Services Actifs et Passerelles Programmables

Hoa Binh Nguyen

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

No attribué par la bibliothèque
|--|--|--|--|--|--|--|--|--|--|--|

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : "Informatique : Systèmes et Communications"

préparée au laboratoire **LSR-IMAG**
dans le cadre de **l'Ecole Doctorale "Mathématiques, Sciences et Technologie de l'Information"**
présentée et soutenue publiquement

par

Hoa-Binh Nguyen

le 16 janvier 2004

Services Actifs et Passerelles Programmables

Directeur de thèse : Andrzej DUDA

JURY

Mme. Brigitte Plateau ,	présidente
M. Oliver Festor ,	rapporteur
M. Ken Chen ,	rapporteur
M. Andrzej Duda ,	directeur de thèse
M. Yvon Gourhant ,	examinateur

Remerciements

Tout d'abord, je voudrais exprimer tous mes remerciements à M. Andrzej Duda - mon directeur de thèse - qui m'a accueilli dans son équipe pour effectuer ce travail au sein du laboratoire LSR. Il a consacré beaucoup de temps et d'efforts pendant ces trois années. Il a montré une capacité extraordinaire et un enthousiasme exceptionnel pour la recherche.

J'aimerais également remercier tous les membres du jury : Mme. Brigitte Plateau pour avoir accepté d'être la présidente du jury. M. Olivier Festor et M. Ken Chen pour avoir accepté d'examiner mon travail.

Je remercie l'équipe de France-Télécom R&D à Lannion : M. Yvon Gourhant et M. Bertrand Mathieu pour leur collaboration pendant cette thèse.

Tous mes remerciements à mes collègues dans l'équipe Drakkar : Franck Rousseau, Paul Starzetz, Phuong-Hoang Nguyen, Justinian Oprescu, Laurentiu-Sorin Paun, Gilles Berger-Sabbatel, Dominique Decouchant, Martin Heusse, Pascal Sicard, Jean-Luc Richier, Jacques Chassin-de-Kergommaux, Stéphane Lo-Presti, Leyla Toumi, Cristina Pop, Pawel Hadam, José Antonio Garcia Macias, Beatriz Adriana González Beltrán, Raúl García Ruiz et les autres pour leur sympathie pendant mes trois années dans l'équipe.

Je remercie particulièrement Gilles Berger-Sabbatel de m'avoir aidé à corriger ce manuscript, et pour ses remarques très utiles, à Paul Starzetz qui m'a aidé à implémenter les modules dans le noyau Linux, et à Franck Rousseau pour plusieurs discussions intéressantes pendant ce travail.

Je voudrais aussi remercier toute l'équipe de direction et les secrétaires du LSR : M. Paul Jacquet, M. Farid Ouabdesselam, Mme Christine Collet, Liliane Di Giacomo, Martine Pernice, Pascale Poulet et Solange Roche pour leur travail pour les thésards dans le laboratoire.

Je ne peux pas oublier Francois Challier et Christiane Plumeré pour leur travail exceptionnel pour maintenir le système informatique du laboratoire.

J'aimerais aussi adresser mes salutations aux autres amis vietnamiens, français, mexicains, marocains, tunisiens, brésiliens etc dans le laboratoire où règne un excellent environnement international.

Je voudrais dire toute ma gratitude et mes remerciements à ma famille : ma femme Vân, mon fils Viet Dung (Te), mes parents, mes grands-parents et mon petit frère Duong pour leur amour, leur soutien et leur compréhension depuis toujours.

Finalement je tiens à remercier tous les amis vietnamiens en France et à Grenoble pour leur amitié - je ne pourrais jamais oublier les beaux souvenirs de mes années en France.

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Motivation

Le réseau Internet rencontre actuellement un succès extraordinaire. Il devient un support de connexion pour tout type d'ordinateurs : super calculateur, PC, assistants personnels (PDA), téléphones portables, ou même appareils électroménagers à la maison. Bientôt l'Internet connectera d'autres types d'équipements, comme par exemple des dispositifs enfouis sous forme de capteurs ou d'actionneurs.

Les applications qui utilisent l'Internet sont aussi très variées. La première, le courrier électronique, a connu un succès grandissant et s'introduit maintenant même dans les réseaux de téléphonie mobile. Le transfert de fichiers a évolué progressivement vers un système d'information à l'échelle planétaire : le Web. On constate depuis peu qu'une grande partie du trafic réseau est générée par des systèmes pair-à-pair (P2P) permettant d'échanger des fichiers. D'autres applications sur l'Internet incluent la téléphonie (VoIP), la diffusion de la vidéo et de l'audio, la téléconférence, le commerce électronique, les grilles de calcul (*grid-computing*) et les jeux distribués. L'accroissement du nombre d'hôtes et du trafic véhiculé, l'élargissement du spectre des applications et l'augmentation de la complexité du réseau sont à l'origine de nouveaux besoins en termes de fonctionnalités, performances et sécurité.

L'Internet repose sur le protocole d'interconnexion IPv4 développé dans les années 60-70 [1]. Même si la version future IPv6 [2] est dans un état de normalisation stable et bénéficie déjà d'expériences de déploiement partiel, les fonctionnalités de base n'ont pas évolué de manière importante : à la fois IPv4 et IPv6 permettent de transférer un paquet d'une source à une destination en fonction de son adresse. Il est très difficile d'étendre ces protocoles ou d'ajouter des fonctions spécifiques pour offrir de nouveaux services. Deux exemples typiques sont les mécanismes de qualité de service (QoS) et les services de communication multipoint - même si de tels mécanismes ou protocoles existent, ils ne sont pas déployés à l'échelle de l'Internet. Or, dans beaucoup de situations, des applications communiquantes bénéficieraient

de certaines fonctionnalités qui, placées dans le réseau, amélioreraient les performances ou rendraient un service utile.

Cette vision du réseau minimaliste et difficilement extensible a mené à l'apparition de réseaux *actifs* ou *programmables*. L'idée de ce nouveau type de réseaux est d'ouvrir la boîte hermétique des éléments réseau en permettant d'ajouter du code exécutable à la fonction du transfert de paquets. On définit alors une machine d'exécution associée à la communication dans un nœud du réseau. Ainsi, un nœud actif peut permettre une extension de fonctionnalités de protocoles de base pour répondre aux besoins d'applications, par exemple déployer un nouveau service ou personnaliser le comportement du réseau. L'intérêt de l'exécution dynamique du code sur un élément dans le réseau peut être encore amplifié si la machine d'exécution propose l'accès à certaines fonctions internes du nœud, liées soit à la communication (routage, duplication de paquets, mise en mémoire) soit à l'information sur l'état du réseau (niveau de congestion sur un lien, nombre de flots, voisinage connu).

Le code exécutable peut être soit installé sur un nœud par un moyen externe et s'exécuter au passage des paquets, soit venir des paquets qui traversent un routeur, chaque paquet portant une partie de code à exécuter sur le routeur. Même si la terminologie n'est pas clairement déterminée, nous utiliserons le terme *réseaux programmables* pour désigner la première approche et le terme *réseaux actifs* pour la deuxième. Notons que parfois on utilise le terme *réseaux actifs* dans le sens générique qui englobe les deux approches.

Dans l'approche à base de réseaux programmables, on conserve la structure des unités de protocoles qui traversent le réseau ; le code placé de manière dynamique dans certains éléments permet d'étendre les fonctionnalités du réseau. Le problème important de cette approche est l'interface de contrôle d'un nœud programmable. C'est pour cette raison que le développement des réseaux programmables a été entrepris par la communauté OPENSIG [9] et poursuivis dans le projet P1520 de l'IEEE [10]. Cette voie propose de normaliser des interfaces programmables pour des commutateurs ATM, des routeurs IP et des réseaux de télécommunications sans fil afin de voir et de manipuler des dispositifs physiques du réseau comme des objets distribués avec des interfaces programmables bien définies. Les interfaces ouvertes permettent aux fournisseurs de services de manipuler des états du réseau en utilisant des outils intergiciels tels que CORBA pour construire et gérer de nouveaux services [7].

La deuxième voie a été initiée par les travaux de D. Wetherall et D. Tennehouse au MIT suivis par la communauté de recherche en réseaux, financée de manière importante par plusieurs projets de la DARPA (entre temps D. Tennenhouse est devenu le directeur d'un programme à la DARPA) [12]. Les premiers travaux ont proposé de considérer les paquets comme des *capsules* (cf. chapitre 2) contenant du code pouvant être exécuté dans des routeurs qui fournissent un environnement d'exécution. Cette approche gomme la différence entre les données et le contrôle - un paquet ne devient que du code qui s'exécute dans le réseau. Ceci nécessite un nouveau format pour des capsules : ANEP (*Active Network Encapsulation Protocol*) est le format utilisé dans *Abone* (*Active Backbone*), le réseau expérimental d'interconnexion de routeurs actifs.

La flexibilité offerte par des réseaux actifs et programmables paraît très séduisante, néanmoins l'ouverture des éléments réseau et l'ajout du code doit se faire avec beaucoup de précautions : du moment où l'on peut programmer ou injecter du code dans des routeurs, l'exécution peut aboutir à des effets indésirables à cause d'une erreur de programmation ou d'une malveillance. Les techniques de validation de code, l'évolution des langages de programmation ainsi que la cryptographie permettent d'espérer que ce problème crucial de sécurité trouvera des solutions satisfaisantes. Même si nous considérons le problème de sécurité comme important, notre travail s'est focalisé plutôt sur les aspects fonctionnels d'un nœud actif en laissant les problèmes d'administration et de déploiement pour la suite.

Un deuxième aspect concerne les performances. Ce souci nous impose la recherche de solutions qui ne pénalisent pas le traitement standard des paquets et qui n'ajoutent qu'un faible surcoût dans le cas de l'exécution du code. Il s'agit donc d'optimiser le fonctionnement des nœuds actifs et des plateformes d'exécution. L'augmentation de la vitesse des processeurs aide à atteindre ces objectifs de performances.

1.2 Contribution

Notre travail s'insère dans l'approche d'enrichissement des fonctionnalités de protocoles par l'ajout du code exécutable dans des éléments réseau. Nous partons du constat que l'endroit le plus approprié pour placer ce code est la bordure du réseau, avec une passerelle qui traite des flots de paquets. Dynamiquement et à la demande de l'application ou de l'utilisateur, nous voulons pouvoir activer le traitement de paquets par des processus applicatifs appelés *services actifs*. Nous cherchons à appliquer cette approche aux environnements de communication sans fils, ou aux environnements pervasifs [5] qui intègrent de petits équipements communiquants enfouis.

On peut résumer les principales contributions de cette thèse de la manière suivante :

- Nous avons conçu une passerelle active générique appelée *ProAN* supportant plusieurs environnements d'exécution différents. Elle permet d'intercepter des paquets déterminés selon certains critères et de les remonter dans l'espace utilisateur à destination d'un processus implémentant un service actif. L'activation du filtre d'interception est dynamique pour ne pas introduire un surcoût pour des flots qui ne nécessitent pas de traitement par un service actif.
- *ProAN* nous sert de support pour un environnement d'exécution de services actifs appelé *GateScript* qui offre un langage de script orienté vers la programmation à différents niveaux de protocole, de la couche réseau à la couche application. Le langage dispose de variables correspondant aux champs de protocoles sur lesquels veut travailler un service actif. Un script écrit en langage de script de *GateScript* est précompilé, stocké dans une forme intermédiaire et interpreté pour chaque paquet remonté au service actif.

- L'environnement de *GateScript* fournit des fonctionnalités permettant d'automatiser l'analyse et la construction d'unités de protocoles. Nous utilisons un langage de description pour décrire la structure des unités de protocoles traitées par des services actifs et à partir d'un fichier de description un compilateur produit automatiquement l'analyseur et le générateur des PDUs (*Protocol data Unit*). Les champs reconnus dans les paquets interceptés sont couplés avec des variables du langage de script et peuvent être utilisés directement dans le script d'un service actif. De cette manière, le programmeur de services actifs ne se focalise que sur la partie algorithmique de son script. Ceci rend aussi *GateScript* générique - il suffit de décrire la structure d'un protocole dans le langage de description pour pouvoir programmer de manière aisée des scripts travaillant sur ses PDUs.
- Nous avons exploré une nouvelle classe de services actifs que nous appelons *proactifs*. Ce sont des services actifs qui peuvent découvrir d'autres services sur le même nœud ou sur des nœuds différents. Ils peuvent obtenir des informations sur l'état d'un nœud actif, du réseau ou de l'environnement, ainsi que profiter de la présence des équipements enfouis dans l'environnement. Nous avons défini la notion de moniteur et d'évènement pour rendre ces services réactifs : ils peuvent s'abonner auprès de moniteurs pour recevoir certains évènements. Pour supporter des services proactifs, notre prototype de passerelle programmable *ProAN* intègre un service de découverte.
- Les concepts définis plus haut ont été mis en œuvre dans un prototype. D'abord nous avons développé la passerelle *ProAN* sous Linux. Elle intègre un service de découverte de Jini pour supporter des services proactifs. L'environnement d'exécution *GateScript* a été développé à l'aide de Java. Le langage de description des unités de protocoles utilise le langage Flavor et l'analyseur syntaxique JavaCC. Nous avons aussi proposé et développé plusieurs scénarios de services proactifs pour montrer leur intérêt dans les environnements pervasifs.
- Nous avons testé et mesuré les prototypes développés. Pour valider le concept de la passerelle programmable nous l'avons instantiée pour un certain nombre de protocoles : HTTP/HTML, RTP/RTCP avec le contenu MPEG et le transcodage en H.263, X Windows et SIP. Nous avons évalué les performances de l'interception de paquets dans le noyau et de l'exécution de *GateScript*. Les performances sont encourageantes grâce à la réalisation de certaines fonctions dans le noyau Linux, et se comparent favorablement aux performances de passerelles spécialisées comme Muffin, un proxy HTTP.

L'ensemble de ce travail contribue à l'élaboration d'un outil d'extension de fonctionnalités réseau, outil qui se veut flexible, générique et réactif.

1.3 Organisation de la thèse

Le reste de ce mémoire est organisé comme suit.

Le chapitre 2 présente l'état de l'art sur les réseaux actifs et programmables.

La deuxième partie, composée de quatre chapitres, conçue la contribution principale de la thèse. Le chapitre 3 présente l'architecture de la passerelle programmable ProAN.

Dans le chapitre 4 nous introduisons l'environnement d'exécution *GateScript* ainsi que son langage de programmation.

Ensuite le chapitre 5 présente la notion des services proactifs et montre comment ProAN peut les supporter. Quelques scénarios d'applications pour les environnements pervasifs sont aussi présentés

L'évaluation des prototypes et l'analyse des expérimentations sont présentés dans le chapitre 6.

Le dernier chapitre conclut le travail et discute des perspectives futures. Des informations supplémentaires et des extraits du code développé se trouvent dans l'annexe.

Première partie :

Contexte

Chapitre 2

Réseaux Actifs

Ce chapitre résume l'état de l'art et notre analyse de la recherche dans le domaine des réseaux actifs.

2.1 Introduction

La première approche des réseaux actifs permet à des paquets de contenir non seulement des données mais aussi des fragments de code qui sont exécutés dans les environnements d'exécution des routeurs actifs. Ainsi les paquets peuvent consulter des services, des informations dans le routeur, et décider ce qu'il faut faire au bon moment et au bon endroit. Les paquets deviennent actifs et sont souvent appelés capsules. Les paquets normaux qui ne contiennent pas de code sont dits passifs. Parfois les capsules contiennent seulement une référence à un programme qui sera téléchargé à partir des routeurs voisins ou de serveurs de code s'il n'est pas présent sur le routeur actuel.

Avec l'argument que les paquets n'ont pas toujours besoins d'être actifs et d'être exécutés dans tous les routeurs sur un chemin donné, la deuxième approche appelée *"routeur programmable"* ou *"nœud programmable"* permet de pré-télécharger des programmes spécifiques appelés les *services actifs* aux routeurs. Ces services peuvent traiter des paquets passifs pour donner des bons comportements du routeur en vue de l'état du réseau. Les traitements sont par exemple : la compression, le découpage, le filtrage, la combinaison des paquets. Les paquets passifs sont interceptés grâce au classificateur des paquets du routeur sur des critères concernant les informations sur les entêtes des paquets comme l'adresse source et destination, le numéro de port etc.

2.2 Applications

Pour mieux comprendre les avantages que les réseaux actifs peuvent apporter au réseau Internet, nous allons voir ci-dessous un résumé de leurs applications.

2.2.1 Déploiement rapide des nouveaux protocoles "à la volée"

Une des difficultés dans la gestion des réseaux est de faire évoluer l'infrastructure. En général, cela demande d'arrêter les machines qui doivent être mises à jour. Dans la cas où la mise à jour échoue pour une raison quelconque, le processus doit être refait en sens inverse.

Dans le projet Active Bridging [11], les auteurs voulaient montrer comment une infrastructure active permet de changer les protocoles "à la volée". Ils ont développé un commutateur (*bridge*) actif fournissant une interface programmable à des modules actifs appelés *switchlets* dont le module de gestion d'algorithme *spanning-tree* qui évite des boucles dans des réseaux Ethernet. Pour la démonstration ils ont développé deux modules de gestion de cet algorithme :

- Le switchlet DEC implémente le protocole de spanning-tree de DEC (Digital Equipment Corporation). Ce switchlet utilise les trames de type DEC envoyées à l'adresse multicast de gestion réservée à DEC pour échanger l'information. Ce protocole joue le rôle de "l'ancien protocole" qui doit être changé.
- Le switchlet 802.1D qui implémente le protocole de spanning-tree de 802.1D [15] et qui utilise les trames de type 802.1D envoyées à l'adresse multicast de gestion réservée à 802.1D pour échanger l'information. Ce protocole joue le rôle du "nouveau protocole" qui va remplacer l'ancien ci-dessus.

Ces deux switchlets sont déjà disponibles sur le commutateur actif mais seul celui de DEC est activé.

Le module de contrôle écoute sur l'adresse multicast du 802.1D. Si une nouvelle trame 802.1D arrive, le module de contrôle considère que le réseau est en train de changer le protocole. Il arrête le switchlet DEC et active le switchlet 802.1D. Le module de contrôle se met dans l'état de transition. Il laisse le switchlet 802.1D écouter sur l'adresse multicast de 802.1D, et écoute, lui, sur celle de DEC et supprime toutes les trames DEC arrivées. Le switchlet 802.1D commence à envoyer les trames de configuration à tous les ports. Cela permet à tous les autres commutateurs dans le réseau de faire la même chose s'ils ne l'ont pas encore fait. Le module de contrôle se met ensuite à l'état de surveillant. Il compare le résultat du nouveau protocole avec celui qui précède. Si l'arbre de spanning-tree n'a pas convergé pendant une période prédéterminée, le module de contrôle juge qu'il y a des erreurs dans le nouveau protocole. Dans ce cas, le nouveau protocole serait arrêté et l'ancien serait redémarré. Tous ces processus ont lieu sans intervention humaine.

