Le serveur, contrairement aux ventes aux enchères « classiques » devra rejeter explicitement des offres trop basses surtout lorsqu'il y a une forte concurrence pour un objet. À l'heure actuelle, toutes les offres sont collectées et traitées par le serveur. Les réseaux actifs peuvent permettre de filtrer les offres qui ne sont pas assez fortes au sein du réseau. Ainsi, quand un serveur s'aperçoit qu'il devient surchargé, il peut activer le mécanisme de filtrage dans des nœuds d'un certain voisinage qui vont rejeter les offres trop basses tout en prévenant le client. Le serveur devra tout de même mettre à jour les prix des articles périodiquement dans chacun des routeurs actifs.

Les deux exemples sont assez similaires et ont en commun l'utilisation de caches au sein du réseau pour accroître les performances des applications. Cependant, dans le second exemple, le protocole est, par essence, différent puisqu'il délègue aux nœuds actifs certaines tâches qui incombaient au serveur : rejeter des offres et prévenir le client.

Caches Web. Un système de caches Web adaptatif, extensible et robuste devient crucial afin de pouvoir traiter la croissance exponentielle et le comportement hautement dynamique du « World Wide Web » (la toile). Une part importante du trafic Internet provient d'applications (client Web par exemple) qui vont récupérer des informations sur des serveurs localisés à tout endroit du réseau. Le fait de cacher des objets dans un voisinage proche des clients permet de faire décroître le trafic généré au sein du réseau et le temps nécessaire pour récupérer ces objets. La problématique majeure d'un système de caches est de pouvoir localiser un objet

et de savoir retransmettre des requêtes entre les caches. L'infrastructure de cache Web dominante à l'heure actuelle dans l'Internet consiste en une hiérarchie statique de caches SQUID [70] et les clients Web sont configurés manuellement pour accéder à un cache particulier de la hiérarchie (le proxy du client). Si une requête ne peut pas être satisfaite au niveau du proxy, les caches de même niveau ainsi que le cache père sont contactés. Si la requête n'est pas satisfaite dans les caches de même niveau et dans le cache père, une nouvelle requête HTTP avec comme cible le cache père est générée. Ce processus se répète ainsi récursivement dans toute la hiérarchie jusqu'à ce que la requête soit satisfaite ou qu'elle atteigne la racine de la hiérarchie qui va récupérer l'objet dans le serveur origine. Une fois l'objet récupéré, il est caché tout au long du chemin inverse le menant vers le cache proxy.

Les limites d'un tel système résident dans le fait qu'il est statique, que l'on doit le configurer manuellement et que la racine de la hiérarchie peut devenir un goulot d'étranglement. Comme nous venons de le constater dans les exemples précités, les réseaux actifs peuvent jouer un rôle dans ce cadre et divers travaux ont été réalisés dans ce contexte pour faire évoluer et améliorer l'état actuel des techniques de cache Web. La solution proposée dans [LEG 98a] consiste à maintenir une table d'indirection au sein même des nœuds du réseau afin de pouvoir diriger les requêtes concernant des documents très demandés vers le cache le plus proche. On peut aussi utiliser des services offerts par les réseaux pour router les requêtes vers des nœuds préconfigurés comme caches [LEG 98b], ce qui revient à combiner la recherche dans un cache et le routage de la requête [WET 99c]. Mettre en œuvre et maintenir une distribution uniforme de petits caches ayant des informations sur le contenu des caches présents dans leur voisinage [BHA 98b] fait aussi partie des solutions proposées. Dans [VAN 99], les auteurs proposent une architecture de cache Web dans laquelle les objets sont vus avec une granularité assez fine et sont cachés avec une estampille temporelle. Au lieu de considérer les dates de façon individuelle, ils définissent des classes temporelles permettant de regrouper des objets répondant à des critères temporels de même ordre. Chaque nœud intermédiaire ne traite qu'une seule classe temporelle et les caches sont organisés selon une topologie en arbre, ce qui permet de mieux répartir les données suivant leur appartenance à telle ou telle classe.

Le projet « Adaptive Web Caching » [SCO 98] consiste à proposer des solutions pour offrir un système de caches Web adaptatif, extensible et robuste et peut être réalisé à partir de réseaux actifs ou de protocoles de niveau applicatif. Le système de caches proposé est adaptatif dans le sens où ce projet se fonde sur plusieurs serveurs de caches qui s'auto-organisent en une grille interconnectant plusieurs groupes de multicast non disjoints, permettant de la sorte de diffuser efficacement le contenu de sites Web très demandés.

6.7 Applications diverses.

6.7.1 Attaque par inondation de `TCP SYN` (*SYN-flooding*).

Ce type d'attaque [RIC 99] exploite une faille présente dans le protocole de gestion de connexion employé par TCP qui est construit autour de 3 phases (*3-way handshake*). Le vil attaquant submerge un serveur cible avec un flot de paquets `TCP SYN` (correspondant à des demandes de connexion) et remplit ainsi la file des demandes de connexion en attente puisque à ce stade une connexion « à demie ouverte » existe. Le serveur doit conserver au sein d'une structure de données appropriée toutes les données nécessaires pour compléter ultérieurement cette connexion. Si le client répond avec un `ACK` la connexion est établie. Dans le cas contraire, le serveur doit attendre un certain laps de temps (le plus souvent 75 secondes) avant de pouvoir supprimer la tentative de connexion. Dans la plupart des cas, un serveur ne peut pas se défendre contre les attaques de type inondation de `TCP SYN` souvent couplées à de l'*IP Spoofing* (adresse IP bidon) car il lui est très difficile de distinguer un paquet valable d'un autre factice. De plus, il est difficile, voire impossible, de trouver l'auteur d'une telle attaque car il n'existe pas de moyen de découvrir la réelle source des paquets factices car les routeurs du réseau n'ayant pas la notion de chemin inverse pour remonter de la destination vers la source. Dans [VAN 97], VAN présente la mise en œuvre d'un système développé à partir de l'environnement ANTS qui permet de détecter les attaques de type `SYN`-flooding et de déployer dynamiquement un mécanisme de défense permettant de filtrer les capsules factices au sein même du réseau. Le filtre va s'assurer que l'adresse source des capsules n'est pas falsifiée. Dès qu'une contrefaçon est détectée, les capsules falsifiées sont éliminées et le filtre est progressivement repoussé au sein du réseau pour s'approcher aussi près de l'adversaire que possible. Cette approche met en valeur certains aspects bénéfiques des réseaux actifs par comparaison à une approche classique basée sur un filtrage effectué uniquement dans les routeurs en entrée du domaine (*Ingress routeur*) : elle permet de filtrer tous les paquets, même ceux provenant du domaine du serveur. Le filtre n'est déployé que quand il a lieu de l'être, *i.e.*, durant une attaque et il ne se déploie que dans les parties du réseau concernées par l'attaque.

70. `http://squid.nlanr.net/Squid`

6.7.2 Pont actif.

Dans [ALE 97b], les auteurs présentent leurs travaux concernant l'étude, la mise en œuvre et les tests de performance d'un nouvel élément introduit dans les réseaux qu'ils nomment « *active bridge* ». Il s'agit d'un pont qui peut être reprogrammé à la volée au moyen de modules nommés *switchlet* que l'on peut charger dynamiquement. Ceci permet de mettre à jour un réseau qui utilise une version périmée d'un protocole et d'employer une nouvelle version de ce protocole de façon transparente. Les réseaux actifs permettent dans l'exemple présenté de tester le nouveau protocole et de revenir en arrière vers l'ancienne version du protocole si des erreurs sont constatées. Ceci permet de protéger le pont actif des erreurs de programmation algorithmique pouvant se trouver dans les switchlets. Cet article présente tout d'abord le cas d'un pont actif configuré simplement comme répéteur. Ce pont est ensuite mis à jour via les switchlets afin d'ajouter les fonctionnalités d'auto apprentissage des tables de routage, de permettre d'exécuter des protocoles d'arbre de recouvrement et d'adapter le choix du protocole d'arbre de recouvrement que l'on doit employer en fonction de certaines informations présentes dans les trames Ethernet.

7 Les logiciels disponibles

La plupart des projets autour des réseaux programmables et/ou actifs ont donné lieu à la réalisation de logiciels disponibles sur Internet. Cette section a pour objectif de référencer ces composants.

7.1 Environnements d'exécution

Tous les projets présentés dans ce rapport offrent des environnements d'exécution associés. Ceux-ci sont accessibles via la page web RAF[71].