Cette démonstration montre la flexibilité qu'un commutateur actif peut offrir. Mais c'est seulement vrai pour un protocole simple comme un protocole gérant l'algorithme "spanning-tree". Ce serait plus compliqué si on voulait changer le protocole IPv4 par le protocole IPv6, par exemple, avec le même mécanisme. Car dessus de IPv4, il y a d'autres protocoles comme TCP, UDP dont l'implémentation est en relation directe avec celle du protocole IPv4. Par conséquent à notre avis, une seule interface programmable offerte par des nœuds actifs n'est pas suffisante pour permettre de changer les protocoles aussi complexe que IPv4 à la volée. Il faut aussi standardiser l'interface commune entre différents protocoles et avoir un mécanisme permettant de changer dynamiquement le code dans tous les systèmes d'exploitation. Malheureusement ce n'est pas le cas aujourd'hui. De nouveaux systèmes d'exploitation et de nouveaux protocoles apparaissent de plus en plus nombreux mais sans prendre en compte l'évolution des protocoles supportés.

2.2.2 Gestion des réseaux

Dans la gestion des réseaux, la scrutation (*polling*) est utilisée pour collectionner des informations anormales dans les équipements des réseaux. Parce qu'il y a une forte variabilité du nombre de nœuds, de leur complexité et du traffics dans les réseaux, l'état d'un équipement peut varier très vite. Par conséquent, la gestion a besoin d'une nouvelle technique qui demande moins de communication et permet d'avoir des actions plus efficaces dans des équipements des réseaux. Le projet Smart Packets [31] rend les nœuds programmables pour en permettre une meilleure gestion. Les centres de gestion peuvent envoyer des programmes contenant des règles de gestion à ces nœuds. Ces règles de gestion sont exécutées sur places et ces programmes peuvent traverser plusieurs nœuds avant de retourner au centre original. Il y a donc trois avantages :

- L'information retournée au centre de gestion peut être coupée ou conçue pour l'intérêt du centre. Donc cela réduit le trafic.
- Beaucoup de règles de gestion utilisées au centre de gestion peuvent maintenant être encapsulées dans des programmes qui, quand ils sont envoyés aux nœuds, identifient automatiquement et corrigent des problèmes sans demander plus d'intervention du centre.
- Un seul paquet traversant les réseaux peut remplacer des séries d'opérations SET et GET.

2.2.3 Contrôle de congestion

Le système traditionnel de contrôle de congestion détecte et corrige la congestion seulement de bout en bout : les routeurs ne participent pas à cette tâche. Les nœuds ne se rendent compte de la congestion que lorsqu'il y a des paquets perdus : délai d'attente dépassé (timeout) ou duplication de paquet d'accusé de réception (Ack). Dans le système ACC [32]

de Faber, le contrôle de congestion est migré dans des routeurs qui détectent la congestion, et réagissent tout de suite en modifiant le trafic déjà entré dans le réseau et en avertissant l'émetteur. Le routeur est programmé par le premier paquet de la connexion qui lui indique comment réagir à la congestion, et les paquets suivants contiennent l'état de l'algorithme de contrôle de congestion de l'émetteur (la taille de la fenêtre par exemple). Quand le routeur détecte la congestion, il détermine quelle action l'émetteur aurait fait s'il avait détecté lui-même la congestion. Cela se fait sur la base de l'état de l'émetteur. Le routeur installe ensuite des filtres qui suppriment des paquets que l'émetteur n'aurait pas envoyé.

Dans [33] et [34], S. Bhattacharjee propose un nœud actif qui contient un nombre de fonctions spéciales conçues pour rejeter des paquets quand il y a de la congestion. Dans les routeurs, un tampon sera alloué pour chaque flot de donnée. Chaque paquet contient un identificateur indiquant la fonction de rejet utilisée pour traiter ce paquet. Un exemple d'une telle fonction est celle appelée "*Unit-Level Dropping*" (ULD) qui rassemble des paquets en unités d'application. Par exemple une unité d'un flot vidéo MPEG peut être une trame ou un GOP (*Group Of Picture* - une trame I suivies par des trames P et B jusqu'à la prochaine trame I). Donc si un paquet est rejeté, la fonction de rejet va aussi jeter tous les paquets appartenant à la même unité, puisque l'unité serait perdue s'il manque un composant. Le résultat obtenu est nettement meilleur que le résultat traditionnel. On peut construire bien sûr une autre fonction de rejet qui analyse l'importance des trames et rejette seulement des paquets dans des trames qui sont moins importantes que les autres (des trames B par rapport à P par exemple).

Ces projets ont bien montré qu'il est possible de migrer la fonction de contrôle de congestion dans les routeurs mais nous pensons que le meilleur endroit où on a intérêt à placer des routeurs avec cette capacité de contrôle de congestion intégrée est dans les frontières entre le réseau fixe et le réseau sans fil qui a souvent des débits limités et des variations importantes.

2.2.4 Communication multi-points (Multicast)

Dans l'architecture d'AMnet [35], les nœuds actifs ont un environnement d'exécution pour des services actifs qui adaptent la qualité de service du flot de données (e.g. en le transformant en différents formats) pour des participants ayant des capacités différentes dans une session de communication multi-points. Dans le cadre du projet PANAMA (*The Protocols for Active Networking with Adaptive Multicast Applications*), le protocole AER [36] (*Active Error Recovery Protocol*) a été utilisé par les services actifs appelés services de réparation - *repair services*. Ces services sont injectés dans des routeurs actifs (appelés serveurs de réparation - *repair servers*) à des points stratégiques dans l'arbre de distribution pour stocker l'ensemble de paquets récents a fin de répondre rapidement à la demande de retransmission des paquets perdus. Des serveurs de réparation demandent eux-même des paquets perdus aux serveurs de réparation du niveau supérieur ou de la source. Cela évite le problème d'explosion des paquets de notification à la source.

Nous pensons aussi que les routeurs actifs trouveraient leur applications dans le domaine des communications multi-points. Pour l'instant il y a de plus en plus des applications à communications multi-points comme la découverte des services ou l'attribution automatique des adresses IPs mais elles restent seulement dans le contexte des réseaux locaux.

2.2.5 Amélioration du cache

Dans [37], Legedza et Guttag proposent une solution basée sur la technologie de réseaux actifs pour augmenter l'efficacité de la recherche de documents Web non populaires parce que la recherche de tels documents invoque en général une série de transmissions entre des serveurs de cache et des vérifications inutiles. Par conséquent cela génère des trafics non désirés. Dans leur solution, chaque routeur participant maintient une table indexée par des URLs [21] associés avec des adresses de serveurs de cache (ou des pointeurs de re-direction) qui stockent un tel document. Cette table est en fait indexée avec l'index de hachage MD5 [17] de l'URL. Pour éviter de le recalculer à chaque nœud, cet index de hachage est placé dans l'en-tête du paquet SYN (paquet d'établissement de connexion) de la connexion TCP [19] de la requête. Quand ce premier paquet SYN de la requête Web traverse l'Internet vers le serveur de l'URL, les routeurs sur ce chemin cherchent dans la table le pointeur de re-direction en utilisant l'index de hachage de l'URL placé dans le paquet SYN. Le premier routeur qui trouve un pointeur de re-direction va transmettre la requête au serveur de cache. S'il n'en existe aucun, la requête va arriver au serveur de l'URL. Par conséquent, la requête sera toujours transmise sur le chemin le plus court (au sens de l'algorithme de routage) au serveur de l'URL. Pour réaliser ce modèle, il faut mettre l'index de hachage de l'URL dans le paquet SYN de TCP car une requête HTTP [20] invoque toujours un paquet d'établissement de connexion de TCP. Donc on a besoin d'ajouter une option IP et de changer un peu le comportement des routeurs ainsi que des navigateurs Web qui doivent simplement insérer une option avant d'envoyer chaque paquet SYN.

Le mécanisme actuel de cache consiste à mettre des proxies de caches à des endroits stratégiques. Mais dans [38] et [39] une autre approche propose de mettre des petits caches dans tous les nœuds du réseau. Quand une réponse contenant un objet demandé les traverse, ces nœuds décident de stocker l'objet ou pas. Une politique de décision basée sur le rayon qui est le nombre de nœuds traversés a été proposée. Sur le chemin du serveur au client, l'objet est stocké aux nœuds distants d'un rayon. Le rayon qui est un paramètre de la politique de gestion peut être fixé globalement ou en se basant sur des objets, ou même il est différent dans des réseaux différents.

Il y a d'autres alternatives pour augmenter l'efficacité des systèmes de cache autres que les réseaux actifs comme CDN (*Content Delivery Networks*) [40] ou en utilisant l'espace disque local pour cacher des contenus déjà visités qui peuvent répondre tout de suite au besoin des applications ou des fournisseurs des contenus. Parce que les réseaux actifs ne peuvent pas encore montrer un vrai advantage par rapport à ces alternatives, ils ne sont pas très répandus pour ce type d'application de cache.

2.2.6 Aide au serveur

Dans [41], un serveur de vente aux enchères est amélioré grâce à des services actifs effectuant du filtrage déployés dans des nœuds actifs autour de ce serveur. Ces services actifs suppriment des demandes dont le prix de l'article est inférieur à celui qui est dans le serveur. Cela évite au serveur de traiter des demandes périmées. Nous avons remarqué que les serveurs de vente aux enchères comme `ebay.com` ou des serveurs de moteurs de recherche comme `google.com` utilisent souvent le technique de *"server-cluster"* et *"DNS round-robin"* pour distribuer et équilibrer les charges aux différentes machines. Ceci montre encore qu'il y a toujours une solution alternative aux réseaux actifs.

2.2.7 Téléphonie

Maxemchuk argumente dans [42] que les réseaux actifs sont plus avantageux dans les réseaux téléphoniques (PSTN) que l'Internet parce que l'utilisateur d'un PSTN a un terminal moins puissant que dans l'Internet. Par conséquent, l'ajout d'intelligence dans le réseau téléphonique augmente plus les capacités de l'utilisateur. La ligne de l'utilisateur au premier commutateur (la boucle locale) peut supporter beaucoup plus que le signal numérique actuel de 64 kbps ou un signal analogique de largeur de bande de 3 kHz. On peut donc augmenter la qualité d'une connexion en modifiant la fonction d'un commutateur sans changer la boucle locale. Les réseaux actifs permettent de télécharger des codeurs actifs dans les commutateurs pour que chaque connexion puisse choisir le taux de débit (bit rate) ou la largeur de bande. Les fabricants peuvent y introduire des algorithmes de codage de meilleur qualité. De plus, un programme dans le commutateur peut détecter les moments de silence dans une connexion (e.g un accès à Internet à partir d'un modem) et libère cette ligne pour une autre connexion mais en mémorisant la destination et re-établit quand la connexion devient active. Cela diminue le nombre de commutateurs pour une compagnie. Les réseaux actifs peuvent aussi utiliser le réseau téléphonique pour améliorer la qualité de l'Internet pour que ce dernier puisse véhiculer des voix de bonne qualité. Une capsule contient un programme et des données qui sont nécessaires pour observer le délai et la perte quand la capsule traverse le réseau. Si une partie de l'Internet est congestionnée, un nœud actif peut utiliser le réseau téléphonique pour transmettre le segment bloqué.

Ces idées sont déjà incorporées dans le concept et l'infrastructure de la nouvelle génération des réseaux téléphoniques - NGN (*Next Generation Networks*) [43] qui transforme les réseaux téléphoniques en réseaux de données et offre aussi une interface programmable aux fournisseurs de services.

2.2.8 Sécurité

Un problème de sécurité très connu est celui de *SYN-flooding* [22]. Il consiste à attaquer un serveur en profitant de la phase d'établissement de connexion du protocole TCP. Un client notifie un serveur qu'il veut établir une connexion TCP en lui envoyant le paquet de synchronisation appelé SYN. Le serveur répond au client avec le paquet SYN-ACK et va attendre le paquet ACK du client. A ce moment là, chez le serveur, une demi-connexion existe déjà - le serveur garde toutes les informations nécessaires pour compléter une connexion dans une structure de données. Il est dans l'état SYN-RECEIVED. Si le client répond avec le paquet ACK, la connexion sera complète (il passera à l'état ESTABLISHED) et sinon au bout d'un certain temps cette structure sera effacée. Un malveillant peut réaliser une attaque de type "déni de service" en envoyant au serveur un grand nombre de paquets SYN a fin de le saturer (la queue des connexions en état SYN-RECEIVED est pleine). Et le serveur ne peut pas alors répondre à d'autres demandes de service. Beaucoup d'efforts ont été consacrés à ce problème et quelques solutions ont été proposées comme : augmenter le nombre de demi-connexions accepté par le serveur, *SYN-Cookies* [24], *SYN-Cache* [23] ou *Ingress Filter* [25].

Dans la solution *SYN-Cache*, le serveur alloue une structure plus petite que normale avec un minimum d'information quand le premier paquet SYN arrive. La structure complète serait allouée totalement quand la connexion est bien établie. Cela permet au serveur d'accepter plus de connexions. Dans celle de *SYN-Cookie*, le serveur continue à recevoir les paquets SYN quand la queue des connexions en état SYN-RECEIVED est pleine. Si cette queue est pleine, le serveur crée un index cryptographique de 32 bits appelé "cookie" avec les informations (l'adresse source et destination, le numéro de port etc.) du nouveau paquet SYN venant d'arriver. Aucune structure n'est créée sur le serveur à ce moment. Ce cookie est utilisé comme numéro de séquence dans le paquet SYN-ACK qui sera renvoyé au client. Le cookie (plus 1) sera retourné au serveur comme le numéro de notification dans le paquet ACK du client légitime. Grâce à ce cookie, le serveur peut créer la structure complète représentant l'état ESTABLISHED d'une connexion TCP sans passer par l'état SYN-RECEIVED. Par conséquent SYN-Cookie permet au serveur d'accepter des connexions même dans le cas où la queue des connexions en état SYN-RECEIVED est pleine. Nous constatons aussi que dans les deux solutions le serveur modifie la stratégie d'allocation de ressources, mais aucune de ces solutions permet d'arrêter l'attaque.

La solution *Ingress Filter* demande à tous les fournisseurs d'Internet (ISP) de filtrer des paquets IP sortant d'un réseau pour assurer qu'ils aient une bonne adresse de source avant d'entrer dans l'Internet. Mais cela dégrade la performance car il faut tout filtrer. Dans [44] Van C. Van propose une solution appliquée seulement dans un contexte des réseaux actifs en utilisant le système ANTS [13]. Cette solution consiste à déployer des filtres dynamiques qui suppriment les faux paquets SYN. Ces filtres actifs vont être déployés de plus en plus haut vers la source de l'attaque pour filtrer les faux paquets le plutôt possible. Cela évite de déployer des filtres partout comme dans la solution *Ingress Filter*.

Comme indiqué, la solution proposée par Van C. Van s'applique seulement aux réseaux totalement actifs. Mais les attaques les plus récentes que nous constatons sont souvent liées au malfonctionnement des serveurs (Appache, OpenSSH, ou Windows 2000) et des virus se répandent par le courrier électronique comme *Blaster*, *Red Code* etc. Après avoir affecté une machine, le virus prend le contrôle et commence à lancer d'autres attaques. Ces trous de sécurité ne sont généralement pas liés aux réseaux mais aux applications. Il serait très intéressant si une solution des réseaux actifs peut limiter ce phénomène.

2.2.9 Réseaux mobiles

Dans les réseaux mobiles, le débit et le taux d'erreur des liens sans fil changent au cours du temps ainsi que la connectivité des terminaux mobiles. Dans ces réseaux, les codes de correction d'erreur FEC (*Forward Error Correction*) sont souvent utilisés pour corriger les erreurs. Mais en ajoutant les codes FECs, on augmente la taille des paquets à transmettre. En cas du taux d'erreur élevé, plus de bits FECs seront nécessaires. En cas du taux d'erreur faible, ajouter des bits FECs serait inefficace. Par conséquent un code FEC constant n'est pas adapté au cas de taux d'erreur variable. Un service actif peut surveiller le taux d'erreur d'un lien sans fil et ajuster en conséquent le taux des codes FECs insérés dans les paquets transmis.

Quand le terminal mobile est déconnecté, un autre service actif peut cacher les paquets destinés à ce terminal ou il peut supprimer des paquets qui sont périmés ou des paquets moins importants. Quand la connexion est re-établie, le service retransmet ces paquets au terminal.

Plusieurs autres applications des réseaux actifs dans les réseaux mobiles ont été évoquées dans [45]. Une de ces applications est de pouvoir télécharger un algorithme de décision de hand-off qui tient compte des paramètres de l'environnement dans le terminal actif. Par conséquent, selon chaque situation de l'utilisateur (type de mouvement : en train, en voiture, à pied etc.), on peut déployer différents algorithmes de décision de hand-off qui décide d'attacher le terminal au point d'accès correspondant le mieux à cette situation.

Nous pensons aussi que les réseaux actifs peuvent contribuer à l'évolution des réseaux mobiles mais que les services actifs doivent avoir un comportement plus réactif par rapport à l'état du réseau ou de son environnement. Dans cette thèse, nous développerons un tel comportement que nous appelons "proactif". Il s'agit de services actifs pouvant réagir aux changements de l'environnement.

2.2.10 Réseaux ad-hoc

Le projet ARRCANE (*Active Routing and Resource Control for Ad Hoc Networks*) [49] a proposé d'utiliser l'approche des réseaux actifs pour les protocoles de routage dans les réseaux

ad-hoc. Puisque les connexions entre les nœuds dans ces réseaux sont souvent changées, cela demande des adaptations et des mises à jour de la table de routage constantes. Différents algorithmes de routage peuvent être implémentés comme des applications actives et sont facilement déployés et exécutés parallèlement dans ces nœuds ad-hoc. Cela permet de choisir le meilleur protocole pour une situation donnée.

Nous pensons qu'il est important de développer des applications dans les réseaux ad-hoc et ensuite d'en déduire des algorithmes de routage appropriés. La caractéristique ad-hoc devrait apparaître dans la plupart des applications pour les environnements pervasifs dans l'avenir. En combinaison avec les environnements mobiles, nous pensons que les réseaux actifs trouveraient leur place dans ce type d'environnements.