7.2 Composants ANEP

De nombreux projets disposent d'une souche de traitement et/ou de démultiplexage de paquets ANEP. Ceux-ci sont également disponibles sur le site RAF.

7.3 Composants SAPF

Il existe aujourd'hui un codec SAPF[72] disponible sur le Web. Ce codec est programmé en langage C.

7.4 Réseaux virtuels

7.4.1 Le ABone

Le ABone est un réseau virtuel actif international construit sur des tunnels IPv4. Il relie aujourd'hui plus de cent machines à travers le monde et permet aux organisations qui y adhèrent de déployer et d'expérimenter des protocoles actifs. Au départ, ce réseau a été sponsorisé par le DARPA afin de permettre l'expérimentation à grande échelle de services et protocoles actifs. Une présentation de l'architecture de ce réseau est donnée dans [BER 99]. Ce document précise la définition des applications actives (AA), EE et NodeOS dans le contexte de l'ABone. Il comprend notamment une définition des AA non fournie dans la norme initiale de l'architecture de noeud actif.

Définition 8 *Une application active (AA) est une application dédiée à un ou plusieurs usagers qui réalise des fonctions de communication et opère soit dans le plan de données, dans le plan de contrôle ou dans le plan de supervision.*

Une AA s'exécute toujours dans le contexte d'un environnement d'exécution.
Les composants du ABone sont les suivants:

- les noeuds core : ces nœuds ont vocation à être très stables, partagés par l'ensemble de la communauté et à ne supporter que des EE permanents toujours disponibles aux développeurs d'applications actives. Cela implique un support de la part des développeurs de ces EE ainsi que leur bon fonctionnement en

71. `http://www.loria.fr/~festor/RAF/RAF.htm`
72. `http://ryyppy.ifi.unizh.ch/~sapf`

toute condition (même après un crash d'un nœud, les EE permanents sont relancés automatiquement). Le déploiement d'EE d'expérimentation sur ces nœuds est possible mais non maintenu par le centre de coordination (ABOCC[73]) ;

– les nœuds frontière : ces nœuds sont des noeuds privés supervisés par les personnes ou organisations qui les détiennent. Contrairement aux nœuds core, ils ne sont pas assujettis à une disponibilité permanente et non supervisés ni maintenus par le centre de coordination.

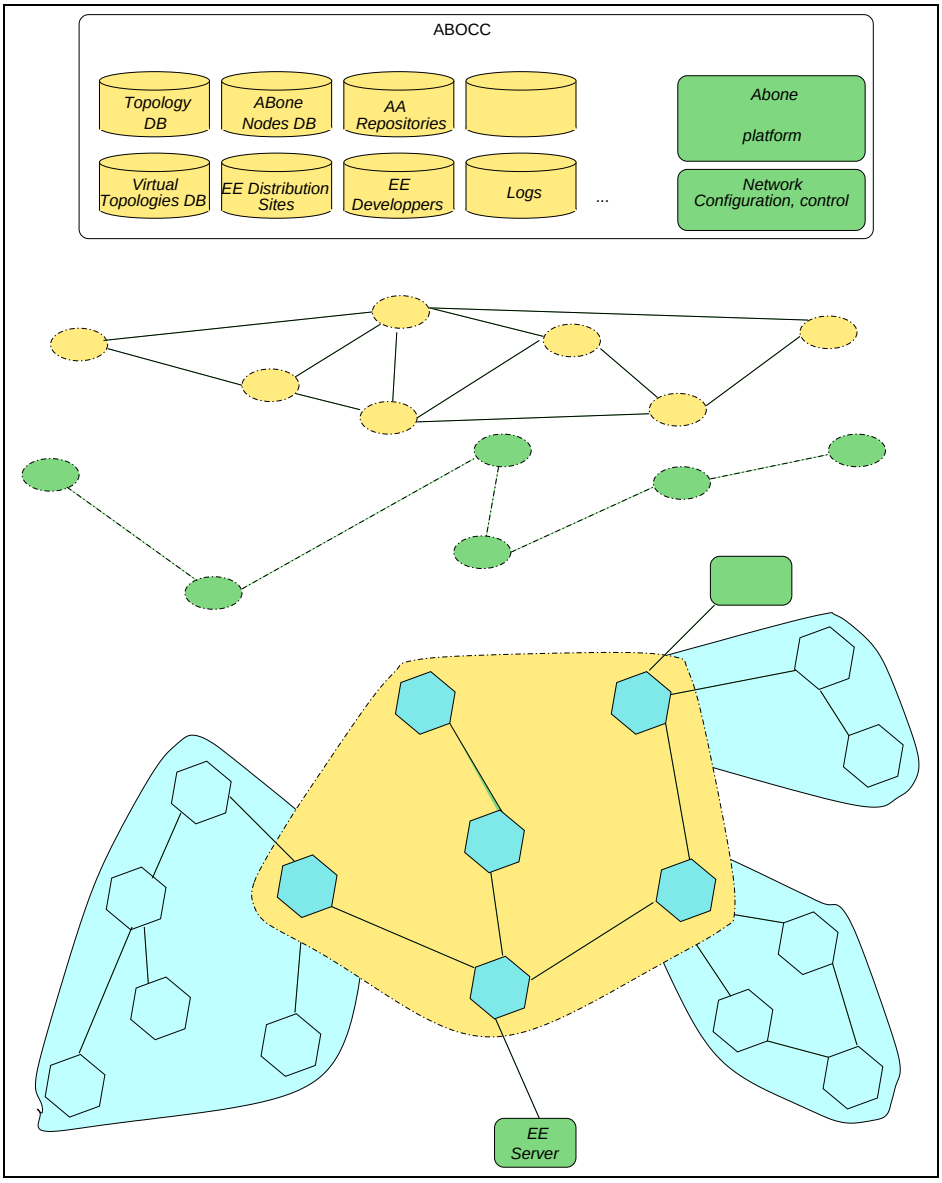

FIG. 37 – *Les composants d'un futur ABone*

Afin d'exploiter les nœuds core et les EEs permanents et d'éviter les conflits de nommage, d'adressage et de gestion de ressources, l'ABone se dote d'une entité de régulation appelée ABOCC qui offre les services :

– d'enregistrement de développeurs d'EEs et d'AAs;
– d'enregistrement des nœuds actifs, des serveurs de code actif, des identificateurs d'EE ainsi que de la/les topologie(s) du réseau d'overlay;
– de support de gestion des droits d'accès ainsi que des mécanismes de supervision, détection et correction d'erreurs;

73. Active BOne Coordination Center.

– de création dynamique de topologies virtuelles à la demande de concepteurs et développeurs d'EEs et/ou d'AAs.

Un service supplémentaire proposé par l'ABOCC est de fournir une base centrale de disponibilité d'applications actives ayant fait leur preuve et pour lesquelles une demande d'utilisation forte est exprimée.

L'administration des nœuds et des environnements d'exécution stable est assurée par les développeurs de ces EEs via la plate-forme logicielle du ABOCC. En plus des serveurs de code, celle-ci comprend deux démons d'administration disponibles sur chaque nœud du réseau. Ces démons sont Anetd et QCMD. Le permier permet le déploiement, la configuration et le contrôle dynamique d'environnements d'exécution à distance. Il offre également un service de démultiplexage ANEP en fonction du type des paquets au niveau d'un nœud. Le second maintient et coordonne les contrôles d'accès. Il fonctionne en mode client/serveur; un serveur étant instancié sur le site du ABOCC, les clients étant chacun sur un nœud du réseau.

Actuellement le ABone fonctionne en topologie virtuelle fixe, bien que pour certains EEs, on puisse configurer des tables de routage au lancement de l'EE. L'interconnexion est assurée par des tunnels sur UDP. Cette architecture est en train d'évoluer avec l'arrivée d'implantation de multiplexage/démultiplexage ANEP au niveau IP et même liaison de données.

L'ensemble des informations d'accès, de souscription et d'installation de nœud actif est disponible à l'adresse suivante: `http://www.csl.sri.com/ancors/abone/` Un outil de visualisation de l'état et de la configuration actuelle est également disponible sur le site.

7.4.2 ANON

ANON[74] est un second réseau expérimental basé sur les paquets et le routage ANON présentés précedemment. Ce réseau est très peu déployé à ce jour et seule quelques expériences autour de M0[75] semblent encore s'y produire.

8 OS pour les réseaux actifs

Le succès des réseaux actifs dépend totalement de la disponibilité de systèmes d'exploitation adaptés aux exigences de performance et de gestion fine des ressources requises pour ce type d'approche. Dans cette section, nous résumons les travaux entrepris sur ce domaine.