2.2.11 Grilles de calcul (Grid-Computing)

La grille de calcul [50] est une nouvelle application qui propose de regrouper des machines distantes pour résoudre des problèmes demandant beaucoup de ressources de calcul comme l'observation terrestre, la recherche des genes, la recherche des extra-terrestres etc. L. Lefèvre et les autres dans [51] ont proposé d'utiliser les nœuds actifs pour gérer des clusters de machines qui sont dédiées à ces traitements. Ces nœuds actifs peuvent améliorer le processus de déploiement des applications en implémentant un protocole de communication multipoint fiable. De plus, ces nœuds actifs peuvent fournir des informations intéressantes concernant les états des réseaux et des tâches s'exécutant dans les clusters. Des fonctions comme l'agrégation et la compression des données peuvent aussi améliorer la qualité de service de ces applications.

2.2.12 Réseaux privés virtuels(VPN)

L'idée d'injecter des programmes dans des passerelles programmables pour fournir des VPNs (*Virtual Private Networks*) est présentée dans le travail de R. Maresca [48]. Dans cette architecture, les nœuds actifs effectuent toutes les tâches concernant la communication sécurisée entre des réseaux locaux distants. Les fonctions comme la gestion des hôtes du VPN, le chiffrement des paquets, la QoS entre les flots, les algorithmes de classification, la fonction d'envoi des paquets sont implémentées comme des programmes injectables dans des nœuds actifs. Cela permet par exemple de changer et de choisir les algorithmes de chiffrement sur mesure pour chaque type de flot de communication, ou de fournir la QoS pour certains flots de données à la demande de l'utilisateur.

2.2.13 Conclusion

Nous voyons jusqu'à maintenant à travers ces divers projets de recherche qu'une architecture ouverte présente beaucoup d'avantages. Elle permet une évolution rapide des services et

une meilleure adaptation des fonctionnalités aux besoins d'applications. Les réseaux informatiques et de télécommunications convergeraient pour former un réseau global unique plus efficace, plus sûr et plus performant. Une architecture ouverte semble indispensable, mais cela demande aussi beaucoup de temps et d'efforts pour remplacer l'infrastructure existante.

Après le panorama des plusieurs applications des réseaux actifs à travers différents projets de recherche en cours, nous présentons dans la section suivante différentes architectures de ces réseaux.

2.3 Architectures des nœuds actifs

2.3.1 Première génération

ANTS (*an Active Node Transfer System*)

Le première prototype d'un réseau actif est celui de l'ANTS [13] développé au MIT par D.J. Wetherall. Voici les composants principaux de ce système :

1) **Capsules** : Les paquets sont remplacés par des capsules. Une capsule est une combinaison de données et du code qui traite ces données. Le code dans une capsule est exécuté dans tous les nœuds que la capsule traverse. Ce code est fixé par l'émetteur et ne peut pas être modifié quand la capsule traverse le réseau. Un protocole est un ensemble de types de capsules concernés. Le code dans une capsule peut accéder aux données qu'elle transporte grâce aux méthodes suivantes :

- `getSrc()` : retourne l'adresse source de la capsule
- `getDst()` : retourne l'adresse destination de la capsule
- `setDst()` : change l'adresse destination
- `getPrevious()` : retourne l'adresse du nœud précédent traversé
- `getType()` : retourne le type de la capsule
- `getResources()` : retourne le nombre d'exécution restant(TTL-*Time To Live*).

Le code d'une nouvelle capsule contient les méthodes suivantes pour que les nœuds actifs puissent les manipuler :

- `evaluate()` : c'est la fonction de routage. Elle accepte le nœud local en paramètre ce qui permet d'accéder aux services de ce nœud. Le nœud local est représenté par un objet de la classe `Node`. Les utilisateurs peuvent implémenter différents comportements de la capsule à travers cette méthode.
- `serialize()` : transforme l'objet de capsule en binaire pour la transmission
- `deserialize()` : construit l'objet de capsule à partir de la représentation binaire reçue du réseau.

2) **Nœuds actifs** : Une des difficultés dans l'architecture d'un réseau programmable est le fait d'accepter l'exécution de programmes définis par les utilisateurs en se protégeant des interventions non voulues dans les nœuds actifs au sein du réseau. En effet, les nœuds doivent non seulement offrir à des protocoles une vue cohérente du réseau, mais aussi allouer et maintenir des ressources. Un nœud actif (représenté par un objet de la classe `Node`) dans ANTS propose quatre catégories d'APIs que le code d'une capsule peut invoquer :

- l'accès à l'environnement (pour demander l'adresse du nœud, le table de routage etc.)
- la manipulation des capsules (accéder à l'entête et aux données des capsules)
- des opérations de contrôle (permettre à des capsules de créer d'autres capsules, de se copier ou de s'effacer elles-mêmes).
- des opérations pour manipuler les états des objets définis par une application pendant une courte durée de vie.

Les services suivants de la classe `"Node"` représentant un nœud actif ANTS sont disponibles :

- `address()` : retourne l'adresse du nœud local.
- `routeForNode()` : envoie une capsule à un nœud donné en utilisant la table de routage par défaut.
- `sendToNeighbors()` : envoie une capsule à tous les nœuds voisins
- `deliverToApp()` : délivre une capsule à une application donnée.
- `time()` : retourne l'heure actuelle
- `softstate.get(), softstate.put() , softstate.remove()` : pour manipuler des objets dans le cache.

ANTS est basé sur Java. La naissance de languages comme Java a favorisé l'imagination et la création de valeur ajoutée dans des services comme le Web, les agents mobiles, les systèmes embarqués et récemment les réseaux actifs. L'avantage que ANTS apporte est de pouvoir spécifier tout le comportement d'un réseau et des machines participantes par un langage de programmation. Donc, on n'a plus besoin de normaliser aucun protocole.

Netscript

Netscript [68] utilise le modèle dit *orienté* traitement de flots des données qui consiste d'un ensemble de programmes (appelés aussi des boîtes) interconnectés par leur portes d'entrée et de sortie pour traiter un flot de données.

Ces programmes sont écrits en langage de *Netscript* qui est différent des langages impératifs conventionnels (comme le C ou le Java) du fait que :

- des programmes ou des protocoles sont construits en connectant des boîtes plus simples. Dans les langages impératifs l'unité de composition est la fonction.
- c'est l'arrivée d'un flot de données qui détermine quelle boîte doit fonctionner. Dans les langages impératifs c'est le compteur du programme qui détermine quelle part du programme est active.
- et enfin dans ce modèle il n'y a pas de primitive pour contrôler le partage de l'état des objets (i.e. verrous, semaphores) car les boîtes sont indépendantes.

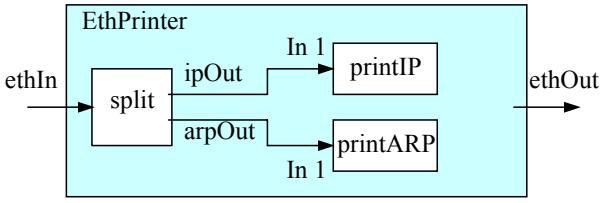

FIG. 2.1 – Composition des boîtes sous Netscript

La figure 2.1 montre un exemple de ce modèle de composition. Dans cet exemple, l'EthPrinter est une boite qui filtre et imprime des trames Ethernet. Elle reçoit un flot de trames Ethernet de sa porte d'entrée (ethIn). Elle a également sa porte de sortie (ethOut). Chaque trame Ethernet est passée dans une boite appelée "Split" qui répartit son flot d'entrée en deux flots de sortie et écrit dans deux portes : ipOut et arpOut. La boite Split examine le champ de protocole du paquet encapsulé dans la trame d'entrée et envoie des trames contenant des paquets IP à la porte de sortie ipOut et des trames contenant des paquets ARP à la porte arpOut. Ces deux portes de sorties du Split sont connectées respectivement aux boites printIP et printARP qui impriment simplement des trames d'entrée à l'écran.

Les programmes en *Netscript* sont compilés vers des classes de Java et sont ensuite téléchargés dans l'environnement d'exécution de *Netscript* pour exécuter.

L'environnement d'exécution de *Netscript* est appelé le Moteur du Réseau Virtuel ou VNE (*Virtual Network Engine*). Un nœud peut avoir un ou plusieurs VNEs. Quand un paquet contenant une entête de *Netscript* arrive à un VNE, c'est l'en-tête du paquet qui détermine quel programme de *Netscript* doit traiter ce paquet.

2.3.2 Deuxième génération

Après quelques années, en 1999 le groupe des réseaux actifs du DARPA a publié des documentations sur l'architecture commune des composants et des interfaces d'un nœud actif appelé : l'Architecture pour les Réseaux Actifs (*Architectural Framework for Active Networks*) [27]. Cette architecture décrit comment traiter des paquets et gérer des ressources dans un nœud actif. La fonctionnalité d'un nœud est divisée en trois composants principaux : le Système d'Exploitation du Nœud (*Node Operating System* - NodeOS), les Environnements d'Exécutions (EEs) et les Applications Actives (AAs) comme montrés dans la figure 2.2.

Le NodeOS est responsable de l'allocation et de la gestion des ressources du nœud actif (la bande passante des liens, le CPU ou le stockage) et fournit un ensemble de capacités de base

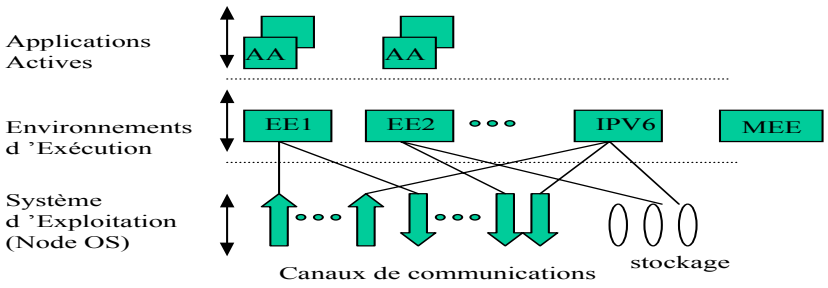

FIG. 2.2 – Architecture d'un nœud actif

utilisées par des EEs pour implémenter une machine virtuelle. Un EE offre une interface de programmation ou une machine virtuelle qui peut être programmée par les AAs. Les utilisateurs obtiennent des services du réseau actif via ces AAs. Le NodeOS isole un EE de la gestion des ressources et de l'effet du fonctionnement des autres EEs. De leur côté les EEs cachent au NodeOS les détails de l'interaction avec les utilisateurs. Quand un EE demande un service ou une ressource au NodeOS, la demande est accompagnée d'une signature grâce à laquelle, le NodeOS peut décider d'allouer le service ou pas.

Chaque NodeOS a un EE particulier appelé l'Environnement d'Exécution de Gestion (MEE) qui assure les tâches suivantes :

- Maintenance de la base de données de la politique de sécurité.
- Chargement de nouveaux EEs, ou mise à jour et configuration des EEs existants.
- Support de l'installation des services de gestion des réseaux depuis un centre de gestion distant.

Il est recommandé que les EEs offrent la possibilité de servir plus d'une AA en même temps. Une première spécification de l'interface d'un NodeOS est décrite dans [28] ainsi qu'une architecture pour la gestion des réseaux actifs dans [29].

Voici quelques premiers systèmes d'exploitation du nœud actif.

Bowman

Bowman [30], implémenté en espace d'utilisateur du système Unix System V, supporte les multiprocesseurs et offre une bonne performance en utilisant les extensions de POSIX pour contrôler les processeurs. Il est un NodeOS pour l'environnement d'exécution CANE (*Composable Active Network Elements*).

Bowman offre trois abstractions de base pour supporter les fonctionnalités du réseau :

- `Les canaux` (Channels) : qui représentent la ressource de communication exportée par

Bowman. Ce sont des points de communication pour envoyer et recevoir des paquets. Les canaux peuvent inclure des traitements complémentaires : compression ou FEC (*Forward Error Correction*) sur des paquets. Bowman exporte un ensemble de fonctions aux EEs pour créer, détruire et communiquer avec des canaux. Bowman propose un algorithme de classification de paquets très flexible et configurable, et par conséquent les EEs peuvent spécifier des paquets qu'ils veulent recevoir.
- `A-flows` : c'est l'abstraction d'une unité d'exécution qui comprend le contexte d'exécution et des états d'utilisateurs. Chaque A-flow consiste au moins un thread et s'exécute de la part d'un utilisateur identifié.
- `State store` : offre un mécanisme permettant à des A-Flows d'enregistrer et de récupérer des états qui sont indexés par une clé unique.

Joust

Joust [52] comprend trois composants :
- Le système d'exploitation de Scout [53] (Scout OS)
- Une machine virtuelle de Java (JVM) optimisée pour supporter les applications du réseau actif.
- Un compilateur JIT (*Just-in-Time*) qui translate des byte-codes de Java en instructions natives.

Scout est écrit en C et s'exécute sur des processeurs Intel Pentium et Digital Alpha. C'est un système d'exploitation configurable par des modules qui sont des unités indépendantes. Ces modules offrent des fonctionnalités bien définies. Les bons exemples sont des modules implémentant des protocoles du réseau comme IP, UDP, TCP ou des modules implémentant des composants de stockage du système comme NFS ou SCSI. Les modules sont connectés par un graphe afin de donner une configuration du noyau de Scout pour une application. De plus, Scout ajoute une abstraction orientée communication - le canal (chemin) qui englobe des données d'entrée et de sortie quand elles passent par le système. Chaque canal définit l'ensemble des modules appliqués aux données. Les canaux sont dynamiquement créés et détruits. Scout fournit également un mécanisme de gestion des ressources consommées par des canaux en limitant le nombre de modules associés à un canal.

La machine virtuelle Java (JVM) de Joust est un module de Scout qui implémentant une machine virtuelle qui est modifiée pour supporter efficacement des applications de réseaux actifs ainsi des fonctionnalités de Scout comme la création des canaux. La JVM de Joust permet aussi aux applications d'intervenir dans les décisions d'allocation de ressources.

Des classes de java (des capsules) sont dynamiquement chargées dans la JVM de Joust et sont transformées en codes natifs en utilisant le compilateur JIT pour obtenir une bonne performance.

Snow on Silk

Snow on Silk [55] comprend deux parties : Silk (*Scout in Linux Kernel*) est une version du système d'exploitation de Scout présenté ci-dessus dans le noyau du Linux. Snow est

l'implémentation de la spécification de l'interface du NodeOS sur Silk. Cela traduit le nom " Snow on Silk ". Snow s'exécute dans l'espace d'utilisateur du Linux. Et c'est la première implémentation du NodeOS dans Linux.

Janos - *Java-oriented active network operating system*

Janos [54] est une autre architecture pour les nœuds actifs basée sur Java. Le but de Janos est de proposer une machine virtuelle Java ayant la possibilité d'isoler différentes applications actives. Par conséquent, Janos est plus fiable et sécurisé pour les applications actives écrites an Java. Une autre propriété de Janos est qu'il propose aussi un mécanisme de contrôle de ressources comme la mémoire, les cycles de CPU, la bande passante, et aussi les stockages des données en cache. L'architecture de Janos est présentée dans la figure 2.3 et comprend les principaux composants suivants :

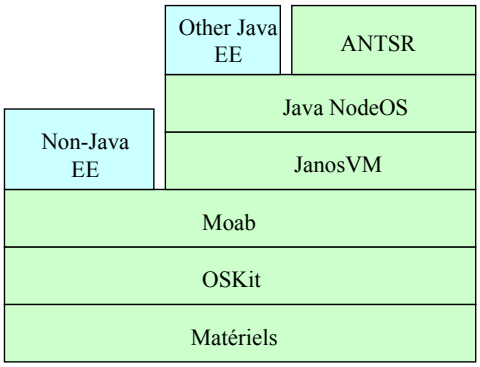

FIG. 2.3 – Architecture du Janos

- L'architecture OSKit comprend des librairies pour construire un système d'exploitation.
- Moab est l'implémentation du standard NodeOS API sur l'OSKit en C. La combinaison de l'OSKit et Moab fournit un NodeOS implémenté purement en langage C.
- JanosVM est une machine virtuelle de Java avec le contrôle de ressource.
- Java NodeOS est un système d'exploitation implémenté en Java sur JanosVM pour offrir une implémentation des APIs de NodeOS en Java.
- ANTSR est l'implémentation de l'environnement d'exécution d'ANTS sur Java NodeOS de Janos pour offrir aux nœuds actifs dans ANTS des possibilités de contrôle des ressources vis à vis des capsules.

PromethOS

PromethOS [56] basé sur l'architecture de Netfilter [73] est aussi un routeur pour les applications actives sous Linux implémenté dans le noyau. Chaque application active sous PromethOS est un module (plugin) chargeable dans le noyau Linux. PromethOS fournit une architecture pour la gestion de ces plugins.

Noeuds actifs hautes performances

Plusieurs approches ont été proposées pour augmenter la performance des nœuds actifs : le support par un cluster de machines ou un assistant du routeur. Dans Tamanoir [58], chaque nœud actif est un cluster de machines reliées par un réseau Gigabit. Les flots actifs sont répartis par une machine de distribution de charge. Un compilateur JIT (*Just in Time*) pour Java est aussi utilisé pour augmenter la performance de l'exécution des services écrits en Java.

Dans le travail du Router Assistant [57], l'Assistant est une machine ou une groupe de machines reliées avec le routeur traditionnel par un lien haut débit de type gigabit ethernet par exemple. Le routeur traditionnel fait le travail de relayage des paquets. S'il rencontre un flot actif, il le renvoie à l'Assistant qui supporte l'exécution de code actif. Tous les paramètres (table de routage, les états des interfaces etc) du routeur sont transférés à l'Assistant pour simuler l'environnement d'exécution du routeur sur l'Assistant. Par conséquent les codes actifs croient qu'ils sont exécutés sur le routeur principal. Après le traitement, les paquets actifs sont renvoyés au routeur pour continuer leur chemin. De cette manière, les paquets passifs ne subissent pas de perte de performance. De plus, on peut déployer facilement les nœuds actifs n'importe où dans le réseau, y compris au cœur du réseau Internet.

2.3.3 Nouvelles approches

Services actifs

Pendant que beaucoup de projets se concentrent sur l'architecture et l'implémentation des nœuds actifs ainsi que sur des langages spécifiques pour la composition des services actifs, une nouvelle approche est née en se basant sur l'argument suivant : un ensemble très large d'applications (i.e. service de transcodage ou proxy) n'ont pas vraiment besoin d'une architecture active complexe où beaucoup de problèmes ne sont pas encore totalement résolus. Ces applications peuvent être supportées par une architecture de services programmables construits au dessus des services d'Internet existants si elle permet aux utilisateurs de télécharger et d'exécuter du code (application) à des emplacements stratégiques dans le réseau. Ce code ou application est appelé "*service actif*". Bien sûr une telle approche ne permet pas de programmer des niveaux plus bas dans le réseau, ce n'est donc pas la solution pour résoudre des problèmes comme "SYN-flooding" or le routage intelligent etc. Pourtant cette infrastructure de services actifs enrichit le domaine des réseaux actifs et elle résout le problème de compatibilité avec le réseau Internet actuel.