8.1 Scout, Joust

Scout[76] [MOS 97] et son extension Java Joust [HAR 99] représentent un système candidat pour la mise en œuvre d'environnement d'exécution de réseaux actifs.

Scout est un système d'exploitation spécialisé pour les communications offrant une forte flexibilité de configuration et permettant une gestion fine des ressources (CPU, mémoire, bande passante). Les concepts de base de ce système d'expoitation sont le module, le graphe de modules et le chemin. Ces abstrations sont illustrées dans la figure 38.

Le module est le composant logiciel de base. Un module est une entité logicielle comportant un code de traitement opérant sur des flux de données. Le graphe est une composition de modules exprimant les dépendances entres modules. Finalement, le chemin (*path*) est une définition d'un circuit de transit de flux de données dans le système d'exploitation. Celui-ci encapsule la séquence de modules à appliquer à un flux et définit l'entité de séquencement des traitements sur ce flux.

L'ensemble de ces composants permettent la génération d'un système d'exploitation optimisé pour le traitement des flux générés. Une fois le système d'exploitation généré, celui-ci est opérationnel et permet la création ou la destruction dynamique de chemins en son sein. L'ajout de nouveaux types de chemins n'est pas possible à l'exécution.

L'ordonnancement est assuré à deux niveaux dans Scout. Le premier niveau est celui du système qui fournit un ordonnanceur pour l'ensemble des chemins. Le second niveau est un ordonnanceur dédié à chaque chemin dont la politique est dépendante des choix de l'application à la création du chemin. Cet ordonnanceur assure le partage des traitements entre les différents threads associés à un chemin. Afin de fournir le noyau le plus performant, l'environnement s'appuie sur des techniques d'optimisation de code telles que l'*outlining*, le clonage de fonctions

74. `http://ryyppy.ifi.unizh.ch/~anon`
75. `http://cuiwww.unige.ch/tios/msgr/home.htmld`
76. `http://www.cs.arizona.edu/liquid/`

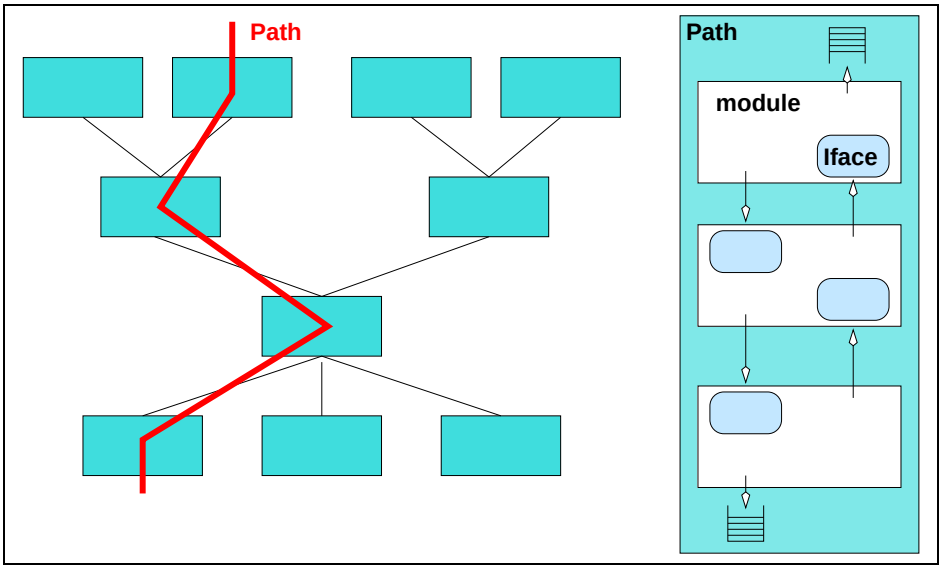

FIG. 38 – *Les abstractions de Scout*

et l'optimisation d'appels dans un chemin. Scout [77] est disponible sur Internet. Il permet la génération d'OS pour processeurs Intel Pentium et DEC.

Joust [HAR 99] est une extension Java construite sur l'environnement Scout. Joust comprend :

- le système d'exploitation Scout;
- un compilateur JIT;
- une machine virtuelle optimisée incluant une API étendue intégrant les concepts de module et de chemin propres à Scout.

Cette interface fournit des fonctions de création et d'extension de chemin. L'ordonnancement des traitements dans Joust repose sur celui de Scout. La figure 39 comprend le graphe des modules de Joust ainsi que l'architecture des composants.

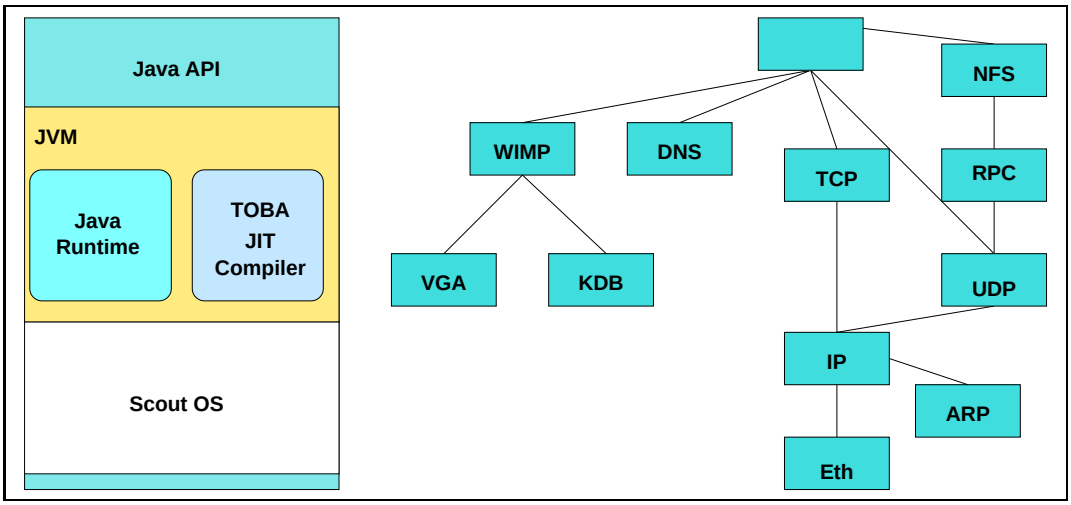

FIG. 39 – *Les composants de Joust*

Afin d'obtenir des performances adaptées aux flux multimédia dans l'OS, la machine virtuelle comporte également de nombreuses optimisations par rapport à celle de SUN. Ces optimisations portent notamment sur la gestion des threads et des exceptions.

Joust apparaît aujourd'hui comme l'un des systèmes d'exploitation le mieux adapté à un nœud actif exécutant un environnement d'exécution en Java. Cependant cette architecture, bien que conçue pour les communications,

77. `http://www.cs.arizona.edu/scout/`

ne dispose pas de gestion de ressources propres à l'application. Ceci représente un handicap par rapport aux besoins exprimés dans l'architecture de nœud en cours de standardisation dans la communauté des réseaux actifs.

8.2 Exokernel

Exokernel[ENG 98] est un système d'exploitation dont l'objectif est d'offrir une gestion de ressources fine, sûre et ouverte aux applications. Le concept principal de ce système est de sortir toutes les fonctions de gestion des ressources du noyau, de les mettre dans des librairies utilisateurs et de maintenir dans le noyau les fonctions de protection de ces ressources à un niveau très fin (ex. bloc disque). Cette approche rompt avec les systèmes d'exploitation monolytiques qui offrent une vision très abstraite des ressources et implantent un mécanisme unique de gestion des ressources pour toutes les applications.

La figure 40 illustre l'architecture d'*Excokernel*. Les applications utilisateurs peuvent définir leurs propres mécanismes de gestion des ressources en recodant les librairies systèmes et en les utilisant pour leurs besoins propres. *Exokernel* fournit des librairies par défaut et le noyau offre une vision très primaire des ressources du hardware. Les librairies systèmes définies par les applications se chargent de la gestion de ces ressources. La protection des ressources (ex. s'assurer qu'une application ne contrôle que ses ressources propres) est conduite par le noyau. Un mécanisme particulier permet aux applications de définir des filtres d'abonnement aux paquets de données provenant des interfaces de communication. Ce mécanisme est rendu nécessaire dans *Exokernel* afin que le noyau puisse assurer la protection des ressources de communication entre applications.

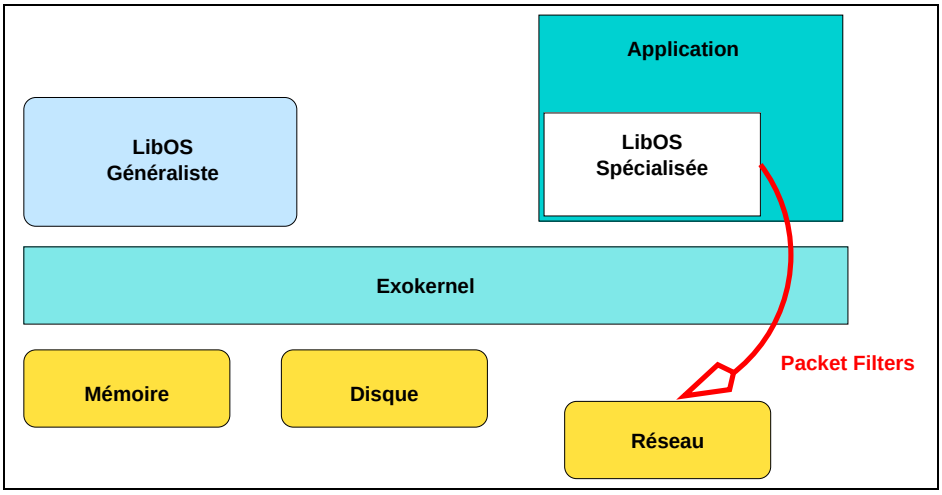

FIG. 40 – *Le noyau Exokernel*

Contrairement à Scout qui est un système d'exploitation dédié pour les communications et qui offre des abstractions de haut niveau aux applications, *Exokernel* est un système généraliste qui n'offre aucune abstraction des ressources. C'est aux applications, via des librairies de système spécialisées, de se construire leurs propres abstractions.

Les principaux intérêts de ce système dans le cadre des réseaux actifs sont bien sûr sa flexibilité et les performances atteignables via l'utilisation de librairies système adaptées à un traitement donné (en général un *speedup* de facteur 5 par rapport à un OS UNIX standard sur la même architecture). Cependant, il n'est pas acquis que la majorité des concepteurs de réseaux actifs se donne aujourd'hui les moyens humains et techniques nécessaires pour réaliser de telles librairies. Des travaux sont en cours dans certains projets sur l'utilisation d'exokernel pour la création d'un noeud actif, mais cet OS n'est pas le seul candidat. *Exokernel* existe pour la famille des architectures x86 et MIPS.

Il existe plusieurs projets qui construisent un OS de nœud compatible avec l'architecture de nœud standard définies dans les sections précédentes. Parmis ces projets, le projet AMP[78] développé aux TIS Labs at Network Associates a réalisé un OS de nœud conforme aux recommandations de la communauté des réseaux actifs étendu par des mécanismes de sécurité pour le traitement des autorisations et de l'authentification lors de la création et de l'exploitation de flux sur un nœud. Ces mécanismes sont couplés à des extensions pour le démultiplexage, l'ordonnancement de paquets à émettre et le contrôle de respect de QoS par flux.

78. `http://www.nai.com/nai_labs/asp_set/infrastructure/amp.asp`

8.2.1 ANN

ANN[79] (*Active Network Node*) [DES 98b] est une définition d'un nœud actif à haute performance pour des réseaux gigabits. Cette architecture comprend :

- une architecture hardware connectée à un réseau ATM ;
- un système d'exploitation de nœud performant et adapté à l'architecture physique proposée ;
- un environnement d'exécution spécifique nécessaire à l'architecture à la gestion du téléchargement de code actif sur un nœud.

Trois postulats ont guidé la conception de cette architecture :

- les traitements actifs se font sur des flux de paquets et non de manière isolée paquet par paquet ;
- les utilisateurs finaux ne programment pas le réseau. Les services actifs sont fournis par le fournisseur de services ;
- une approche machine virtuelle ne peut être retenue pour des réseaux à haut débit en raison de la fenêtre temporelle extrêmenent faible durant laquelle un processeur peut effectuer un traitement sur un paquet (380 nano secondes pour un flux de 10 GB avec des paquets de 512 octets). Les codes doivent être fournis sous forme de code natif.

L'architecture physique réalisée dans ANN repose sur une carte de traitement par port (155 Mb - 1.2 Go). Cette carte comprend un contrôleur ATM, une CPU, 4 GO de mémoire et un FPGA[80] pour le support efficace de fonctions avancées. Plusieurs cartes peuvent être cascadées sur un port si des performances de traitement supérieures sont nécessaires. Dans ce cas, un algorithme de partage de charge répartit les traitements entre les cartes. Les cartes sont reliées entre elles par un fond de panier ATM.

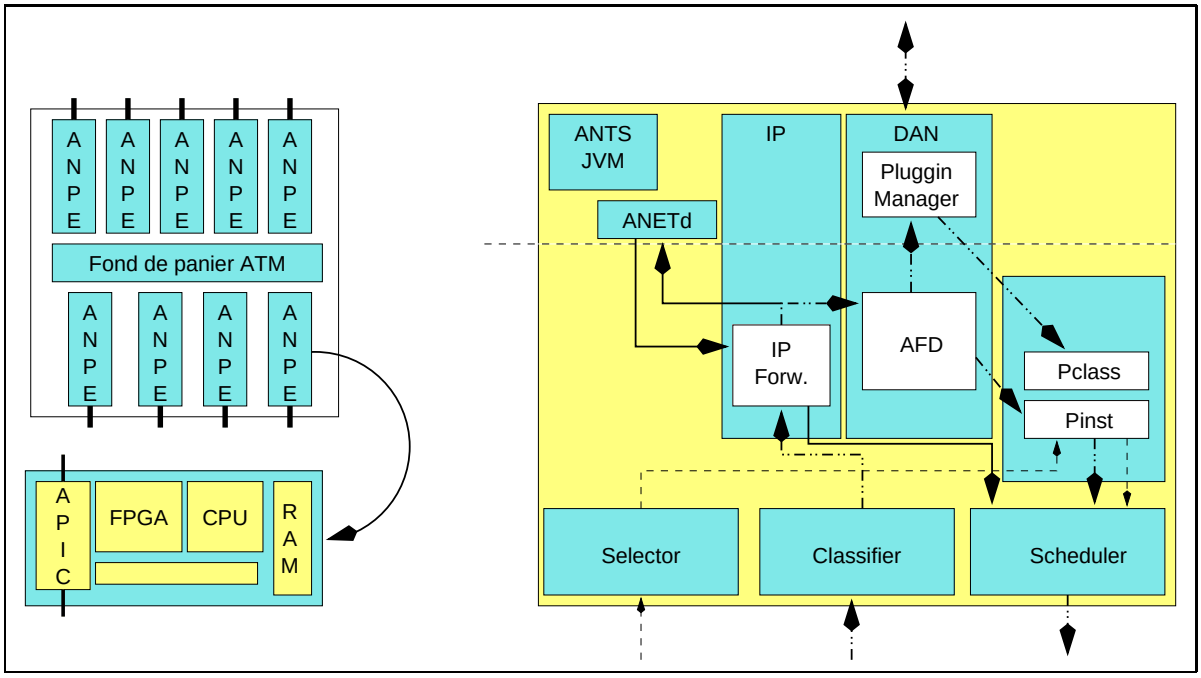

FIG. 41 – *L'architecture de noeud ANN*

Le modèle de flux de paquets actifs repose sur une classification de ces derniers identique à celle retenue dans les *Router Pluggins* (adresses source et destination, port source et destination, identificateur de protocole) qui forment un index de flux utilisé dans l'encapsulation SAPF des paquets actifs.

Le système d'exploitation est une extension de NetBSD intégrant les capacités de chargement de modules actifs définis dans les *Router pluggins*. La répartition des ressources se fait équitablement sur les différents flux à l'aide d'un contrôleur de ressources.

79. `http://www.ail.wustl.edu/arl/projects/ann/ann.html`
80. Field Programmable Gate Array

L'insertion de modules actifs se fait de manière identique aux *Router Pluggins* étendus avec un mécanisme de chargement à distance via un serveur de code et des caches permettant un téléchargement à la demande très efficace. Cette fonction de téléchargement est assurée par un EE spécifique appelé DAN [DES 98c]. Il dispose dans le noyau d'un module de dispatching des paquets actifs vers la chaîne de traitement associée (voir figure 41). Si une fonction n'est pas disponible sur le nœud, ce module fait appel à la fonction de chargement en lui passant en paramètre l'identificateur du *pluggin* manquant. Cette fonction vérifie les droits associés et consulte sa base locale. Si le *pluggin* est dans la base, il est chargé dans l'unité de contrôle de pluggin au sein du noyau qui va pouvoir l'instancier. Si le code n'est pas disponible, la fonction de chargement s'adresse à un serveur de code distant pour récupérer le package (code objet pour l'architecture cliente + signature digitale du développeur + informations d'authentification du serveur + informations d'installation et de configuration du paquetage).