Dans [59] E . Amir et les autres ont proposé une infrastructure appelée AS1 (*Active Service version 1*) comprenant six composants suivants :

- *Service de localisation* : c'est l'entité avec laquelle un client doit prendre contact pour initialiser un service actif désiré. Le service actif est appelé "agent de service" ou *"servent"*. Il y a un mécanisme pour qu'un client aie un rendez-vous avec un AS1. On peut soit utiliser la communication multi-points soit utiliser une méthode centralisée grâce à un serveur.
- *Service d'environnement* définit la partie active des services actifs, i.e. le modèle de programmation et les interfaces pour manipuler les ressources disponibles pour des services actifs dans l'AS1. Mais cet environnement ne permet pas aux services actifs de manipuler la table de routage, d'utiliser des fonctions d'envoi des paquets ou de gestion des réseaux.
- *Service de gestion* alloue des ressources aux services actifs pour avoir un équilibre de charges du CPU. Ce service effectue aussi le contrôle d'admission des services actifs ou la création de ceux-ci.
- *Service de contrôle* : une fois un service actif instancié dans AS1, le client doit pouvoir dynamiquement le re-configurer et le contrôler via un service de contrôle.
- *Service d'attachement* aide un client qui n'a pas de moyen de contacter directement les AS1s à avoir ce contact grâce à un mécanisme de " tunnel ".
- *Service de composition* permet à des clients de contacter plusieurs AS1s et d'interconnecter des services actifs dans ces AS1s.

Les auteurs ont implémenté le model AS1 en se basant sur la plateforme "*MASH*" qui est un interpréteur du langage Tcl [107] avec des capacités de multimédia en temps réel et de réseaux. L'interface proposée par AS1 est un ensemble de classes d'objets Tcl qui peuvent être invoqués par des services actifs (des programmes Tcl). Un service de transcodage de vidéo a été ainsi réalisé comme un service actif.

Une autre architecture dans cette approche a été proposée par M. Fry dans [60]. Le système ALAN (*Application Level Active Networking*) est composé d'un ensemble de serveurs - *Dynamic Proxy Servers* (DPS) qui se trouvent dans des points stratégiques dans le réseau. DPS permet d'améliorer ou étendre les fonctionalité de l'Internet actuel. Il est possible de télécharger des services ou des entités de protocoles appelés *Proxylet* dans des DPSs. La communication avec un DPS se fait par l'appel à distant RMI car le système est implémenté sous Java. Le DPS accepte les commandes suivantes :

- `Load` : pour charger un Proxylet dans le DPS.
- `Run` : pour exécuter un Proxylet.
- `Modify` : pour modifier le fonctionnement d'un Proxylet.
- `Stop` : pour arrêter un Proxylet.

Les auteurs ont aussi proposé un ensemble de protocoles dynamiques qui peuvent être utiles pour plusieurs Proxylet. Par conséquent, il n'est pas nécessaire de tout implémenter dans un Proxylet. Les Proxylets peuvent partager des piles de protocoles : par exemple la

pile du protocole RTP.

Processeurs de réseau (*Network Processors*)

Une autre nouvelle approche est basée sur l'intégration de processeurs appelés NP (*Network Processors*) [61] [62] spécialisés dans le traitement des paquets dans les interfaces (cartes) réseaux. Cette approche est suivie par des grands constructeurs comme Intel et IBM. Les systèmes d'exploitation et les outils de compilation pour ces processeurs sont aussi développés pour déployer des codes actifs dans les cartes réseau. On obtient ainsi une amélioration de performance, parce que les paquets sont traités dans les cartes réseau sans passer au processeur central des routeurs. La figure 2.4 illustre cette approche.

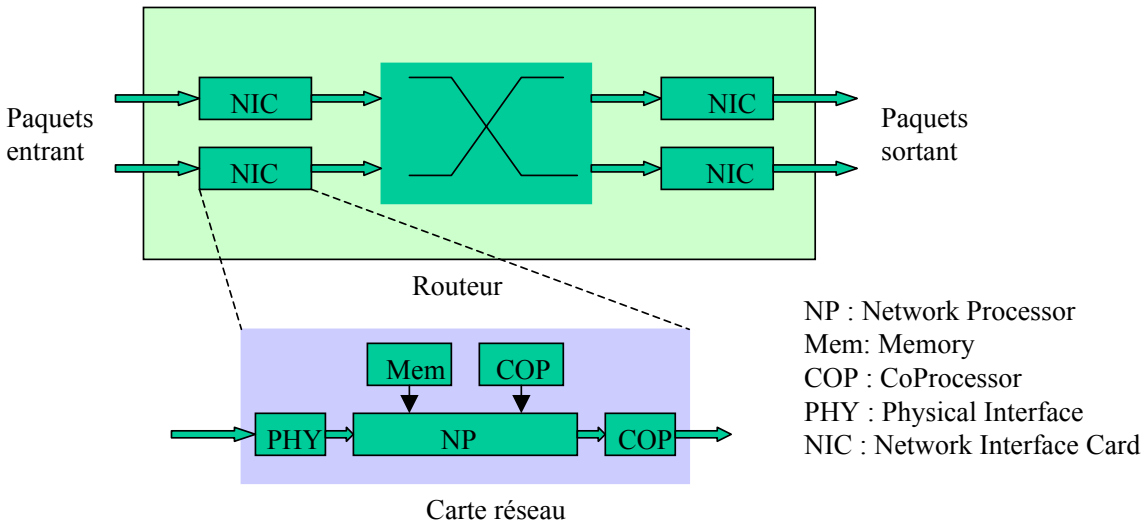

FIG. 2.4 – Architecture d'un processeur de réseau

Les co-processeurs aident les processeurs de réseau pour certaines tâches comme la recherche du plus long préfixe (*longest-prefix lookup*), la classification des paquets, la fonction de hachage, le calcul du checksum, l'accès à la mémoire etc. Plusieurs processeurs de réseau peuvent être reliés pour former une chaîne de traitement. L'architecture peut être pipeline où parallèle. Dans le modèle parallèle, chaque fil d'exécution (*thread*) dans chaque processeur reçoit un paquet différent et exécute le programme entier sur ce paquet. Dans le modèle de pipeline, chaque processeur reçoit chaque paquet et exécute une portion de code pour ce paquet. L'architecture de l'Intel IXP [63] est principalement basée sur le modèle de pipeline tandis que l'architecture de l'IBM PowerNP [64] est basée sur le modèle parallèle. Le processeur IXP2800 de l'Intel est capable de traiter un débit de 10 Gb/s ou 28 millions de

paquets par seconde. Néanmoins, pour l'instant il n'y a pas de standard de l'interface de programmation pour ces processeurs et chaque type de processeur est dédié à une application spécifique.

2.4 Conclusion

Nous avons présenté dans ce chapitre l'évolution des réseaux actifs à travers plusieurs projets de recherche. On peut constater qu'il n'y a pas une architecture commune pour tous les types d'applications. Il y a de plus en plus de travaux pour appliquer l'idée des réseaux actifs dans différents domaines comme les réseaux ad-hoc, les réseaux des sondes (*sensor networks*) etc. Cet état de l'art n'est pas exhaustif, mais il a essayé de présenter les principales applications. En dépit du grand nombre de travaux qui leur sont consacrés, il faut admettre que les réseaux actifs n'ont pas encore pu démontrer leurs advantages par rapport à d'autres alternatives. Ils n'ont pas non plus connu de grand succès auprès des constructeurs de routeurs comme Cisco ou Alcatel.

Deuxième partie :

Contributions

Chapitre 3

ProAN : une passerelle active générique

3.1 Motivation

Même s'il existe beaucoup de plate-formes pour exécuter des services actifs dans les éléments du réseau, la programmation des services qui travaillent sur des données de différents niveaux de protocole reste encore difficile. Les services actifs sont souvent programmés dans des langages comme Java (ALAN [60] ou PAN [65]), C (Router Plugins [47], LARA++ [14]) ou TCL (AS1 [59]). Les codes d'analyse et de formattage des PDUs, qui rencontrent souvent beaucoup d'erreurs et demandent beaucoup de temps de test, sont combinés avec le code de traitement des données. Par conséquent quand la structure de données ou de PDU est changée, tout doit être recompilé, rechargé, et ces codes sont parfois complexes et difficiles à maintenir et développer (beaucoup d'efforts pour tester, corriger des fautes et des erreurs sont nécessaires). En fait, les utilisateurs ou les programmeurs sont seulement intéressés à spécifier comment les données doivent être traitées.

Nous proposons un environnement d'exécution de *GateScript* avec son langage de programmation pour programmer les services actifs. Cet environnement aide les programmeurs des services actifs en automatisant les tâches d'interprêtation/construction des PDUs. Plus concrètement, l'analyseur et le générateur des PDUs d'un service actif sont générés automatiquement à partir des fichiers de description de PDUs. Le langage de script de *GateScript* offre aux programmeurs des services actifs la possibilité de spécifier les opérations sur les données aux différents niveaux de protocole de PDUs. Il fournit des variables prédéfinies concernant un protocole donné, et une bibliothèque extensible des fonctions travaillant sur le contenu des données. Quand les valeurs de ces variables sont disponibles grâce à l'analyseur de PDU, elles sont passées au programme de script de *GateScript* pour les modifier ou exécuter des actions. Toutes les modifications de données sont automatiquement prises en compte par le générateur de PDU pour construire la PDU désirée. L'environnement *GateScript* supporte

aussi des moniteurs de l'environnement qui surveillent l'état de l'environnement et informent ensuite les services actifs des changements via des événements.

Nous avons conçu et mis en œuvre ProAN, une passerelle active générique supportant l'environnement d'exécution *GateScript*. Une telle passerelle, en plus des fonctionnalités standards d'un nœud actif, devrait offrir :

- la possibilité de supporter différents environnements d'exécution pour les services actifs
- l'association des services actifs avec des flots de données choisis, faite dynamiquement par les services eux-mêmes
- des moniteurs capables de détecter des conditions variables dans l'environnement (réseaux, passerelles actives, services, dispositifs, utilisateurs),
- un mécanisme de communication asynchrone qui permet à des moniteurs d'informer des services actifs,

3.2 ProAN

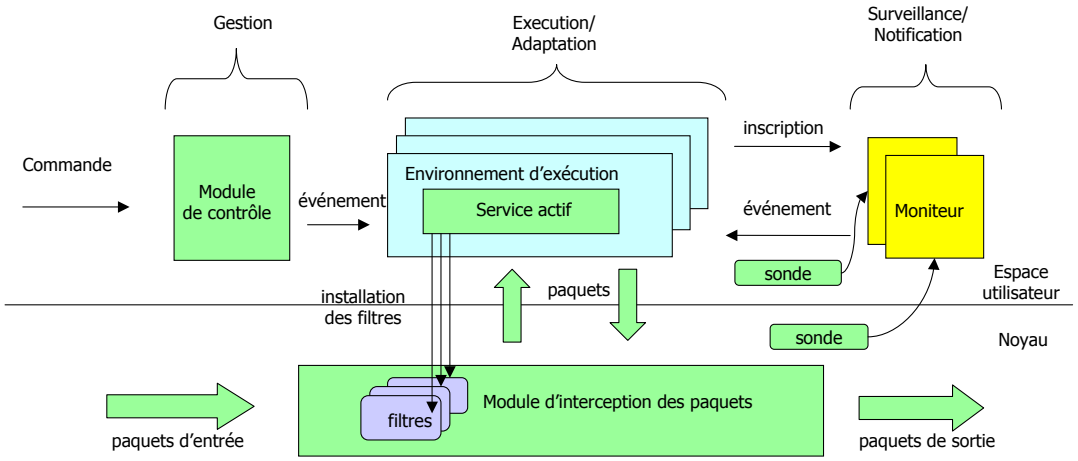

FIG. 3.1 – Architecture de ProAN

La figure 3.1 présente une vue générale de ProAN. L'architecture fournit deux niveaux de base d'abstraction. La couche inférieure prend soin de l'expédition des paquets. La couche supérieure fournit le support pour l'exécution des services actifs, l'adaptation et la surveillance. L'architecture inclut les éléments suivants :

- **Module d'interception des paquets** : Les paquets d'entrée passent par ce module qui laisse installer et désinstaller dynamiquement des filtres chargés d'intercepter des paquets et de les passer aux services actifs. Les paquets qui ne satisfont aucun filtre sont expédiés de manière standard aux sorties.
- **Environnements d'exécution** : un environnement d'exécution doit fournir des fonctionalités qui permettent aux services actifs écrits en un langage de programmation spécifique de s'exécuter sur la passerelle.
- **Services actifs** : Les services actifs sont des modules de code téléchargés dynamiquement pour augmenter les fonctionnalités du réseau afin de traiter des flots de données. Ils sont développés par un tiers de confiance, déployés et activés par des utilisateurs ou d'autres applications après une procédure d'authentification. Un service activé installe dynamiquement un filtre de paquets pour recevoir les paquets choisis selon des critères impliquant n'importe quelle portion de paquet IP, comme les adresses IP, le type de protocole, les numéros de ports au niveau de transport, ou même des données au niveau d'application. Il peut ensuite analyser ces paquets (par exemple, analyser le contenu), les modifier et les ré-injecter dans le noyau. Quand le service n'a plus besoin de recevoir des paquets, il désinstalle le filtre pour que les paquets passent par la passerelle sans rencontrer de délais additionnels.
- **Module de contrôle** (le contrôleur de la passerelle) : Le module de contrôle s'occupe de l'authentification des administrateurs, des utilisateurs et des services. Il active ou suspend les services selon des demandes d'utilisateurs ou d'autres applications. Les utilisateurs peuvent passer des paramètres aux services activés grâce à ce module qui informera à son tour les services concernés au travers des événements. Cela évite aux utilisateurs de s'authentifier deux fois : la première fois quand ils demandent d'activer un service et la deuxième fois lors qu'ils envoient une commande à un service activé. Ce module contrôle également les ressources de la passerelle : unité centrale, mémoire, stockage etc.
- **Moniteurs** : Pour devenir réactif vis à vis de l'état de l'environnement, un service souscrit à un ensemble de services spécialisés appelés des moniteurs capables de recueillir des événements intéressant le service actif, et de les lui transmettre pour lui permettre de réagir de manière appropriée. Les moniteurs peuvent détecter par exemple la congestion sur un lien spécifique, les états dégradés d'un lien sans fil, l'espace de stockage insuffisant, la gique accrue pendant un certain temps, un dispositif mobile débranché, et beaucoup d'autres informations liées à l'état du réseau, du nœud actif, des dispositifs eux-mêmes (état de batterie ou d'autres ressources) ou de l'utilisateur (par exemple son état de santé). De cette façon, un service actif peut installer un filtre de paquets et traiter les paquets quand nécessaire.

Le fait de laisser les services actifs effectuer des actions vis à vis d'un tel événement

sort les utilisateurs de la boucle d'interaction avec l'application. Ainsi, dans beaucoup de situations, le service actif peut réagir plus rapidement que l'utilisateur humain pour prendre les bonnes actions.

Dans ProAN, l'interception des paquets se fait par le service actif lui-même. C'est à dire que l'injection et l'annulation des filtres de paquets se fait au bon moment par le service. Dans certaines architectures, quand l'utilisateur demande d'activer un service actif, le module de contrôle du nœud actif installe tout de suite le filtre et les paquets sont toujours passés au service actif, même si, dans certains cas, ce dernier n'a pas besoin de traiter les paquets.

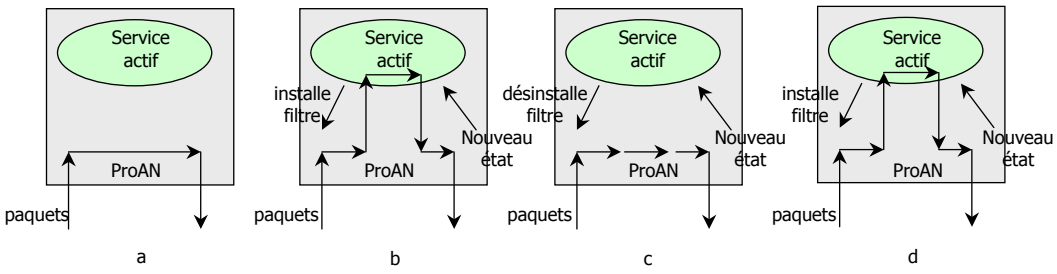

FIG. 3.2 – Comportement d'un service actif

La figure 3.2 illustre le comportement dynamique d'un service actif. Supposons que, lorsque le service actif est activé, l'état de l'environnement soit tel qu'aucun traitement ne soit exigé. Le service est endormi (bloqué), et les paquets sont envoyés sans passer par le service. Les ressources de la passerelle sont dédiées à l'envoi des paquets, et non à passer les paquets au service (figure 3.2.a). Quand le moniteur détecte un changement de l'état du réseau, par exemple, la congestion sur un lien, un événement est envoyé au service. Le service se réveille et installe un filtre approprié pour intercepter les paquets pour le traitement (figure

3.2.b). Quand l'état change à la nouvelle fois (la congestion disparaît), un autre événement est envoyé au service qui désinstallera le filtre de paquets. Les paquets ne sont plus interceptées. Le service retourne dans l'état bloqué (figure 3.2.c). Ce processus peut ensuite se répéter (figure 3.2.d).

Cette manière d'opérer évite de passer des paquets au service actif quand cela n'est pas nécessaire. Cette approche diffère d'autres architectures dans lesquelles un module de contrôle installe un filtre dès l'activation d'un service actif de sorte que le service reçoive des paquets même s'il n'effectue pas de traitement utile sur celles-ci. Nous tirons deux avantages de ce mode de fonctionnement : la passerelle active peut épargner des ressources, et les flots de données n'encourent pas de pénalité quand il n'y a aucun besoin de traiter les données.

3.3 Implémentation de ProAN sous Linux

Nous avons implémenté ProAN sous Linux. Linux est un bon candidat pour un tel nœud en raison de ses propriétés intéressantes : le support de l'expédition des paquets, le support des modules chargeables dans le noyau, et la facilité de modifier le comportement du noyau. Le module d'interception des paquets de ProAN est implémenté dans le noyau de Linux et tous les autres composants sont implémentés en tant que processus dans l'espace utilisateur. En particulier, chaque service actif est exécuté comme un processus séparé.

3.3.1 Module d'interception des paquets

Le module d'interception des paquets laisse installer et désinstaller dynamiquement des filtres de paquets dans le noyau. Depuis la version 2.4, le noyau Linux a une architecture ouverte de traitement des paquets appelée : *Netfilter* [73]. Elle permet aux utilisateurs d'insérer des modules de traitement dans le chemin d'expédition des paquets pour faire des opérations supplémentaires sur ces paquets. Netfilter permet aussi de passer des paquets à un processus dans l'espace d'utilisateur. Après le traitement, ces paquets sont reinjectés dans le noyau pour continuer leur chemins.