L'implantation de cette architecture, principalement la partie hardware, est en cours de réalisation à l'universté de Washington et le serveur de code ainsi que l'environnement DAN est développé conjointement à l'ETHZ et à l'universté de Washington.

8.3 Autres OS actifs

Il existe de nnombreux autres OS en cours de développement pour les réseaux actifs. Ceux-ci sont simplement énumérés ici :

- AMP qui est construit sur *Exokernel* pour faire une architecture de nœud actif;
- JANOS[81]
- ODYSSEY (Georgia Tech) pour une réalisation d'un système d'exploitation pour nœud actif sur SOLARIS.
- x-kernel
- Horus
- les OS standards
- Nemesys
- Bowman

9 L'industrialisation

Plusieurs startups ont été créées pour réaliser des solutions industrielles autour des réseaux programmables. Ces entreprises sont issues des différents projets de recherche du domaine. Autour de ces startups, de nombreux équipementiers et opérateurs participent activement à la définition de ces architectures programmables dans le but de pouvoir les mettre en oeuvre dans leurs produits d'une part, les déployer sur leurs réseaux d'autre part, ceci dans des délais raisonnables (2-5 ans).

CPlane[82] est l'exemple type de la startup pour des architectures programmables. Issue de l'université de Cambridge, *CPlane* vise à industrialiser l'approche Tempest et plus particulièrement le concept de *Switchlet* pour la virtualisation de commutateurs ATM et d'étendre ce concept à d'autres types de réseaux, notamment IP.

X.bind[83] est issue de l'université de Columbia. Cette startup vise également à offrir une architecture logicielle pour la création, le déploiement et l'exploitation de services sur une architecture programmable. Les interfaces définies par Xbind, notamment pour ATM, sont en cours de standardisation dans le projet P1520 de l'IEEE.

IKV++[84] est une entreprise allemande qui commercialise la plate-forme à agents mobiles *Grasshoper* utilisée dans différents projets européens autour du réseau intelligent comme plate-forme de déploiement de composants actifs.

Autour des réseaux programmables, un forum appelé Multiservice Switching Forum[85] regroupant les principaux opérateurs, fournisseurs de services et fournisseurs d'équipements ainsi que les startups du domaine a été mis en place en novembre 1998. Ce forum vise à définir d'une part une architecture capable de dissocier clairement le plan de contrôle du plan de transport dans des équipements de commutation ATM afin de permettre à ces équipements d'intégrer rapidemment de nouvelles fonctions de contrôle. D'autre part, le forum vise à promouvoir des interfaces de programmation au sein même de ces commutateurs permettant la division logique

81. `http://www.cs.utah.edu/flux/java/index.html\#JANOS`
82. `http://www.cplane.com/`
83. `http://www.xbind.com`
84. `http://www.ikv.de/`
85. `http://www.msforum.org`

des ressources (partitionnement, virtualisation) et, au travers de ces interfaces, de permettre la possibilité de déployer des réseaux virtuels multiples ayant chacun son propre plan de contrôle.

Comme nous l'avons présenté dans le contexte des réseaux programmables, le projet de standard P1520 de l'IEEE est également très actif. Tout comme dans le MSForum, de nombreux opérateurs, équipementiers et fournisseurs de services sont impliqués dans ce projet au sein duquel des permières interfaces sont en cours de standardisation.

Dans le domaine des réseaux actifs, les perspectives d'industrialisation sont plus frileuses et aucune startup ne semble aujourd'hui se focaliser sur de telles architectures.

10 Conclusion

Dans ce rapport, nous avons présenté l'état de l'art de la recherche, de la standardisation et de l'industrialisation dans les réseaux programmables et les réseaux actifs. Nous avons vu que le nombre de projets et l'effort associé est très important et que de nombreux résultats extrêmement puissants et prometteurs ont déjà émergé.

La recherche dans ce domaine bat aujourd'hui son plein et les efforts soutenus permettrons d'aboutir à des architectures qui feront certainement partie des architectures de réseaux de demain. Sur la base des travaux énumérés dans ce rapport, nous proposons une vision de l'avenir pour les deux principales familles de réseaux programmables présentées. Cette vision est donnée ci-dessous

10.1 Les réseaux programmables : un avenir radieux

Si les différents intervenants arrivent assez rapidement à un consensus sur les spécifications précises des différentes interfaces, des premières études de terrain à grande échelle vont pouvoir être réalisées asez rapidemment (d'ici 1 à 2 ans) et permettre le déploiement industriel d'ici 24 à 36 mois.

10.2 Les réseaux actifs : un avenir prometteur

Bien que de nombreux projets de recherche se soient lancés sur cette voie et que de nombreux résultats intéressants ont pu être publiés, la communauté reste divisée sur l'intérêt véritable de cette approche. En effet, d'une part les réseaux actifs apportent une flexibilité jamais atteinte dans les réseaux, mais, revers de la médaille, ils apportent énormément de problèmes non résolus à ce jour ou dont les solutions sont trop coûteuses pour le moment.

Il est évident que l'apport potentiel des réseaux actifs surpasse largement les problèmes, notamment de sécurité et de performance qu'ils posent aujourd'hui mais qui, grâce à la recherche dans ces domaines, seront résolus demain.

Les concepteurs d'ANTS ont une vision légèrement différente. En effet, ceux-ci prédisent que dans l'avenir peu de services actifs seront disponibles[86] et que ce type d'architecture permettra principalement de déployer des protocoles d'expérimentation [WET 99a]. Nous partageons partiellement ce point de vue. L'utilisation de ces techniques pour le déploiement de protocoles expérimentaux est un progrès indéniable. Cependant en ce qui concerne la limitation des services actifs offerts à l'avenir, nous pensons que certains domaines tels que la supervision peuvent profiter pleinement de ce type d'architecture et nous pensons que les composants actifs sont amenés à se développer fortement.

Références

[ADA 98] Adam C., Huard J.-F., Lazar A., , Lim K.-S., Nandikesan M.Shim E., Proposal for standardization of ATM Binding Interface Base 2.1, October 1998, IEEE/WG P1520/TS/ATM-005 Working Document.

[ADA 99a] Adam C., Lazar A.Nandikesan M., The qGSMP Protocol, June 1999, draft-adam-gsmp-qgsmp-00.txt.

[ADA 99b] Adam C., Huard J.-F., Lazar A., Nandikesan M.Shim E., Proposal for modification of the ATM Switch Service Interface, July 1999, IEEE/WG P1520/TS/ATM-020 Working Document.

[ADA 99c] Adam C., Lazar A.Nandikesan M., ATM Switch Resource Abstractions, March 1999, IEEE/WG P1520/TS/ATM-017 Working Document.

86. L'argument avancé étant que tout service actif peut potentiellement être implanté de façon native en non-actif.

[ADA 99d] ADAM C., LAZAR A.NANDIKESAN M., Switch abstractions for designing open interfaces, March 1999, IEEE/WG P1520/TS/ATM-016 Working Document.

[ALE 97a] ALEXANDER D., BRADEN B., GUNTER C., JACKSON A., KEROMYTIS A., MINDEN G.WETHERALL D., Active Network Encapsulation Protocol (ANEP), July 1997.

[ALE 97b] ALEXANDER D., SHAW M., NETTLES S.SMITH J., Active Bridging, *Proc. ACM SIGCOMM'97*, Cannes, France, September 1997.

[ALE 97c] ALEXANDER S., A Generalized Computing Model of Active Networks, February 1997.

[ALE 98a] ALEXANDER D., ALIEN: A Generalized Computing Model of Active Networks, PhD thesis, University of Pennsylvania, December 1998.

[ALE 98b] ALEXANDER D., ARBAUGH W., HICKS P., KAKKAR P., KEROMYTIS A., MOORE J., C.A. G., S.C. N.SMITH J., The SwitchWare Active Network Architecture, *IEEE Network Special Issue on Active and Controllable Networks*, 12, 3, 1998, 29-36.

[ALE 98c] ALEXANDER D., ARBAUGH W., KEROMYTIS A.SMITH J., A Secure Active Network Environment Architecture: Realization in SwitchWare, *IEEE Network*, 12, 3, 1998, 37-45.