L'architecture de Netfilter est présentée dans la figure 3.3. Les paquets arrivant au noyau traversent le crochet[1] 1 appelé *PRE_ROUTING*. Après le crochet 1, les paquets vont au module de routage qui décide s'ils doivent être réexpédiés ou livrés à l'hôte local. Si les paquets sont destinés à l'hôte local, ils sont passés au crochet 2 appelé *INPUT*. Si ces paquets doivent être expédiés, ils vont au crochet 3 *FORWARD*, puis au crochet 4 *POST_ROUTING*. Les paquets envoyés à partir d'une application locale passent par le crochet 5 *OUTPUT* avant d'être passés au crochet 4.

Chaque crochet est associé à une ou plusieurs tables. Il y a trois types de tables dans

[1] point d'insertion de module

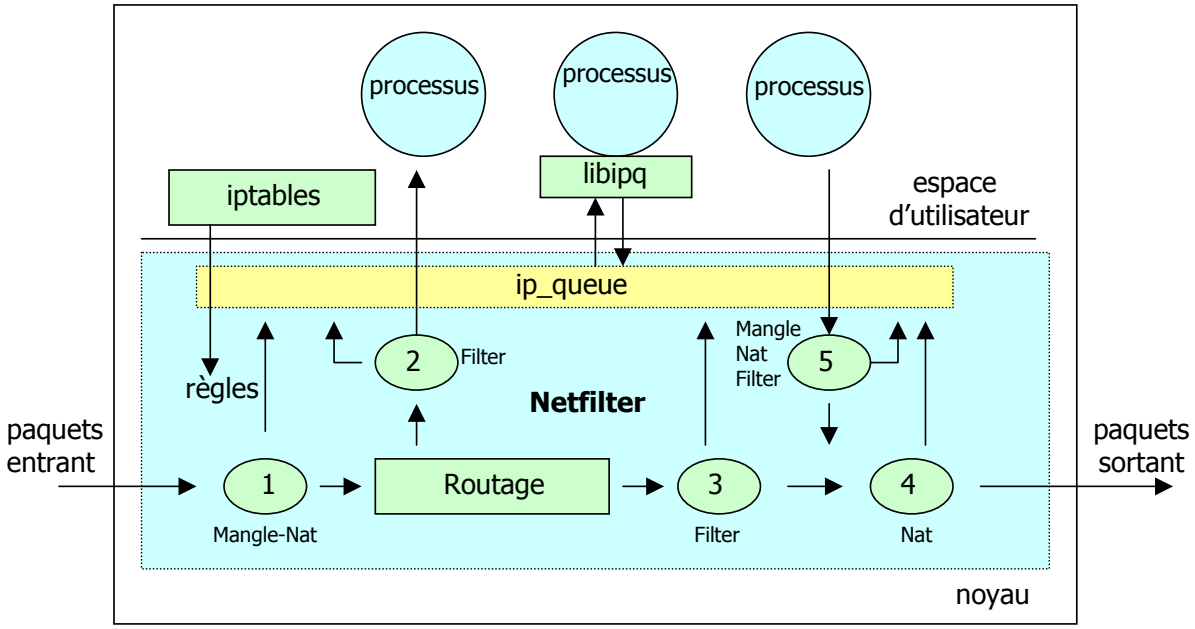

FIG. 3.3 – Architecture du Netfilter

Netfilter : *MANGLE*, *FILTER* et *NAT*. Par exemple, au crochet *PRE_ROUTING*, il y a deux tables : *MANGLE* et *NAT*.

À chaque crochet, nous pouvons insérer un ou plusieurs modules de traitement dans les tables associées. Ces modules seront exécutés quand un paquet traverse le crochet et satisfait le filtre de paquets qui leur est associé. Ces modules sont aussi appelés "des actions" du filtre correspondant. Une règle est la combinaison d'un filtre de paquet et de l'action à exécuter sur les paquets satisfaisant le filtre. Un filtre de paquets est la spécification d'un flot de paquets. La section 3.3.4 expliquera plus en détail cette spécification.

Chaque table accepte seulement un type de traitement. La table *MANGLE* est souvent utilisée pour modifier les paquets. *FILTER* est utilisée pour filtrer les paquets. Et enfin la table *NAT* est utilisée pour faire le service de translation d'adresse du réseau *(Network Address Translation)*.

Les modules de traitement sont libres de modifier les paquets et ils doivent retourner à Netfilter une des valeurs de verdict suivantes qui déterminent l'action correspondante sur le paquet :

- NF_ACCEPT : le paquet peut continuer son chemin.
- NF_DROP : le paquet doit être supprimé.
- NF_STOLEN : le paquet a été volé par le module ; ne pas continuer le traitement.
- NF_QUEUE : le paquet devrait aller au module `ip_queue` pour passer à un processus

dans l'espace d'utilisateur.
– NF_REPEAT : le module doit être exécuté encore une fois.

Netfilter peut passer des paquets à un processus dans l'espace d'utilisateur par le module *ip_queue* de n'importe quel crochet. Le processus utilisateur doit aussi retourner une des cinq valeurs de verdict au module *ip_queue* qui la passera ensuite au Netfilter. Le processus utilisateur emploie la bibliothèque *libipq* [81] pour communiquer (lire un paquet, envoyer des paramètres au noyau) avec le module *ip_queue* par les sockets de type `NETLINK` [90] qui sont un moyen de la communication entre des processus dans l'espace d'utilisateur et le noyau Linux.

Iptables [73] est un outil dans l'espace d'utilisateur qui permet aux utilisateurs de configurer Netfilter (ajouter, supprimer, remplacer les règles).

Par exemple, la commande suivante indique à Netfilter d'ajouter la règle "-d 129.88.38.101 -p tcp -j DROP" à la table "FILTER" du crochet "OUTPUT". Le flot de paquets TCP ayant l'adresse de destination 129.88.38.101 satisferait le filtre de paquets de cette règle et sera passé au module de traitement appelé "DROP" qui supprime simplement ces paquets :

```
iptables  -t filtre -A OUTPUT -d 129.88.38.101 -p tcp -j DROP
```

Voici la signification de ces options :

-t pour indiquer la table à laquelle la règle sera ajoutée.

-A pour ajouter (-D pour enlever, -I pour insérer etc.)

OUTPUT : le crochet

-d : l'adresse destinataire

-p : le protocole concerné

-j : l'action ou module qu'il faut faire ou appeler.

La commande suivante indique à Netfilter d'ajouter la règle "-p udp -j QUEUE" à la table "MANGLE" du crochet "PREROUTING". Les paquets UDP qui satisfont le filtre de paquets "-p udp" vont être passés au module *ip_queue* qui les envoie ensuite à un processus dans l'espace d'utilisateur.

```
iptables  -t mangle -A PREROUTING  -p udp -j QUEUE
```

Dans ProAN, chaque service actif est exécuté comme un processus utilisateur séparé. Par conséquent, il nous faut un mécanisme pour passer différents flots actifs aux différents

services actifs en même temps. Mais Netfilter, et particulièrement le module *ip_queue* sont limités pour les raisons suivantes :

- Un seul processus dans l'espace d'utilisateur peut recevoir des paquets du noyau.
- Dans l'*ip_queue*, un identificateur est affecté à chaque paquet. Si l'identificateur d'un paquet n'est pas dans la liste des identificateurs connus, ce paquet sera supprimé. Par conséquent, le processus dans l'espace d'utilisateur ne peut pas générer un nouveau paquet, il peut seulement modifier les données du paquet reçu. Ceci est très contraignant pour les services actifs qui veulent générer des nouveaux paquets.
- De plus, l'*ip_queue* garde toujours une copie du paquet passé à l'espace d'utilisateur dans le noyau. Cela peut saturer très vite le tampon du noyau, et les nouveaux paquets arrivants seront supprimés.

Le travail du projet *ipqmpd* [84] a essayé de régler le premier problème en ajoutant la possibilité de passer différents flots de paquets aux différents processus d'utilisateur. Il utilise un démon dans l'espace d'utilisateur appelé le multiplexeur d'*ip_queue*. Ce démon communique avec différents processus utilisateurs en utilisant d'autres mécanismes d'IPC (sockets). C'est inefficace, parce que les paquets doivent réentrer au noyau avant l'arrivée au processus utilisateur destinataire. De plus l'*ipqmpd* ne résoud pas les deux derniers problèmes indiqués ci-dessus.

Nous avons modifié le module *ip_queue* pour qu'il réponde à ces trois problèmes. La nouvelle version d'*ip_queue* permet de passer différents flots de paquets directement aux différents processus dans l'espace d'utilisateur sans passer par un démon multiplexeur. Ceci nous donne de bonnes performances.

Chaque flot destiné au module *ip_queue* est marqué par le PID (*process id*) du service actif qui a demandé ce flot. Grâce à ce PID, le module *ip_queue* peut envoyer ensuite le flot demandé au service actif correspondant (car chaque processus a un identificateur PID unique sous Linux).

Par exemple, les deux commandes suivantes permettent à Netfilter de passer les paquets UDP à un service actif ayant le PID 1931.

```
iptables -t mangle -A PREROUTING -p udp -j MARK --set-mark 1931
iptables -t mangle -A PREROUTING -m 1931 -j QUEUE
```

La première règle avec l'action "MARK" qui marque avec la valeur 1931 tous les paquets UDP. La deuxième règle avec l'action "QUEUE" qui indique à Netfilter de passer ces paquets UDP déjà marqués au module *ip_queue* qui les passera ensuite au service actif ayant la valeur du PID de 1931.

En combinant les deux actions "MARK" et "QUEUE", nous avons construit une nouvelle action appelée "MARK_AND_QUEUE" qui regroupe les fonctionnalités de ces deux dernières actions. Le module de traitement représentant la nouvelle action "MARK_AND_QUEUE"

marque aussi les paquets mais au lieu de retourner la valeur de verdict "NF_ACCEPT" à Netfilter comme l'action "MARK", il retourne la valeur "NF_QUEUE" qui indique à Netfilter de passer toute de suite les paquets marqués au module *ip_queue*. Avec un changement simple, les deux commandes ci-dessus sont réduites à une seule commande suivante :

```
iptables -t mangle -A PREROUTING -p udp -j MARK_AND_QUEUE
--set-mark 1931
```

Donc, une seule règle "-p udp -j MARK_AND_QUEUE –set-mark 1931" peut dire à Netfilter de passer les paquets UDP au service actif dont le PID est de 1931.

Notre module d'*ip_queue* a trois modes de fonctionnement. La table 3.3.1 récapitule les modes originaux et nouveaux.

Mode	Comportement	Version
IPQ_COPY_NONE	Mode initiale, packets sont écartés	Originale
IPQ_COPY_META	Copier le méta-donnée (adresse, port etc.) du paquet au service actif ; garder une copie au noyau	Originale
IPQ_COPY_PACKET	Copier les méta-données et les données du paquet au service actif et garder une copie au noyau	Originale
IPQ_PASS_PACKET	Passer les méta-données et les données du paquet au service actif sans garder aucune copie au noyau	Nouvelle

TAB. 3.1 – Modes dans le nouveau module *ip_queue*

Les services actifs exigent souvent le mode *IPQ_PASS_PACKET* qui permet de passer des paquets entiers à l'espace d'utilisateur sans en garder une copie dans le noyau. Quand un paquet est passé à un service actif dans l'espace d'utilisateur dans ce mode, le service emploie une nouvelle valeur de verdict appelée *NF_INJECT* (définie dans notre version d'*ip_queue*) pour injecter le paquet au noyau après traitement. Cette nouvelle valeur de verdict permet aussi à un service actif d'injecter un nouveau paquet dans le noyau (dans le cas de duplication d'un paquet ou de génération d'un nouveau paquet).

Par conséquent, le module d'interception de paquets de ProAN est la combinaison de Netfilter, nouveau module *ip_queue* avec le mode *IPQ_PASS_PACKET* par défaut, et module représentant l'action "MARK_AND_QUEUE".

De plus, nous constatons que Netfilter est plus ou moins modifié d'une version du noyau à une autre. Mais l'interface entre Netfilter et le module d'*ip_queue* reste inchangée. Par conséquent, notre module d'*ip_queue* est indépendant des versions du noyau Linux. Nous l'avons testé avec les versions 2.4.16 et 2.4.18 et n'avons eu aucun problème.

Notre module d'*ip_queue* soutient actuellement 40 files correspondant à 40 flots différents, pour autant de services actifs dans l'espace d'utilisateur.

3.3.2 Environnement d'exécution

La figure 3.4 illustre un des avantages de la propriété générique de ProAN qui est le support de plusieurs environnements d'exécution différents.

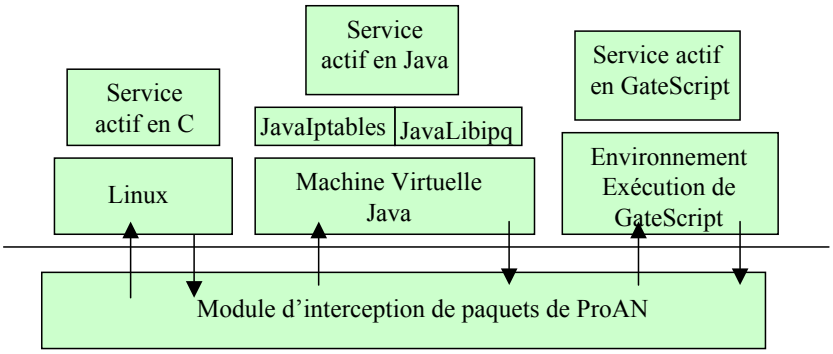

FIG. 3.4 – Environnements d'exécution sous ProAN

Pour l'instant, trois environnements d'exécution sont disponibles sous ProAN : Linux, Java et *GateScript*. Linux supporte les services actifs écrits en langage C ou C++. Ces services sont exécutés comme des processus dans l'espace utilisateur de ProAN.

Les services actifs écrits en Java sont exécutés par les machines virtuelles Java. Nous avons développé une interface JNI (*Java Native Interface*) et une librairie en Java appelée *JavaLibipq* qui permet aux services actifs développés en Java d'employer la bibliothèque *libipq* pour s'interfacer avec le module d'interception des paquets au noyau (principalement avec le module *ip_queue*) de ProAN. Le code de cette librairie est partiellement présenté en annexe 8.3. De son côté, la librairie *JavaIptables* permet aux services en Java d'injecter des filtres de paquets dans le noyau.

L'environnement d'exécution de *GateScript* supporte les services actifs écrits en langage de script *GateScript*. Le chapitre 4 expliquera en détail cet environnement d'exécution.

3.3.3 Services actifs

Nous proposons d'utiliser *GateScript* - notre langage de script - pour écrire les services actifs. Le chapitre 4 décrit en détail les avantages de cette approche. Néanmoins, les services actifs peuvent aussi être programmés purement en C, C++ ou Java et peuvent être déployés sous ProAN dans l'environnement d'exécution Linux ou Java respectivement.

Nous décrivons ci-après le mécanisme général de communication entre les services actifs écrits en C pour l'environnement Linux et le module d'interception de paquets de ProAN dans le noyau.

Pour injecter ou enlever un filtre de paquets, un service actif utilise la librairie *ProAN-iptables* qui est l'outil *iptables* modifié pour s'interfacer avec Netfilter dans ProAN. Pour chaque injection d'un filtre de paquets, *ProAN-iptables* doit générer la commande suivant :

```
-t mangle -A PREROUTING {filtre de paquets} -j MARK_AND_QUEUE --set-mark PID
```

Cette commande indique à Netfilter d'ajouter le filtre de paquets dans la table "MANGLE" du crochet "PREROUTING" et l'action (spécifiée par l'option -j) de ce filtre est MARK_AND_QUEUE, qui va marquer tous les paquets satisfaisant le "filtre de paquets" avec la valeur PID (*Process ID*) du service actif et les passe au service par l'intermédiaire du module *ip_queue*.

Pour enlever un filtre de paquets, le module *ProAN-iptables* doit envoyer une autre commande avec l'option -D (*delete*) au lieu de -A (*add*) à Netfilter. Par exemple, cette commande va enlever la règle ayant le filtre de paquets spécifié de la liste des règles de Netfilter :

```
-t mangle -D PREROUTING {filtre de paquets} -j MARK_AND_QUEUE --set-mark PID
```

Pour recevoir et injecter des paquets traités dans le noyau, les services actifs utilisent la librairie appelé *libipq*. La figure 3.5 illustre ces interactions.

Comme chaque service est exécuté comme un processus séparé, nous avons choisi les sockets de type *AF_UNIX* comme moyen de communication entre les services actifs et l'extérieur (module de contrôle de ProAN ou moniteurs). Parce qu'un service actif a besoin de recevoir sans interruption des paquets du noyau et d'attendre des événements provenant des moniteurs et du module de contrôle de ProAN, les services actifs sont souvent implémentés en tant que processus à fils d'exécution multiples (*multi-threads*) ayant au moins deux fils. L'un est responsable de la communication, et les autres sont responsables des traitements des paquets. Chaque service actif a un nom unique. Nous utilisons aussi le PID d'un service actif avec son nom unique pour former le nom utilisé par le socket *AF_UNIX*. Par conséquent, chaque instance d'un service a un point de communication unique pour que les moniteurs et le module de contrôle de ProAN puissent distinguer des instances différentes d'un même service, activées par des utilisateurs différents.

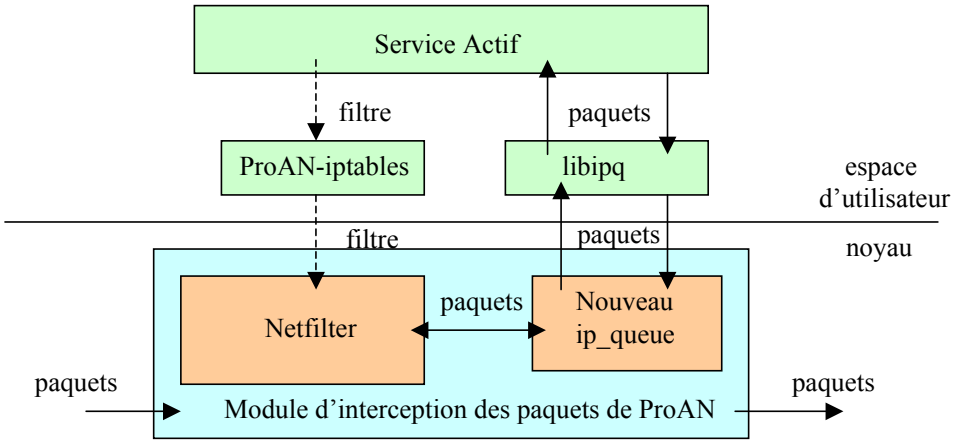

FIG. 3.5 – Service actif sous ProAN

3.3.4 Filtre de paquets

Un filtre de paquets est une spécification d'un flot de paquets basée sur les informations concernant n'importe quelle portion des paquets IP, comme les adresses IP, le type de protocole, les numéros de port au niveau de transport, ou même des données au niveau application.

Si un nouveau paquet satisfait un filtre de paquet, il est remonté au service propriétaire de ce filtre.

Les adresses source (indiquée par les options '-s' ou '–source')ou destination (les options '-d' ou '–destination') peuvent être spécifiées de quatres manières :

- la plus commune est d'utiliser le nom complet, par exemple :"www.cnn.com" ou "zanzibar.imag.fr".
- la deuxième est d'utiliser l'adresse IP complete comme : '129.88.38.101'. Par exemple le filtre "-d 129.88.38.101" correspond à tous les paquets ayant '129.88.38.101' come l'adresse de destination.
- la troisième et la quatrième sont d'utiliser des plages d'adresses comme : '129.88.38.0/24' ou '129.88.38.0/255.255.255.0'.

Le protocole est spécifié avec l'option '-p' ou '–protocol'. Le protocole peut être un numéro

ou un nom dans quelques cas spéciaux comme 'TCP'[2], 'UDP' ou 'ICMP'.

L'inversion (spécifiée par '!') est possible pour les options comme '-s','-d' ou '-p'. Par exemple le filtre "-s !129.88.38.101" correspond aux paquets dont l'adresse source est différente de '129.88.38.101'.

Après l'option "-p TCP" ou "-p UDP", on peut ensuite spécifier les numéros de port par les options '–source-port' ou '–sport' pour les ports de source et '–destination-port' ou '–dport' pour les ports de destination.

Les paquets longs sont souvent fragmentés en plusieurs paquets pour pouvoir être envoyés sur un lien. Les fragments sont regroupés à la destination. Le problème est que le premier fragment contient toutes les informations comme les entêtes IP, TCP, UDP ou ICMP. Mais les fragments suivants contiennent seulement les entêtes IP. Par conséquent, ils ne sont pas détectés par des filtres ayant des options du niveau de transport comme le numéro de TCP ou UDP. Mais nous pouvons spécifier un filtre pour tous les fragments en utilisant l'option '-f'. Par exemple le filtre "-f -d 192.168.1.1 -p TCP" détectera tous les paquets (avec les fragments) de TCP ayant '192.168.1.1' comme l'adresse destination.

Les autres options concernant les entêtes IP ou le niveau transport (TCP, UDP), sont précisées dans le document de Netfilter [73].

En profitant des extensions de Netfilter, le module d'interception des paquets de ProAN nous permet aussi de spécifier des filtres concernant des informations du niveau application des paquets IP. Netfilter est extensible et on peut y ajouter de nouveaux modules spécifiquement pour tester si un paquet satisfait certaines conditions. Ces modules sont appelés "*module de test*" ou "*match*" en anglais. Un nouveau module de test est responsable de dire à Netfilter si un nouveau paquet arrivé satisfait ses conditions. Un module de test a aussi des options. On peut utiliser un nouveau module de test pour construire un nouveau filtre de paquets. Par exemple un nouveau module de test appelé "ttl" qui teste si la valeur du champs "ttl" (*time to live*) d'un nouveau paquet est supérieure à une valeur donnée peut contribuer à construire un nouveau filtre de paquets. Le filtre "–match ttl –ttl-gt 3" correspond aux paquets dont la valeur du champs ttl est supérieure à 3. L'option "–match" ou "-m" spécifie le nom d'un nouveau module de test. Les options suivantes (–ttl-gt[3]) sont spécifiques à ce nouveau module. Un autre exemple : le filtre "-d 192.168.1.1 –match ttl –ttl-gt 3" correspond aux paquets ayant "192.168.1.1" comme adresse destination, et une valeur du champs ttl supérieure à 3.

Par conséquent on peut développer des nouveaux modules de test qui inspectent des données de n'importe quel niveau protocole (y compris au niveau application) d'un paquet IP.

Un des nouveaux modules de test est connu sous le nom "*u32*" [74]. Ce module de test

[2]'tcp' est aussi possible

[3]–ttl-gt : ttl greater than

peut extraire 4 octets consécutifs d'un paquet IP à partir de n'importe quelle position. On peut ensuite appliquer un masque à ces 4 octets pour obtenir la valeur de n'importe quel bits ou octet de ces 4 octets. Cette valeur est ensuite comparée à une tranche de valeurs qu'on veut tester. Par conséquent, avec ce module de test, on peut déjà accéder à n'importe quelle donnée d'un paquet IP et tester si elle satisfait une condition.

Les filtres de paquets sont stockés en ordre dans une liste dans la table "MANGLE" du crochet "PREROUTING" au noyau. Les nouveaux filtres sont ajoutés à la fin de la liste. Un nouveau paquet arrivé est testé avec le premier filtre de paquets. S'il le satisfait, le paquet est passé au module MARK_AND_QUEUE pour être envoyé ensuite au service qui a injecté ce filtre. Après le traitement par le premier service, le paquet peut être réinjecté pour continuer son chemin - c'est à dire pour être testé avec le deuxième filtre. Si tous les filtres sont testés, mais que le nouveau paquet ne satisfait aucun filtre, il est routé normalement.

3.3.5 Module de contrôle de ProAN

Le module de contrôle est implémenté comme un processus démon qui traite les demandes d'activation de service des utilisateurs et d'autres applications. Il est responsable de l'authentification des utilisateurs par l'intermédiaire d'une base de données contenant les informations sur les comptes des utilisateurs.

Les utilisateurs et les applications utilisent le protocole de communication de ProAN décrit en section 3.3.6 pour activer un service.

Sur la demande d'un utilisateur pour l'activation d'un service, le module de contrôle recherche le service dans la base de données, vérifie les droits d'accès de l'utilisateur, et active le service en se dupliquant lui-même pour créer un nouveau processus fils. Il change l'UID (*effective UID*) du processus fils, qui a la valeur de 0 en l'identificateur de l'utilisateur - *user ID*[4]. (car le module de contrôle s'exécute en mode super-utilisateur). L'identificateur de l'utilisateur est utilisé pour la gestion de l'utilisation des ressources : services activés, quota de stockage, temps de CPU, etc.

Le processus fils remplace ensuite son image par le code de l'environnement d'exécution concernant le service actif demandé grâce à l'appel de la function "`execpv`". Le nom du service est passé comme un paramètre à l'environnement d'exécution qui charge ensuite le service pour exécution. Une commande d'activation de service contient souvent des paramètres pour le service. Le module de contrôle les passera au service à travers l'environnement d'exécution comme des paramètres de l'environnement. Dans le cas où il s'agit d'un service écrit en C pour l'environnement d'exécution Linux, le processus fils du module de contrôle remplace tout de suite son image par le code du service actif demandé.

Le service attend ensuite la capacité *CAP_NET_ADMIN* [83] qui sera envoyée par le

[4]Chaque utilisateur ayant un compte sous Linux a un unique identificateur appelé *UserID*.

module de contrôle. Le service actif exige une telle capacité pour pouvoir communiquer avec le module d'interception des paquets dans le noyau par le socket de type *NETLINK*.

Comme tous les services actifs sont des processus fils du module de contrôle, quand un processus de service meurt anormalement, un signal de type *SIGCHLD* sera envoyé au module de contrôle qui libère toutes les ressources et nettoie les filtres de paquets installés dans le noyau pour le service.

La figure 3.6 dans la section 3.3.7 illustre la communication entre le module de contrôle et les services actifs, ainsi que les moniteurs. À côté d'un point de communication avec l'extérieur basé sur les sockets de type *PF_INET* (TCP), le module de contrôle a aussi un deuxième point de communication locale pour communiquer avec les services actifs et les moniteurs. Ce point de communication locale est basé sur le socket de type *AF_UNIX*. L'implémentation actuelle de ce module a pris "*Module_Controle_ProAN*" comme nom par défaut pour construire ce socket. Par conséquent, les services actifs et les moniteurs connaissent toujours ce point de communication.

3.3.6 Protocole de communication de ProAN

Le protocole de communication de ProAN est utilisé par :

- les utilisateurs ou les applications pour activer un service actif ou un moniteur auprès du module de contrôle de ProAN.
- les services actifs pour s'inscrire ou se désinscrire auprès d'un moniteur,
- les moniteur ou le module de contrôle de ProAN pour envoyer les événements aux services actifs.

Ce protocole est textuel et de type "*commande - response*". Le syntaxe de la commande est la suivante :

```
COMMANDE " " Nom_de_Service_de_Moniteur_ou_Evenement "\r\n"
(paramètre ":" valeur "\r\n")*
"\r\n\r\n"

COMMANDE := "ACTIVER"|"CONFIGURER"|"ARRETER"|"TELECHARGER"
|"INSCRIRE"|"DESINSCRIRE|"EVENEMENT"
Nom_de_Service_de_Moniteur_ou_Evenement := <chaine de charactères>
paramètre := <chaine de charactères>
valeur := <chaine de charactère>
```

La table 3.2 présente l'utilisation de chaque commande.

La réponse est aussi très simple, et indique si la commande a été bien exécutée ou non.

Commande	Utilisation
ACTIVER	Pour activer un service ou un moniteur
CONFIGURER	Pour configurer un service
ARRETER	Pour arrêter un service ou un moniteur
TELECHARGER	Pour télécharger un service ou un moniteur d'un serveur distant
INSCRIRE	Utilisée par les services pour s'inscrire auprès d'un moniteur
DESINSCRIRE	Utilisée par les services pour desinscrire auprès d'un moniteur
EVENEMENT	Utilisée par les moniteurs ou le module de contrôle de ProAN pour envoyer un événement à un service

TAB. 3.2 – Commandes du protocole de communication de ProAN

```
("OK"|"ERREUR") "\r\n"
(paramètre ":" valeur "\r\n")*
"\r\n\r\n"
```

La liste des paramètres est ouverte et extensible pour toutes les commandes et les réponses. Voici quelques paramètres déjà définis pour la commande "ACTIVE" :

- *Utilisateur* qui indique le nom de l'utilisateur
- *Mot_de_Passe* : le mot de passe de l'utilisateur
- *Filtre_de_paquets* : le filtre de paquets pour le service

Par exemple la commande qui active le service TCP_SNOOP expliqué dans la section 4.6 :

```
ACTIVE TCP_SNOOP
Utilisateur : root
Mot_de_Passe : root_pass
Filtre_de_Paquets : -p TCP
```

Sous l'environnement d'exécution de *GateScript*, les paramètres sont disponibles comme des variables avec le même nom.

Le protocole d'authentification des utilisateurs est très important mais pour l'instant il n'est pas chiffré - les mots de passe circulent en claire sur le réseau. Nous laissons ce-point pour la suite du projet.

3.3.7 Moniteurs

Les moniteurs sont aussi des services sous ProAN. Ils sont responsables de surveiller l'état d'une ressource ou de l'environnement. Par exemple, l'usage du CPU, le quota de l'espace de stockage, le débit d'un lien etc. Chaque ressource a ses propres données et unités de mesures. Un service actif peut choisir de surveiller l'état d'une ressource en s'inscrivant auprès du moniteur approprié. Un moniteur peut servir plusieurs services en même temps. La communication entre un moniteur et un service est asynchrone et utilise aussi les sockets de type *AF_UNIX*. Comme chaque moniteur a aussi un nom unique, les services actifs peuvent connaître tout de suite le point de communication avec les moniteurs[5]. Comme les services actifs, les moniteurs sont exécutés en tant que processus à fils d'exécution multiples (*multi-threads*) ayant au moins deux fils. L'un est responsable de la communication et les autres sont responsables de la surveillance.

Un service peut utiliser un moniteur disponible sur ProAN ou activer un nouveau moniteur.

La figure 3.6 illustre la communication entre les services actifs, les moniteurs et le module de contrôle de ProAN.

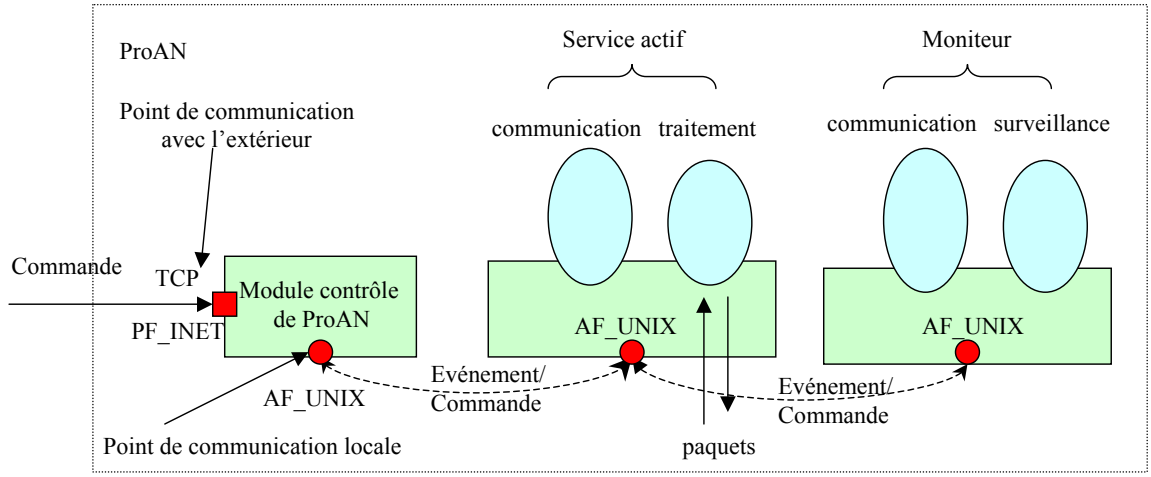

FIG. 3.6 – Communications entre les services actifs et les moniteurs

[5]Pour communiquer avec les sockets de type AF_UNIX, il suffit de connaître leur nom

Les services utilisent la commande "INSCRIRE" du protocole de communication de ProAN décrit en section 3.3.6 pour s'inscrire auprès d'un moniteur. Cette commande a le paramètre "*De :*" qui porte le nom du socket *AF_UNIX* du service, ainsi que les paramètres concernant la ressource à surveiller. Par exemple, la commande suivante inscrit le service de cache auprès du moniteur de déconnexion. La ressource à surveiller est une machine :

```
INSCRIRE  Moniteur_Deconnexion
De : Service_cache_1470

Hote : marie.imag.fr
```

Pour se désinscrire d'un moniteur, les services actifs utilisent la commande "DESINSCRIRE" .

```
DESINSCRIRE  Moniteur_Deconnexion
De : Service_cache_1470

Hote : marie.imag.fr
```

Nous avons développé les moniteurs suivants :

- une sonde RTCP qui peut détecter si la communication d'une session RTP est dégradée ou pas,
- un moniteur de présence qui détecte la présence d'un utilisateur devant son ordinateur en se basant sur les événements de la souris ou du clavier,
- un moniteur de déconnexion pour détecter la déconnexion d'une machine,
- un moniteur de débit pour surveiller le débit d'un lien,
- un moniteur d'usage CPU pour collecter le temps d'utilisation de l'unité centrale pour un service actif.

3.3.8 Événements

Chaque moniteur supporte un ensemble d'événements. Chaque événement a un nom unique et représente un changement de l'état de la ressource surveillée par le moniteur. Les paramètres accompagnés d'un événement donnent plus d'information sur le nouvel état de la ressource. Par exemple, l'événement suivant est envoyé au service de cache quand la machine surveillée est déconnectée :

```
EVENEMENT  Deconnexion
De : Moniteur_Deconnexion
Hôte : marie.imag.fr
Etat : Deconnexion
```

3.3.9 Sécurité

La sécurité est cruciale pour les réseaux actifs. Dans ProAN, chaque service est exécuté comme un processus séparé dans l'espace utilisateur. Ceci limite déjà les problèmes si un service meurt ou rencontre des erreurs dans le traitement des paquets. Les autres services voisins peuvent continuer à s'exécuter. Cela n'est pas possible dans le cas de *PromethOS* [56] où les services sont des modules injectés au noyau Linux. Un module erroné peut entraîner des conséquences graves pour le système.

Un autre problème est d'assurer qu'un service actif activé par un utilisateur ne puisse pas intercepter et modifier des flots de paquets d'autres utilisateurs. (Sauf pour l'administrateur qui peut tout intercepter). Nous résoudrons ce problème en mettant une contrainte sur le filtre de paquets qu'un service actif peut injecter au noyau. Le filtre de paquets doit avoir au moins l'option '-s' ('–source') ou '-d' ('–destination') qui spécifient l'adresse source ou destination. Une de ces adresses doit être l'adresse de la machine à partir de laquelle l'utilisateur (ou l'application) lance la commande d'activation du service. Cela est possible car le module de contrôle de ProAN qui traite toutes les commandes d'activation de service peut connaître l'adresse de cette machine. Et avant chaque injection, le module *ProAN-iptables* vérifie le filtre pour s'assurer au moins une bonne adresse source ou destination ait été incluse. Par conséquent, un service actif peut seulement intercepter les flots de paquets provenant de, ou destinés à la machine à partir de laquelle l'utilisateur du service l'a activé.

Comme les services et les moniteurs utilisent les sockets de type *AF_UNIX* pour la communication, ces services ne peuvent pas être la cible d'attaques de type "refus de service" comme dans les autres plate-formes comme ALAN ou AS1 où chaque service est un mini serveur qui attend les requêtes de l'extérieur. L'extérieur ne peut pas communiquer directement avec ces services, car les sockets de type *AF_UNIX* concernent seulement les communications locales. Dans ProAN, seul le module de contrôle est un tel serveur et est potentiellement la cible de ce type d'attaque.

Malgré tout, ProAN n'a pas encore résolu tous les problèmes de sécurité concernant les réseaux actifs comme le contrôle de ressource ou la garantie de la qualité de service. D'autres travaux doivent être menés sur ce sujet dans le futur.

3.3.10 IPv6 sous ProAN

Pour l'instant, ProAN supporte seulement les paquets IPv4. Mais nous disposons de suffisamment d'éléments pour que ProAN puisse supporter aussi les paquets IPv6.

Netfilter supporte les paquets IPv6. Nous avons aussi le module *ip6_queue* [82] pour passer les paquets IPv6 à l'espace utilisateur ainsi que l'outil *ip6tables* pour injecter des filtres de paquets concernant les paquets IPv6 à Netfilter.