[ALE 99a] ALEXANDER D., ARBAUGH W., KEROMYTIS A.SMITH J., Security in Active Networks, *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, Springer Verlag, LNCS State of the Art Series, 1999.

[ALE 99b] ALEXANDER D.SMITH J., The Architecture of ALIEN, COVACI S., , *Active Networks: Proc. First International Working Conference, IWAN'99*, Berlin, Germany, June 1999, Springer Verlag, LNCS 1653, 1-12.

[AMI 98] AMIR E., MCCANNE S.KATZ R., An Active Service Framework and its Application to Real-time Multimedia Transcoding, *Proc. SIGCOMM'98*, Vancouver, B.C., 1998, ACM.

[ANG 98] ANGIN O., CAMPBELL A., KOUNAVIS M.LIAO R., The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking, *IEEE Personal Communications Mag., Special Issue on Adapting to Network and Client Variability*, 5, 4, 1998, 32-44.

[BER 99] BERSON S., BRADEN B.RICCIULLI L., Introduction to the ABone, Research report, November 1999.

[BHA 96] BHATTACHARJEE S., CALVERT K.ZEGURA E., Implementation of an Active Networking Architecture, *Proc. Gigabit Switch Technology Workshop*, Washington University, St. Louis, July 1996.

[BHA 97a] BHATTACHARJEE S., CALVERT K.ZEGURA E., Active Networking and the End-to-End Argument, *ICNP*, Atlanta, GA, October 1997.

[BHA 97b] BHATTACHARJEE S., CALVERT K.ZEGURA E., An Architecture for Active Networking, *High Performance Networking (HPN'97)*, White Plains, NY, April 1997.

[BHA 98a] BHATTACHARJEE S., CALVERT K.E.W. Z., Reasonning About Active Networks, *Proc. ICNP'98*, Austin, TX., October 1998.

[BHA 98b] BHATTACHARJEE S., CALVERT K.ZEGURA E., Self-Organizing Wide-Area Network Caches, *Proc. IEEE INFOCOM'98*, 1998.

[BHA 99] BHATTACHARJEE S., CALVERT K.ZEGURA E., Network Support for Multicast Video, *Submitted to IWQoS'99*, 1999, see also Tech. rep. GIT-CC-98-16.

[BIS 99] BISWAS J., HUARD J.-F., LAZAR A., LIM K., MAHJOUB S., PAU L.-F., SUZUKI M., TORSTENSSON S., WEIGUO W.WEINSTEIN S., Application Programming Interfaces for Networks, January 1999, IEEE P1520 WG Draft White Paper.

[BRA 97] BRADEN R., ZHANG L., BERSON S., HERZOG S.JAMIN S., Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205, , September 1997.

[BRA 98] BRANCH A., EFFELSBERG W., TSCHUDIN C.TURAU V., Multicasting Multimedia Streams with Active Networks, *Proc. LCN'98*, Boston, MA., October 1998.

[BRA 99a] BRADEN B., CERPA A., FABER T., LINDELL B., PHILLIPS G.KANN J., ASP EE: An Active Execution Environment for Network Control Protocols, Technical report, December 1999, USC/ISI.

[BRA 99b] BRADEN B.RICCIULLI L., A Plan for a Scalable ABone - A Modest Proposal, January 1999.

[CAL 98] CALVERT K., BHATTACHARJEE S., ZEGURA E.STERBENZ J., Directions in Active Networks, *IEEE Communications Magazine*, , 1998.

[CAM 99a] CAMPBELL A., DE MEER H., KOUNAVIS M., MIKI K., VICENTE J.VILLELA D., The Genesis Kernel: A Virtual Network Operating System for Spawning Network Architectures, *2nd Int'l Conf. on Open Architectures and Network Programming*, N.Y., May 1999.

[CAM 99b] CAMPBELL A., KOUNAVIS M., VILLELA D., VICENTE J., DE MEER H., MIKI K.KALAICHELVAN K., Spawning Networks, *IEEE, Network*, , 1999, 16-28.

[CAM 99c] Campbell A., De Meer H., Kounavis M., Miki K., Vicente J.Villeta D., A Survey of Programmable Networks, *ACM Computer Communications Review*, , 1999.

[CAM 99d] Campbell A., Kounavis M.Liao R., Programmable Mobile Networks, *Computer Networks and ISDN Systems, Computer Networks*, 31, 1999.

[CAR 99] Cardoe R., Scott A.Shepherd W., LARA : A prototype System for Supporting High Performance Active Networking, Covaci S., , *Active Networks: Proc. First International Working Conference, IWAN'99*, Berlin, Germany, June 1999, Springer Verlag, LNCS 1653, 117-131.

[CHA 98] Chandra P., Fisher A., Kosak C., Eugene Ng T., Steenkiste P., Takahashi E.Zhang H., Darwin: Resource Management for Value-Added Customizable Network Service, *Sixth IEEE International-Conference on Network Protocols (ICNP'98)*, Austin, TX, 1998.

[CHE 98] Chen T., Jackson A., Bhattacharjee S., Calvert K., Zegura E., Partridge C., Strayer T., Schwartz B., Jackson A., Reed D., Saltzer J.Clark D., Commentaries on "Active Networking and End-To-End Arguments", *IEEE Network*, , 1998, Special Issue on Active and Controllable Networks.

[DAS 98a] Da Silva S., Programming in the NetScript Toolkit, September 1998, http://www.cs.columbia.edu/ dasilva/pubs/netscript-0.10/doc/tutorial.html.

[DAS 98b] Da Silva S., Florissi D.Yemini Y., NetScript: A Language-Based Approach to Active Networks, *Technical Report, Computer Science Dept., Columbia University*, , 1998.

[DAV 98] David C. Feldmeier D., McAuley A., Smith J., Bakin D., Marcus W.Raleigh T., Protocol Boosters, *IEEE Journal of Selected Areas in Communications*, , 1998.

[DEL 99] Delgrossi L., Di Fatta G., Ferrari D.Lo Re G., Interference and Communication among Active Network Applications, Covaci S., , *Active Netwoks: First International Working Conference, (IWAN"99)*, 1653 *LNCS*, Berlin, Germany, July 1999, Springer Verlag, 97–108.

[DEN 98] Denazis S., Miki K., Vicente J.Campbell A., Designing IP Router L-Interfaces, March 1998, IEEE/WG P1520/TS/IP-005 Third Draft.

[DEN 99] Denazis S., Miki K., Vicente J.Campbell A., Designing Interfaces for Open Programmable Routers, Covaci S., , *Active Networks: Proc. First International Working Conference, IWAN'99*, Berlin, Germany, June 1999, Springer Verlag, LNCS 1653, 13-24.

[DES 98a] Descarper D., Dittia Z., Parulkar G.Plattner B., Router Pluggins A Software Architecture for Next Generation Routers, *Proc. ACM SIGCOMM'98*, Vancouver, BC, 1998.

[DES 98b] Descarper D., Parulkar G., Choi S., DeHart J., Wolf T.Plattner B., A Scalable, High Performance Active Network Node, October 1998.

[DES 98c] Descarper D.Plattner B., DAN : Distributed Code Caching for Active Networks, *Proc. INFO-COM'98*, 1998.

[ENG 98] Engler D., The Exokernel Operating System Architecture, PhD thesis, Massachusetts Institute of Technology, October 1998.

[EST 97] Estrin D., Farinacci D., Helmy A., Thaler D., Deering S., Handley M., Jacobson V., Liu C., Sharma P.Wei L., RFC 2117: Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification, 1997.

[FRY 99] Fry M.Ghosh A., Application level active networking, *Computer Networks*, 31, 1999, 655-667.

[GRO 98] Group A. N. C. S. W., Composable Services for Active Networks, September 1998.

[HAR 96] Hartman J.Manber U., Liquid Software : A New Paradigm for Networked Systems, Technical Report TR-96-11, 1996, University of Arizona, Dept. of Computer Science.

[HAR 99] Hartman J., Bigot P., Bridges P., Montz B., Piltz R., Spatscheck O., Proebsting T.Peterson L., Joust: A Platform for Liquid Software, *IEEE Computer*, , 1999.

[HIC 97a] Hicks M., PLAN Service Programmer's Guide, November 1997.

[HIC 97b] Hicks M., The PLAN Active Router, November 1997.

[HIC 98] Hicks M., Kakkar P., Moore J., Gunter C.Nettles S., PLAN: A Packet Language for Active Networks, *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages*, ACM, 1998, 86-93, Available at `www.cis.upenn.edu/~switchware/papers/plan.ps`.