3.3.11 Scénario d'application

Ce service de transcodage de vidéo démontre l'intérêt de laisser les services actifs choisir le bon moment pour intercepter des paquets : économiser les ressources de la passerelle quand le traitement des paquets n'est pas nécessaire. Un utilisateur mobile reçoit un flot de vidéo MPEG émis par un serveur de vidéo. Le fournisseur de réseau a déployé sur une passerelle active près du lien sans fil un service de transcodage. Au début, seul un moniteur de RTCP est activé sur la passerelle pour examiner la qualité de transmission. Dès le début le moniteur de RTCP injecte un filtre de paquets concernant seulement le flot RTCP pour intercepter les paquets RTCP contenant les informations de statistique envoyées par le terminal de l'utilisateur. Si la qualité de transmission est bonne, le flot RTP traverse la passerelle active sans intervention de celle-ci. Le service de transcodage est activé mais dors (bloqué), aucun filtre de paquet concernant le flot RTP n'est injecté dans le noyau. Par conséquent, les ressources de la passerelle sont réservées à l'acheminement des paquets, et aucune charge du CPU n'est générée pour passer des paquets de RTP au service de transcodage. Quand le moniteur de RTCP détecte une dégradation de la qualité de transmission (quand le client mobile est éloigné du point d'accès), il réveille le service de transcodage. Le service de transcodage injecte un filtre dans le noyau pour intercepter le flot de RTP, et commence la tâche de transcodage. La définition de la vidéo peut alors être réduite de moitié, ou encore le flot est transcodé dans le format H263 afin de réduire le trafic envoyé au client. Grâce à ce service, le client peut encore voir la vidéo. L'implémentation de ce service est décrit plus en détail en section 6.2.1.

3.4 Conclusion

Dans ce chapitre nous avons présenté l'architecture d'une passerelle active générique. Si on la compare avec d'autres architectures comme *Router Plugins* [47], *PromethOS* [56] ou ALAN [60] etc, l'architecture de ProAN présente les nouveaux concepts suivants :

- Elle est générique car elle peut supporter plusieurs environnements d'exécution différents.
- Les filtres de paquets sous ProAN peuvent impliquer n'importe quelle portion de données d'un paquet IP - du niveau réseau à celui d'application.
- Les services actifs sous ProAN peuvent aussi traiter des PDUs de n'importe quel niveau protocole : réseau, transport ou application
- Les services actifs sous ProAN décident eux-même du moment d'installer et désinstaller des filtres de paquets. Dans d'autres architectures ces filtres sont injectés dès l'activation du service et sont enlevés au moment de la fin du service. Ceci demande au nœud actif de passer le flot de paquets demandé au service, même dans le cas où aucun traitement n'est nécessaire.
- Dans d'autres architectures, la communication entre l'utilisateur et un service actif activé sur un nœud actif est directe. Dans ProAN cette communication se fait par l'intermédiaire du module de contrôle de la passerelle. C'est à ce module de contrôle

d'avertir, par des événements, le service actif des commandes envoyées par l'utilisateur. Par conséquent les services actifs n'ont plus besoin d'inclure le module d'authentification et de gestion d'utilisateurs dans le code.

Dans le chapitre suivant, nous présenterons un environnement d'exécution appelé *GateScript* et une approche générique pour écrire des services actifs sous ProAN.

Chapitre 4

Environnement d'exécution de GateScript

4.1 Motivation

Un service actif doit généralement traiter le contenu d'un flot des données et personnaliser le comportement d'un protocole que l'on veut enrichir. Il analyse, traite et formatte des PDUs. En général, ces fonctionnalités sont combinées dans un même code. Par conséquent quand la structure de données ou de PDU est changée, tout doit être recompilé, rechargé, et ces codes sont parfois complexes et difficiles à maintenir et développer (beaucoup d'efforts pour tester, corriger des fautes et des erreurs). Notre approche propose de séparer la fonction d'analyse et de formattage de celle de traitement des données. Nous proposons également d'employer un langage de description pour décrire la structure des PDUs et des données. Un compilateur est ensuite utilisé pour produire l'analyseur et le générateur des PDUs automatiquement à partir du fichier de description. Ce langage de description facilite la tâche de la programmation du service actif avec des flots de données. Chaque fois que la PDU change de structure (une option ou une entête est ajoutée) il suffit de recompiler le fichier de description des PDUs.

De plus, nous fournissons un langage de script générique appelé *GateScript* pour composer des services actifs, que ceux-ci traitent des paquets au niveau réseau, ou des PDUs au niveau application. Le langage *GateScript* supporte aussi les services proactifs en permettant de traiter les événements et les variables de l'environnement.

Pour atteindre notre but de séparer la fonction d'analyse et de formattage de celle de traitement des données, nous présentons dans la section suivante une architecture générique des services actifs pour l'environnement d'exécution *GateScript*.

4.2 Architecture générique des services actifs

Cette architecture vise à fournir des composants génériques des services actifs pour traiter les contenus de différents flots de données et adapter leur comportement. L'architecture peut être spécialisée pour un protocole donné, en se basant sur la description de la structure de PDU et sur la syntaxe du contenu de données. Le programme de script est dynamiquement chargé et associé au flot de données traversant le service actif. Il est exécuté chaque fois qu'une nouvelle PDU arrive. Ce script emploie des variables prédéfinies représentant les propriétés des PDUs.

La figure 4.1 présente cette architecture.

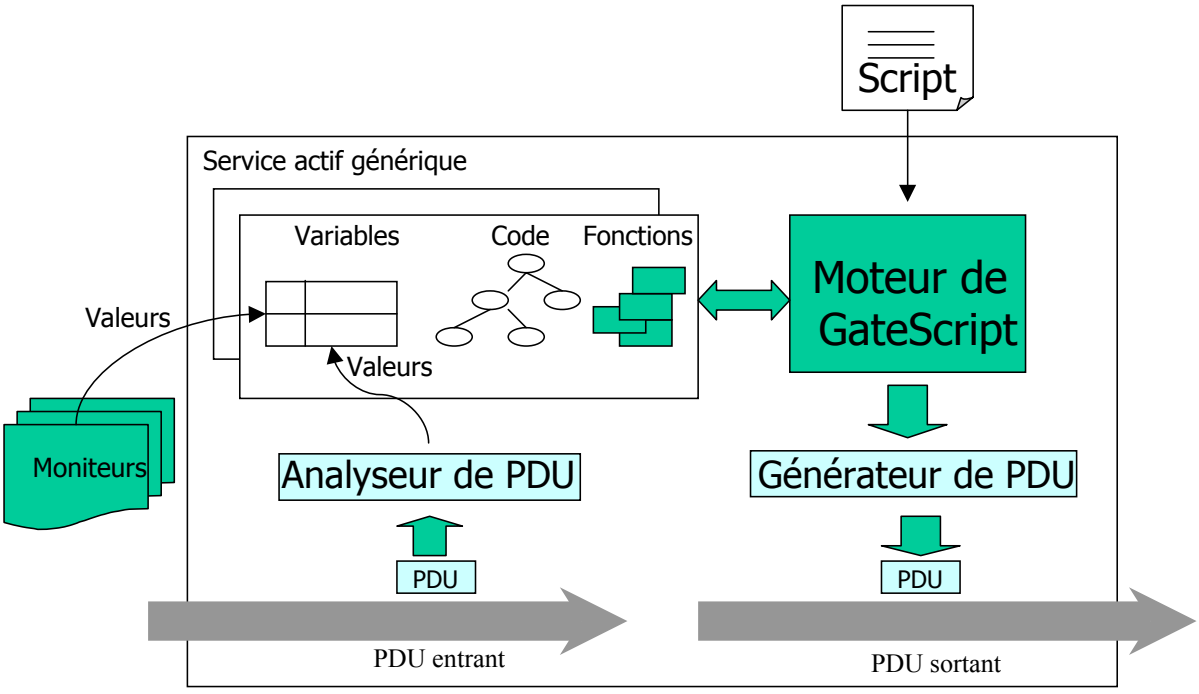

FIG. 4.1 – Architecture générique du service actif

Cette architecture se compose des éléments suivants :

- *un programme de script* écrit en langage *GateScript*. Ce programme utilise des variables prédéfinies représentant un protocole donné ou l'état de l'environnement. La section 4.4 présentera plus en détail ce langage de script.
- *un moteur d'exécution de* GateScript pour exécuter le script une fois que les variables concernant le protocole ont des valeurs affectées.

- *un analyseur de PDUs* pour reconnaître la structure des PDUs entrant, puis analyser leur contenus pour assigner les valeurs aux variables prédéfinies utilisées par le script.
- *un générateur de PDUs,* pour générer des PDUs à partir des valeurs des variables utilisées par le programme de script. L'analyseur et le générateur des PDUs sont automatiquement générés à partir d'un fichier de description du protocole.
- *les variables de protocole et de l'environnement.* Les variables de protocole représentent des champs des données dans des PDUs. Ces variables sont disponibles pour les programmes de script qui exécutent des codes en fonction de leurs valeurs. Les variables de protocole sont prédéfinies pour un protocole donné. Les variables de l'environnement sont fournies par les moniteurs de l'environnement et représentent l'état de l'environnement.
- *une librairie extensible* des fonctions utiles qui permettent au programme de script de faire les opérations demandées par le service actif.
- *des moniteurs de l'environnement* qui sont capables de détecter des changements des conditions dans l'environnement (réseaux, nœuds actifs, services, utilisateurs). *GateScript* supporte les services proactifs qui doivent réagir vis à vis de changements dans l'environnement. Un moniteur signale un changement des conditions au service proactif en envoyant un événement qui peut être testé dans le programme de script.

Cette architecture est générique et peut être spécialisée pour implémenter un service actif qui a besoin de modifier dynamiquement les contenus d'un protocole donné. Tous les éléments peuvent être modifiés et re-combinés pour donner de la flexibilité vis à vis des modifications de structure des PDUs. Un autre avantage de cette approche est l'usage d'un seul langage de script pour programmer des services actifs de n'importe quel niveau de protocole.

Nous analysons cette architecture dans le contexte d'un service de proxy HTTP qui fait le transcodage des données Webs (pages HTML, Images etc) pour des utilisateurs qui en ont besoin.

Dans ce service de proxy HTTP, les variables de protocole prédéfinies sont : `Content_Type`, `Content_Length`, etc qui représentent des en-têtes correspondantes dans le protocole HTTP. L'analyseur de PDU devient l'analyseur des requêtes et des réponses HTTP. Chaque fois que l'analyseur de réponse HTTP reçoit une réponse, il analyse la réponse pour extraire les valeurs des variables prédéfinies ci-dessus. Une fois terminée, ces valeurs sont passées au moteur d'exécution de *GateScript* qui interprète ensuite le programme de *GateScript*. Ce programme contrôle le comportement de ce proxy et contient des instructions concernant ces variables. Un exemple d'un programme *GateScript* est montré ci-dessous. Ce programme filtre toutes les images d'une page web en éliminant toutes les balises d'image dans la réponse s'il s'agit d'une page HTML. Si la réponse est une image, la deuxième instruction dans ce programme va éliminer le contenu de cette réponse. Une fois que l'exécution de ce programme *GateScript* est terminée, les variables prédéfinies prennent de nouvelles valeurs reflétant la modification souhaitée . Ces nouvelles valeurs sont passées au générateur de HTTP qui forme la nouvelle réponse et envoie cette réponse au client web.

```
if $Content_Type contains "text/html" then
```

```
   RemoveTag "img";
endif

if $Content_Type contains "image" then
   ContentDiscard;
endif
```

Dans cet exemple, les utilisateurs peuvent utiliser le langage de *GateScript* pour écrire un petit programme de script afin de personnaliser les réponses HTTP. Par exemple pour éliminer tous les publicités ou toutes les images quand un utilisateur accède à l'Internet à travers un lien faible débit, par exemple à la maison avec un modem ou à travers un lien GPRS. L'utilisateur n'a plus besoin de se préoccuper du code pour analyser le protocole HTTP.

4.3 Environnement d'exécution de GateScript

L'environnement d'exécution de *GateScript* fournit les composants standards suivants pour exécuter des services actifs écrits en langage *GateScript* selon l'architecture décrite en section 4.2 :

- le moteur d'exécution des services actifs écrits en langage de *GateScript*.
- des fonctions permettant aux programmes de *GateScript* d'injecter ou d'enlever des filtres dans le noyau de ProAN.
- un mécanisme permettant de charger des analyseurs ou des générateurs des PDUs de protocole et de les mettre en contact avec la librairie *JavaLibipq* pour recevoir et envoyer des paquets au noyau.
- un mécanisme permettant d'ajouter ou de remplacer des nouvelles fonctions à la librairie extensible du moteur d'exécution de *GateScript*.
- un mécanisme d'acheminement des événements envoyés par des moniteurs de l'environnement.
- un mécanisme de communication avec le module de contrôle de ProAN.

GateScript a été implémenté en Java. La figure 4.2 illustre cet environnement.

Il permet aux services actifs écrits en langage *GateScript* de charger facilement des analyseurs et des générateurs de PDUs. Il leur suffit d'indiquer les noms de deux classes dans le fichier de description de service, l'environnement de *GateScript* prend soin ensuite de les charger dans l'environnement et de les mettre en contact avec la librairie *JavaLibipq* pour recevoir et envoyer des paquets.

De plus, il fournit aux services actifs les fonctions pour injecter ou effacer des filtres des paquets dans le noyau. Ces deux fonctions importantes sont : `InjectPacketFilter` et `DeletePacketFilter` respectivement.

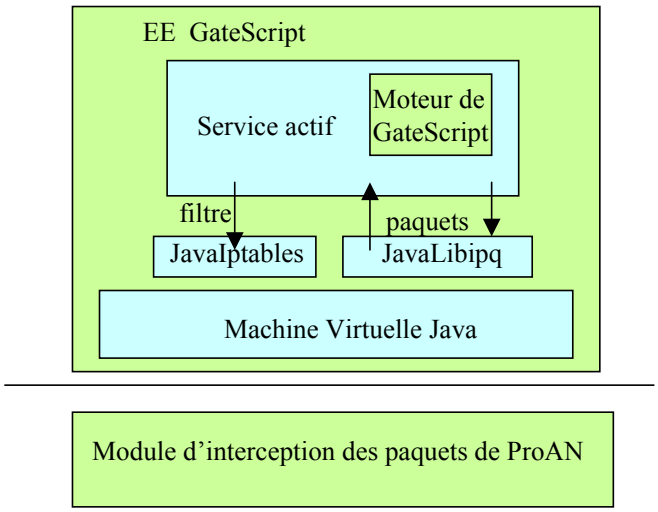

FIG. 4.2 – Environnement d'exécution de *GateScript*

Les paramètres accompagnés de la commande d'activation de service sont passés à l'environnement de *GateScript* par le module de contrôle de ProAN. Ils sont disponibles au programme de script sous forme de variables d'environnement que le programme peut consulter.

Une nouvelle fonction de traitement est intégrée à l'environnement d'exécution de *GateScript* si elle implémente l'interface *Function* de *GateScript*. Le nom de la classe implémentant une fonction doit être exactement le nom de la fonction. Voici l'extrait du code de cette interface en Java :

```
package GateScript;
import java.util.List;

public interface Function
{
public Value execute(List args, ProgramContext p)
                     throws GateScriptRuntimeError;
}
```

Cette interface a une seule méthode appelée "`execute`" qui accepte une liste d'arguments de la fonction et un contexte de l'environnement d'exécution. Quand une fonction est appelée en *Gatescript*, le moteur d'exécution de *GateScript* exécute cette méthode de la classe Java implémentant cette fonction et lui passe une liste d'arguments ainsi que le contexte

d'exécution du programme *GateScript*. Cette méthode peut tester la syntaxe des arguments et accéder à d'autres variables dans ce programme grâce à ce contexte d'exécution.

La valeur retournée est de type "`Value`" et peut être "`null`" si cette fonction ne retourne aucune valeur. Cette méthode peut générer une exception de type "`GateScriptRuntimeError`" si son exécution donne une erreur.

Par exemple la classe suivante implémente la fonction "`JpegScale`" de *GateScript* qui réduit la taille des images JPEG .

```
package GateScript;

import com.sun.image.codec.jpeg.*;
import java.awt.*;
import java.awt.image.*;
import java.io.*;
import java.util.*;

public class JpegScale implements Function {

public Value execute{List args, ProgramContext p)
             throws GateScriptRuntimeError
{
// le code de cette classe
...
...
}

}
```

Après la compilation de cette classe, il suffit d'ajouter le nom "*JpegScale*" dans le fichier de configuration des fonctions "*fonctions.conf*" pour que l'environnement *GateScript* puisse l'ajouter dans sa librairie extensible des fonctions de traitement.

4.4 Langage de script de GateScript

L'environnement d'exécution *GateScript* offre un langage de script permettant de programmer un service actif. Ce langage (*appelé "GateScript" par la suite*) fournit des variables prédéfinies concernant un protocole donné et une librairie extensible des fonctions travaillant sur le contenu des données. Comme c'est un langage de script, il permet aux utilisateurs d'écrire des programmes simples d'une manière presque déclarative - c'est à dire décrire ce qu'il faut faire mais pas comment il faut le faire.

Nous verrons ci-dessous les structures principales du langage *GateScript*. Une description plus formelle est décrite en annexe 8.4.

4.4.1 Instructions

Un programme *GateScript* se compose d'instructions. Chaque instruction peut examiner les valeurs des variables et appeler des fonctions appropriées. On peut accéder à la valeur d'une variable en préfixant son nom par le caractère dollar `$`.

Les variables définies par le programmeur sont déclarées en utilisant l'instruction `set` qui est utilisée aussi pour affecter une nouvelle valeur à une variable.

Il y a plusieurs types d'instructions :

- *instruction d'affectation*, e.g.
  ```
  set State $AckState;
  ```
- *appel de fonction*, e.g.
  ```
  WriteToCache $Ack_Number
  ```
  La fonction `WriteToCache` enregistre le paquet en question dans un fichier de cache dont le nom est une valeur par défaut. Ceci est détaillé dans l'exemple du service *TCP snoop* dans la section 4.6.
- *instruction conditionnelle*, e.g.
  ```
  if ($destination_address = $Client) then
     WriteToCache;
  endif
  ```
- *instruction d'événement* pour attendre une condition reliée à un événement et exécuter une instruction quand l'événement est reçu, e.g.
  ```
  onEvent $EventName = "Deconnexion" then
     InjectPacketFilter "-d $Hote";
  endEvent
  ```
  Quand un moniteur signale l'événement `Deconnexion`, le service exécute la fonction `InsertPacketFilter` qui injecte un filtre de paquet au noyau pour récupérer des paquets.

Le programme le plus simple de *GateScript* consiste seulement en un ensemble de fonctions ordonnées. Chaque fonction étant exécutée pour un traitement spécifique sur des données de PDU. Dans des cas plus complexes, le traitement de PDU peut dépendre des variables.