[HIC 99a] Hicks M.Keromytis A., A Secure PLAN, Covaci S., , *Active Networks: Proc. First International Working Conference, IWAN'99*, Berlin, Germany, June 1999, Springer Verlag, LNCS 1653, 307-314.

[HIC 99b] Hicks M., Moore J., Alexander D., Gunter C.Nettles S., PLANet: an Active Internetwork, *Proc. IEEE INFOCOM '99*, New York, March 1999.

[HUI 94] Huitema C., *Le routage dans l'internet*, Eyrolles, 1994.

[HYM 91] HYMAN J., A.A. L.PACIFICI G., Real-Time Scheduling with Quality of Service Constraints, *IEEE Journal of Selected Areas in Communications*, 9, 1991, 1052-1063.

[JEF 97] JEFFREY A.WAKEMAN I., A Survey of Semantic Techniques for Active Networks, November 1997.

[KAK 97] KAKKAR P., MOORE J.HICKS M., The PLAN Tutorial, November 1997.

[KAK 99a] KAKKAR P., The specification of PLAN, July 1999.

[KAK 99b] KAKKAR P.GUNTER C., Experience with Specifying the PLAN Network Programming Language, *Proc. HOOTS'99*, June 1999.

[KOU 99] KOUBBA H., Les communications multipoints dans les réseaux actifs, Mémoire de D.E.A. Informatique, Université Henri Poincaré, Juin 1999.

[LAZ 97] LAZAR L., Programming Telecommunication Networks, *IEEE Network Magazine*, , 1997, 8-18.

[LEE 99] LEE D., JONES M., MIDKIFF S.ATHANAS P., Towards Active Hardware, COVACI S., , *Active Networks: Proc. First International Working Conference, IWAN'99*, Berlin, Germany, June 1999, Springer Verlag, LNCS 1653, 180-187.

[LEG 98a] LEGEDZA U.GUTTAG J., Using Network-level Support to Improve Cache Routing, *3rd International WWW Caching Workshop*, Manchester, England, June 1998.

[LEG 98b] LEGEDZA U., WETHERALL D.GUNTER J., Improving The Performance of Distributed Applications Using Active Networks, *Proc. IEEE INFOCOM'98*, San Francisco, CA., April 1998.

[LEH 98] LEHMAN L.-W. H., GARLAND J.TENNENHOUSE D., Active Reliable Multicast, *Proc. IEEE INFOCOM'98*, 1998.

[LIN 99a] LIN P., DENAZIS S., MIKI K., VICENTE J., SUZUKI M., REDLICH J.-P., CUERVO F., BISWAS J., WANG W., MIKI K.GUTIEREZ J., Programming Interfaces for IP Networks, A White Paper, June 1999, IEEE/WG P1520/TS/IP-001 Draft.

[LIN 99b] LIN P., DENAZIS S., MIKI K., VICENTE J., SUZUKI M., REDLICH J.-P., CUERVO F., BISWAS J., WANG W., MIKI K.GUTIEREZ J., Programming Interfaces for IP Routers and Switches, an Architectural Framework Document, June 1999, IEEE/WG P1520/TS/IP-003 Draft.

[MAR 98] MARCUS W., HADZIC I., MCAULEY A.SMITH J., Protocol Boosters: Applying Programmability to Network Infrastructures, *IEEE Communications Magazine*, , 1998.

[MAR 99] MARSHALL I., FRY M., VELASCO L.GHOSH A., Active Information Networks and XML, COVACI S., , *Active Networks: Proc. First International Working Conference, IWAN'99*, Berlin, Germany, June 1999, Springer Verlag, LNCS 1653, 60-72.

[MEN 99] MENAGE P., RCANE: A Resource Controlled Framework for Active Network Services, COVACI S., , *Active Networks: Proc. First International Working Conference, IWAN'99*, Berlin, Germany, June 1999, Springer Verlag, LNCS 1653, 25-36.

[MER 98] VAN DER MERWE J., ROONEY S., LESLIE I.CROSBY S., The Tempest – A Practical Framework for Network Programmability, *IEEE Network*, , 1998.

[MOO 97a] MOORE J.HICKS M., A Service Layer Routing Protocol for PLAN, November 1997.

[MOO 97b] MOORE J.HICKS M., PLAN Programmer's Guide, November 1997.

[MOS 97] MOSBERGER D., Scout: A Path-based Operating System, PhD thesis, Department of Computer Science, University of Arizona, 1997.

[MUR 97] MURPHY D., Building an Active Node on the Internet, May 1997, Master Thesis, Massachusetts Institute of Technology.

[PAR 99] PARLAY G., Parlay API Business Benefits White Paper (issue 1.0), June 1999.

[PHI 99a] PHILLIPS G., A Programmer's Tutorial for the ASP Execution Environment, Technical report, December 1999, USC/ISI.

[PHI 99b] PHILLIPS G., BRADEN B., KANN J.B. L., The ASP Execution Environment's Protocol Programming Interface, Technical report, December 1999, USC/ISI.

[POW 96] POWELL D., Group Communication, *Communications of the ACM*, 39, 4, 1996, 50–97, (special section Group Communication).

[PSO 99] PSOUNIS K., Active Networks: Applications, Security, Safety and Architectures, *IEEE Communications Survey*, , 1999, 2-16.

[RAN 99] RANGANATH M., AGRAWAL R., PILLAI R., WANG W.KRISHNA M., Proposal for standardization U-Interface Objects for Wireless ATM, March 1999, IEEE/WG P1520/TS/ATM-012 Working Document.

[RFC 96] RFC1987, General Switch Management Protocol 1.1, August 1996.

[RIC 99] Ricciuli L., Lincoln P.Kakkar P., TCP SYN Flooding Defense, *Proc. CNDS'99*, 1999.

[ROO 98] Rooney S., van der Merwe J., Crosby S.Leslie I., The Tempest: A Framework for Safe, Resource-Assured, Programmable Networks, *IEEE Communications*, , 1998.

[SAL 84] Saltzer J., Reed D.Clark D., End-to-end arguments in system design, *ACM Transactions on Computer Systems*, 2, 4, 1984, 277–288.

[SAV 99] Savage S., Anderson T., Aggarwal A., Becker D., Cardwell N., Collins A., Hoffman E., Snell J., Vahdat A., Voelker G.Zahorjan J., Detour: Informed Internet Routing and Transport, *IEEE Micro*, 19, 1, 1999.

[SCH 99] Schwartz B., Zhou W., Jackson A., Strayer W., Rockwell D.Partridge C., Smart Packets for Active Networks, *Proc. OpenArch '99*, `http://www.net-tech.bbn.com/smtpkts/smtpkts-index.html`, March 1999.

[SCO 98] Scott M., Nguyen K., Rosenstein A., Zhang L., Floyd S.Jacobson V., Adaptive Web Caching: Towards a New Caching Architecture, *3rd International WWW Caching Workshop*, Manchester, England, June 1998.

[SEC 98] SecurityWG, Security Architecture for Active Networks, July 1998.

[SMI 98] Smith J., Farber D., Gunter C., Nettles S., Segal M., Sincoskie W., Feldmeier D.Alexander S., SwitchWare: Towards a 21st Century Network Infrastructure. , 1998.

[SMI 99] Smith J., Calvert K., Murphy S., Orman H.Peterson L., Activating Networks: A Progress Report, *IEEE Computer*, , 1999, 32-41.

[TAK 99] Takahashi E., Steenkiste P., Gao J.Fisher A., A Programming Interface for Network Resource Management, *Second IEEE Conference on Open Architectures and Network Programming (OPENARCH'99)*, New York, march 1999.

[TEN 96a] Tennenhouse D., Garland S.Kaashoek M., From Internet to ActiveNet, January 1996.

[TEN 96b] Tennenhouse D.Wetherall D., Towards an Active Network Architecture, *Computer Communication Review*, 26, 2, 1996.

[TEN 97] Tennenhouse D., Smith J., Sincoskie W., Wetherall D.Minden G., A Survey of Active Network Research, *IEEE Communications Magazine*, 35, 1, 1997, 80-86.

[THI 98a] Thibault S., Consel C.Muller G., Safe and Efficient Active Network Programming, *Proc. IEEE SRDS'98*, October 1998.

[THI 98b] Thibault S., Consel C.Muller G., Safe and Efficient Active Network Programming, Rapport interne RI-1170, Février 1998, IRISA.

[THI 99] Thibault S., Marant J.Muller G., Adapting Distributed Applications Using Extensible Networks, *Proc. IEEE ICDCS'99*, Austin, TX., June 1999.

[VAN 97] Van V., A Defense Against Address Spoofing Using Active Networks, PhD thesis, Massachusetts Institute of Technology, May 1997.

[VAN 99] Vanet G.Kiriha Y., A Self-Configuring Data Caching Architecture Based on Active Networking Techniques, Covaci S., , *Active Netwoks. First International Working Conference (IWAN"99)*, 1653 *LNCS*, Berlin, Germany, July 1999, Springer, 85–96.

[WAK 98] Wakeman I., Jeffrey A., Graves R.Owen T., Designing a Programming Language for Active Networks, June 1998.

[WET 96] Wetherall D.Tennenhouse D., The Active IP Option, *Proc. ACM SIGOPS European Workshop '96*, Connemara, Ireland, September 1996.

[WET 97] Wetherall D., Developing Network Protocols with the ANTS Toolkit, August 1997.

[WET 98a] Wetherall D., Legedza U.Guttag J., Introducing New Internet Services: Why and How, *IEEE Network. Special Issue on Active and Programmable Networks*, , 1998.

[WET 98b] Wetherall D., Guttag J.Tennenhouse D., ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols,, *Proc. IEEE OpenArch'98*, 1998.

[WET 99a] Wetherall D., Active Networks Vision and Reality: Lessons from a Capsule-based System, *17th ACM Symposium on Operating Systems Principles (SOPS'99)*, Kiawah Island, SC, december 1999, ACM, 64-79.

[WET 99b] Wetherall D., Guttag J.Tennenhouse D., ANTS: Network Services Without the Red Tape, *IEEE Computer*, , 1999.

[WET 99c] Wetherall D., Service Introduction in an Active Network, PhD thesis, Massachusetts Institute of Technology, February 1999.

[WG. 99] WG. A. N., Architecural Framework for Active Networks Version 1.0, July 1999.

[WIJ 97] WIJATA Y., Ressource Management in Active Networks, May 1997, Technical Report, U. Kansas.

[WIL 99] WILLIAMSON B.FARREL C., Independent active pogram representation using ASN.1, *ACM Computer Communications Review*, , 1999.

[WIT 98a] WITTMANN R., KRASNODEMBSKI K.ZITTERBART M., Heterogeneous Multicasting based on RSVP and QoS Filters, *International Symposium on Broadband Networks (SYBEN'98)*, Zürich, Switzerland, May 1998.

[WIT 98b] WITTMANN R.ZITTERBART M., Active multicasting for heterogeneous groups, *4th International Conference on Broadband Communications '98*, Stuttgart, Germany, April 1998, IFIP.

[WIT 98c] WITTMANN R.ZITTERBART M., AMnet : Active Multicasting Network, 1998.

[WIT 98d] WITTMANN R.ZITTERBART M., AMnet: Active Multicasting Network, *International Conference on Communications (ICC'98)*, Atlanta, GA, USA, June 1998.

[YEM 96] YEMINI Y.DA SILVA S., Towards Programmable Networks, *Proc. IFIP IEEE Int Workshop on Distributed Systems, Operations, and Managemement*, L'aquila, Italy, 1996.

[YEM 99] YEMINI Y., DA SILVA S., FLORISSI D.HUANG H., The Network Flow Language : A Mark-Based Approach to Active Networks, Technical report, July 1999, Columbia University, Dept. of Computer Science.

A GSMP

GSMP (*General Switch Management Protocol* [RFC 96]) est un protocole gérnérique permettant à un contrôleur d'activer des commandes de contrôle à distance sur un ou plusieurs switchs ATM. La version initiale, proposée par la société Ipsilon date de 1996. Les commandes offertes au travers de GSMP sont :

- des commandes de contrôle de connexion point-à-point ou multi-points (établissement, terminaison, modification, vérification) ;
- des commandes de contrôle de ports (mise en service, mise hors-service, mise en loopback, reset) ;
- des commandes de collecte d'informations statistiques sur les ports et les circuits virtuels (principalement des compteurs de traffic et d'erreurs) ;
- des commandes de configuration qui permettent au contrôleur de disposer d'informations sur la capacité d'un switch. Trois requêtes ont été définies. La première permet d'obtenir des informations administratives sur un switch (type, identificateur). La seconde permet de demander la configuration d'un port spécifique (numéro de session de port [87], l'espace de nommage pour les VCI et VPI au sein du port, sa bande passante, son état administratif, le type d'interface physique sous-jacent (SDH,SONET,...) et l'état de cette interface. La troisième commande permet en une requête de disposer de la configuration de tous les ports d'un switch ;
- un mécanisme d'événements asynchrones, non confirmés issus par un switch pour informer le controleur de modifications importantes (activation/désactivation d'un port, ajout/retrait d'un port).

GSMP fournit également un mécanisme permettant à deux switchs de se coordonner sur un lien.

Les paramètres de QoS pour les connexions sont représentés par un mécanisme de priorité sur un port (indice de priorité par VC). Ce mécanisme simple, n'est cependant pas satisfaisant pour la plupart des approches. Aussi, des extensions ont été proposées dans le cadre des réseaux programmables.

Pour étendre le support des paramètres de QoS dans GSMP, une proposition d'extension du protocole appelée qGSMP [88] est soumise à la standardisation IETF. Elle est présentée dans l'annexe suivante.

B qGSMP

qGSMP (*GSMP with Quality of Service Extensions* [ADA 99a]) est comme l'indique son nom, une extension du protocole de contrôle GSMP permettant un traitement plus fin des caractéristiques de QoS que celui fournit dans la version initiale de GSMP. *qGSMP* se base sur le modèle de commutateur proposé par la société Xbind et repris dans l'approche P1520 (table de commutation, région d'ordonnancement, multiplexeurs de sortie / port, gestionnaire de buffers et ordonnanceur de port) présentée dans la section **??** de ce rapport.

87. Identificateur du port valide pour la durée de disponibilité d'un port.

88. *General Switch Management Protocol with Quality of Service Extensions [ADA 99a]*

Ce protocole propose une définition précise des différents composants intervenants dans la QoS: caractéristiques de trafic, QoS, région d'ordonnancement, politiques de gestion et d'ordonnancement des buffers. Les valeurs possibles de ces paramètres sont données dans la table B ci dessous.

Classe de traffic	qualitatif: Premium, Video, Voice, Audio, Data, autres
	quantitatif: débit crête, débit moyen, ère de mesure
QoS	Délai max, moyen, probabilité de perte, délai max entre pertes, ère de mesure
Estimateurs de région	aucun, propriétairer
Type d'ordonnancement	FIFO, à priorité statique, Round-Robin pondéré, propriétaire, ...
Gestion des buffers	simple, propriétaire

les valeurs des paramètres de QoS définis dans qGSMP

Afin de permettre au controlleur de commutateur de manipuler ces paramètres, *qGSMP* définit un certain nombre de commandes supplémentaires à celles déjà offertes par GSMP. Ces commandes sont :

- ajout, resp. suppression d'un VC dans la table de commutation et acquisition, resp. libération de capacité dans la région d'ordonnancement ;
- ajout, resp. suppression d'un VP (sur une machine source, un switch intermédiaire ou un puit) avec allocation de capacités associées (source et intermédiaire) ;
- possibilité de partition de la table de commutation. Partition en au point-à-point) ;
- consultation des paramètres de QoS d'un commutateur
- demande de configuration de la QoS d'un port ou d'un VP. La réponse comporte la liste des classes de trafic supportées, les paramètres de QoS supportés, les estimateurs supportés, la liste des qualificateurs qualitatifs et quantitafis supportés, les schedulers supportés, le nombre maximal de buffers ainsi que des informations sur la mémoire disponible (voir table B) pour les valeurs possibles de ces paramètres ;
- des opérations de consultation, resp. modification individuelle de ces informations pour un port ou un VPI donné. Les opérations sont : lecture, modification d'une région d'ordonnacement, lect./modif. de la politique d'ordonnacement, lect./modif. de la gestion des buffers (nombre de buffers, classes de trafic, politique de gestion), lect./modif. d'une classe de trafic (infos qualitatives, quantitatives) ;
- une opération de demande de mesure de QoS sur un port ou un VPI ;
- un événement généré par le switch et indiquant au controleur la violation d'une contrainte de QoS.

S'ajoutent à ces opérations l'ensemble des opérations définies dans la version initiale de GSMP.

Unit´e de recherche INRIA Lorraine, Technopˆole de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhˆone-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399