4.4.2 Variables

Dans *GateScript* il y a quatre types de variables : les variables de protocole concernant le protocole, les variables définies par le programmeur, les variables d'activation concernant les paramètres incluant la commande d'activation de service de l'utilisateur, et les variables représentant l'état de l'environnement :

- *Les variables de protocole* sont reliées à un protocole donné et représentent les champs d'en-tête de protocole, les propriétés, les éléments de contenu de PDU, ou des événements

signalés venant des moniteurs. Ces variables sont indiquées dans le fichier de description de la structure des PDUs du protocole. Le fichier de description des PDUs est utilisé pour générer l'analyseur et le générateur de ces PDUs. Ces variables de protocole sont automatiquement passées aux programmes de script par le moteur d'exécution de *GateScript*. Par exemple, si un programme de script travaille sur un flot TCP, il peut accéder à la variable `$window` correspondant à la fenêtre annoncée par le récepteur la variable `$SYN` représentant le drapeau de SYN TCP (voir l'annexe 8.5 pour la description des paquets TCP). Pour des protocoles textuels, considérons l'exemple du protocole de HTTP par lequel un programme de script aura l'accès aux variables `$Content_Type` et `$Content_Length` permettant un traitement intéressant basé sur le type et la taille d'un objet contenu, par exemple, dans une réponse de HTTP. L'analyseur de PDU assigne les valeurs identifiées dans une PDU à ces variables chaque fois qu'une nouvelle PDU arrive.

- *Les variables définies par le programmeur* doivent être déclarées avant leur utilisation. Par exemple la variable `$State` donnée dans l'exemple ci-dessus
- *Les variables d'activation* - quand l'utilisateur ou une application utilise le protocole d'activation de service de ProAN décrit à la section 3.3.6, la commande d'activation peut inclure des paramètres par défaut pour le service. Ces paramètres sont disponibles au programme de *GateScript* sous forme des variables. Par exemple la variable `$Hote` du service de cache dans l'exemple à la section 4.6.
- *Les variables d'état* - elle concerne l'état de l'environnement et sont définies par les moniteurs de l'environnement. Les programmes de *GateScript* peuvent consulter les valeurs de ces variables pour surveiller l'environnement d'exécution afin d'effectuer des actions appropriées. Par exemple la variable `$Etat` prendra la valeur "`Déconnexion`" si l'hôte surveillé par le moniteur de déconnexion devient déconnecté.

Des variables peuvent être combinées en employant des opérateurs pour former des expressions. Des appels de fonctions dans les expressions sont séparés des opérateurs par des crochets [ ] :

```
if ([CheckIfExistPacket $Ack_Number]) then
    ForwardFromCacheToClient $Ack_Number;
    return; endif
```

La fonction "`CheckIfExistPacket`" vérifie si le paquet demandé existe déjà dans le cache ou pas.

4.4.3 Événements

Quand un moniteur détecte une modification dans l'état de l'environnement, il le signale à un service par un événement. Chaque événement a un nom unique et une liste des variables. Considérons l'exemple suivant : un service souscrit à un moniteur de congestion qui détecte

des conditions de congestion dans le réseau et passe quelques informations sur les ressources disponibles :

```
onEvent $EventName = "Congestion" then
   AdaptCoding $AvailableBandwidth;
endEvent
```

Le moniteur signale l'événement `Congestion` et fournit la variable `$AvailableBandwidth` au programme *GateScript*. La variable donne la valeur courante de la largeur de bande disponible. La variable `$EventName` représente le nom de l'événement reçu.

4.4.4 Attributs statiques

Les instructions peuvent être statiques ou non-statiques. Une instruction statique est exécutée seulement une fois. Une telle sémantique d'exécution est nécessaire quand nous voulons initialiser des variables, lancer des moniteurs ou souscrire auprès des moniteurs déjà en exécution. Des instructions non statiques sont exécutées chaque fois qu'une PDU est reçue et analysée. Car, c'est le but des services actifs, les instructions ne sont pas statiques par défaut. Considérons l'exemple suivant :

```
if ($SYN = 1) then
   static set Client $destination_address;
   static set State $SynState;
endif
ForwardPacket;
```

Si le service reçoit un paquet SYN de TCP, il stocke l'adresse de destination dans la variable `$Client` et l'état actuel de la session dans la variable `$State`. La première et deuxième instruction seront exécutées seulement une fois, alors que la troisième (function `ForwardPacket`) sera exécutée chaque fois qu'un nouveau paquet est reçu.

4.5 Moteur d'exécution de GateScript

Les services écrits en *GateScript* sont compilés et stockés sous une forme intermédiaire d'un arbre et sont interprêtés par le moteur d'exécution de *GateScript* dès que les variables concernant des PDUs du protocole en question sont disponibles. Tous les éléments de l'environnement d'exécution *GateScript* se servent d'une structure contenant un ensemble de variables correspondant à une PDU appelée *le Contexte de PDU*. C'est une table de hachage avec toutes les variables reliées au protocole obtenues à partir de l'analyseur de PDU. Quand

l'analyseur de protocole reçoit une PDU, il l'analyse et crée un contexte de PDU. Le moteur d'exécution de *GateScript* l'utilise en exécutant le programme de script et le passe à n'importe quelle fonction appelée. Une fonction peut changer les valeurs des variables ou peut ajouter des variables au besoin. N'importe quelle fonction employée dans un programme de script doit être correctement programmée : des variables reliées au protocole doivent être changées d'une façon appropriée. Par exemple, si une fonction change la donnée d'une réponse de HTTP, elle doit également changer la variable `$Content_Length` de sorte que sa nouvelle valeur reflète la modification de donnée.

4.6 Exemples

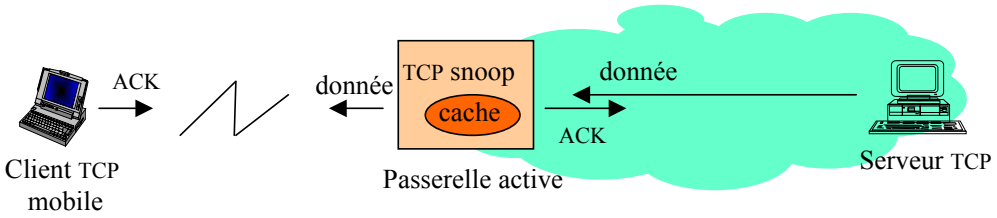

FIG. 4.3 – Service TCP snoop

La figure 4.3 illustre un service actif de *TCP snoop* [75]. Il fonctionne dans une passerelle active situé entre réseaux filaires et réseaux sans fil. Les liens sans fil représentent souvent des pertes des paquets. Un client TCP utilise les paquets contenant le champs ACK pour demander au serveur TCP de transmettre les données à partir du numéro de séquence spécifié pour ce champs. Un service *TCP snoop* a intérêt à cacher tous les paquets contenant les données envoyées par le serveur TCP pour répondre plus rapidement aux paquets ACK dupliqués d'un client mobile. Dans le cas où le paquet demandé n'est pas dans le cache, le paquet d'ACK est envoyé normalement au serveur.

Le programme *GateScript* suivant correspond à ce service

```
static InjectPacketFilter $Filtre_de_Paquets;

static set State 0;
static set SynState 1;
static set AckState 2;
static set EtablishedState 3;


if ($SYN = 1) then
   static set Client $destination_address;
   set static State $SynState;
endif

if ($SYN=1) and ($ACK =1) and ($State = $SynState) then
   set State $AckState;
endif

if ($State = $AckState) and ($ACK = 1) then
   set State $EtablishedState;
endif

if ($State = $EtablishedState) then
   if ($destination_address = $Client) then
      WriteToCache;
   endif

   if ($source_address = $Client) then
      if ([DuplicatedAck $Ack_Number]) then
         ForwardFromCacheToClient $Client $Ack_Number;
         return;
      endif
   endif
endif

ForwardPacket;
```

Au début, le service injecte le filtre "-p TCP" qui est disponible dans la variable `$Filtre_de_Paquets`. L'option "-p" signifie protocole. Cet filtre intercepte tous les paquets de TCP traversant la passerelle. La variable `$Filtre_de_Paquets` est un paramètre transmis au moment de l'activation de service. Ceci est expliqué dans la section 3.3.6.

Le service définit ensuite plusieurs variables pour représenter l'état d'une session de TCP : `$State, $SynState, $AckState` ou `$EtablishedState`. D'autres variables sont définies dans un paquet TCP : `$destination_address, $ACK, $SYN`, etc. Ces variables sont couplées avec un paquet IP/TCP entrant selon la description d'en-tête de TCP dans le langage Flavor [76] (cf. annexe 8.5).

Pour chaque paquet entré pendant l'étape d'ouverture de la session (*three-way handshake*), l'état est modifié. Quand la session est établie, le service met dans un cache tous les paquets provenant du serveur et les expédie à la destination. Quand il détecte qu'un paquet

provenant du client a le champ d'ACK correspondant à n'importe quel paquet stocké dans le cache, il l'expédie directement au client, et le paquet contenant ACK est abandonné. De cette façon, le client obtient rapidement un paquet retransmis par le service au lieu du serveur.

L'exemple suivant présente un service de cache pour un client mobile. Le service souscrit à un moniteur `$Moniteur_Deconnexion` qui contrôle la présence du client mobile en envoyant périodiquement les paquets d'écho ICMP. L'état du client est représenté dans la variable `$Disconnected` mise à jour par le moniteur. Quand l'état change, un événement est envoyé au service : `Deconnexion` ou `Connexion`. Sur la base de ces événements, le service injecte ou enlève le filtre de paquet dans le noyau. Au début, quand le client est présent, le service est activé mais bloqué en attendant un événement, et les paquets passent par le nœud actif sans traitement. Quand le moniteur détecte que le client est déconnecté, il signale le service qui injectera un filtre concernant l'adresse IP du client. De cette façon, le service commence à recevoir des paquets à stocker dans un cache. Quand le client apparaît à nouveau, le service lui envoie directement les paquets qui lui sont destinés, et le filtre de paquets est supprimé de sorte que des paquets ne soient plus passés au service. La protocole d'inscription auprès du moniteur est expliqué dans la section 3.3.7 et les événements sont expliqués dans la section 3.3.8.

```
static Moniteur_Deconnexion $Hote;

onEvent $EventName = "Deconnexion" then
   InjectPacketFilter "-d $Hote";
endEvent

onEvent $EventName = "Connexion" then
   DeletePacketFilter "-d $Hote";
endEvent

if $Etat = "Deconnexion" then
   WriteToCache;
else
   ForwardCacheToClient;
endif;
```

Le dernier exemple est relatif à un environnement pervasif comportant des capteurs. Nous présentons un service qui détecte l'élévation de la température et génère une alarme. D'abord il calibre les données brutes d'un capteur de température. Ensuite il teste pour détecter si la température est supérieure à un seuil prédéterminé. Si c'est le cas, il génère un événement aux services qui se sont inscrits auprès de ce capteur. Si la température est basse, le paquet sera éliminé. Nous considérons une structure simple de paquet avec deux champs : l'identificateur du capteur (`SensorID`) et la mesure brute de la température.

```
static InjectPacketFilter $Filtre_de_Packet;
```

```
static set FireAlarmThreshold 50;
set Temperature [Calibrate $RawMesurement];
if $Temperature > $FireAlarmThreshold then
GenerateEvent "FireAlarm" [GetLocalization $SensorID];
else
 DropPacket;
endif;
```

Un tel service peut être très utile pour un système d'information exploitant des mesures, car il évite que toutes les données venant des sondes n'encombrent le réseau.

4.7 Génération automatique de l'analyseur et du générateur des PDUs orientés bits

Nous employons le langage *Flavor* [76] pour décrire les structures des PDUs des protocoles orientés bits (*bitstream*). Flavor a été développé pour décrire le contenu des fichiers multimédia MPEG, JPEG, GIF etc, et il convient très bien à nos besoins. Les structures de données décrites dans Flavor sont compilées pour produire des classes C++ ou Java qui peuvent être employées pour analyser un bitstream pour identifier les champs définis et pour obtenir leurs valeurs.

La syntaxe d'un protocole est décrite dans un fichier qui est ensuite passé au compilateur de Flavor pour produire une classe Java ayant deux méthodes : l'analyseur et le générateur des PDUs. L'analyseur a pour but de lire une PDU, l'analyser pour extraire les valeurs de ses champs et assigner ces valeurs aux variables prédéfinies dans le fichier de description. Le générateur fait l'inverse. Il relit les valeurs de ces variables pour former une PDU complète. Nous illustrons cette procedure avec un exemple. Le fichier *Test.fl* suivant représente une simple PDU. Cette PDU tient sur un octet et contient deux champs : le premier est formé de deux bits et le deuxième de 6 bits. Ces deux champs sont nommés `a` et `b` dans le fichier de description *Test.fl*, et ils ont le type entier.

```
class Test {
    unsigned int(2) a;  // premier champ représente 2 bits
    unsigned int(6) b;  // deuxième champ représente 6 bits

};
```

Après la compilation avec le compilateur de Flavor, on obtient une classe Java avec le même nom que le fichier de description. Cette classe représente la PDU avec deux variables de type entier `a` et `b`. Elle a deux méthodes : la première avec le nom `get` qui est l'analyseur de PDU. L'argument de cette méthode est l'entrée des données représentée par la structure

propriétaire du Flavor : `Bitstream`. L'analyseur lit deux bits pour former un entier avant de l'assigner à la variable `a`. Les six bits suivants sont extraits pour former le deuxième entier qui est ensuite assigné à la deuxième variable `b`.

```
import flavor.*;
import java.io.*;

public class Test {
    int a;
    int b;

    public int get(Bitstream _F_bs) throws IOException {
        int _F_ret = 0;
        a = _F_bs.getbits(2);
        b = _F_bs.getbits(6);
        return _F_ret;
    }

    public int put(Bitstream _F_bs) throws IOException {
        int _F_ret = 0;
        _F_bs.putbits(a, 2);
        _F_bs.putbits(b, 6);
        return _F_ret;
    }
}
```

La deuxième méthode avec le nom `put` est le générateur de PDU qui fait l'inverse. Elle prend la valeur des deux variables `a` et `b` pour former un octet.

Pour atteindre notre but, c'est à dire obtenir *un contexte de PDU* [1], nous devons ajouter d'autres instructions au fichier de description de Flavor. Flavor présente une excellente propriété qui permet d'ajouter des instructions Java au fichier de description. Ces instructions vont être reproduites dans la classe résultat après la compilation. De plus, Flavor permet de guider le compilateur pour placer ces instructions à l'endroit exact où il faut les générer - c'est à dire dans la méthode `get` ou `put`.

Plus précisément, toutes les instructions de Java se trouvant entre `%.j{` et `%.j}` seront reproduites à l'endroit où elles sont mises. Les instructions se trouvant entre `%p.j{` et `%p.j}` seront reproduites dans la méthode `put`. Et enfin les instructions se trouvant entre `%g.j{` et `%g.j}` seront reproduites dans la méthode `get`. En utilisant cette propriété avec une bonne combinaison, nous pouvons obtenir ce contexte de PDU.

Concrètement dans l'exemple ci-dessus, nous ajoutons les instructions suivantes pour déclarer une variable `PDUContext` de type `Map` qui est une table de hachage en Java avant la déclaration des variables de PDU `a` et `b`.

[1] ou en d'autres termes - une table de hachage représentant toutes les variables de la PDU avec leur valeur à passer au moteur d'exécution de *GateScript*.

```
class Test {
   %.j{ Map PDUContext = new HashMap(); %.j}
   %.j{ Integer temps;          %.j}
```

Ensuite nous ajoutons l'instruction suivante après la déclaration de la variable `a`.

```
unsigned int(2) a;  // premier champ représente 2 bits
%g.j{ PDUContext.put("a",new Integer(a)); %g.j}
```

Cette instruction va être reproduite dans la méthode `get` - qui est l'analyseur de PDU - après l'instruction `_F_bs.getbits(2)`. Cette instruction convertit la valeur de `a` en `Integer` pour qu'elle puisse être ajoutée dans la table de hachage `PDUContext` sous le nom `a`.

On fait la même chose avec toutes les autres variables. Par conséquent on obtient le contexte d'une PDU sous la forme d'une table de hachage. Cette table va être passée au moteur d'exécution de *GateScript* pour interpréter le programme de *GateScript* sous forme d'une référence. Après l'exécution de ce programme, la même table représentera la nouvelle PDU déjà modifiée par le programme de *GateScript*.

Pour que ces nouvelles valeurs des variables représentant la PDU soient prises en compte dans la méthode `put`, qui est le générateur de PDU, on doit également ajouter d'autres instructions au fichier de description.

On ajoute les instructions suivantes avant la déclaration de la variable `a` :

```
%p.j{ temps = (Integer)PDUContext.get("a");
      a = temps.intValue(); %p.j}

unsigned int(2) a;  // premier champ représente 2 bits
```

Ces instructions seront mises avant l'instruction `_F_bs.putbits(a, 2) ;` dans la méthode `put` par le compilateur de Flavor. Cela permet de lire la valeur de la variable `a` dans la table de hachage avant de prendre 2 bits pour former la nouvelle PDU. Par conséquent la nouvelle valeur de la variable `a` sera prise en compte dans la construction de la PDU.

Nous présentons ci-dessous le nouveau fichier de description après avoir ajouté les instructions nécessaires de Java.

```
class Test {
    %.j{ Map PDUContext = new HashMap(); %.j}
    %.j{ Integer temps;          %.j}

    %p.j{ temps = (Integer)PDUContext.get("a");
      a = temps.intValue(); %p.j}
```

```
    unsigned int(2) a;  // premier champ représente 2 bits

    %g.j{ PDUContext.put("a",new Integer(a)); %g.j}

    %p.j{ temps = (Integer)PDUContext.get("b");
      b = temps.intValue; %p.j}

    unsigned int(6) b;  // deuxième champ représente 6 bits

    %g.j{ PDUContext.put("b",new Integer(b)); %g.j}

};
```

Et voici la classe Java générée représentant l'analyseur (méthode `get`) et le générateur (méthode `put`) de cette simple PDU :

```
import flavor.*;
import java.io.*;

public class Test {
    Map PDUContext = new HashMap();
    Integer temps;
    int a;
    int b;

    public int get(Bitstream _F_bs) throws IOException {
        int _F_ret = 0;
        a = _F_bs.getbits(2);
         PDUContext.put("a",new Integer(a));
        b = _F_bs.getbits(6);
         PDUContext.put("b",new Integer(b));
        return _F_ret;
    }

    public int put(Bitstream _F_bs) throws IOException {
        int _F_ret = 0;
         temps = (Integer)PDUContext.get("a");
      a = temps.intValue();
        _F_bs.putbits(a, 2);
         temps = (Integer)PDUContext.get("b");
      b = temps.intValue;
        _F_bs.putbits(b, 6);
        return _F_ret;
    }
}
```

Nous avons voulu que le processus d'ajout des instructions Java ci-dessus dans un fichier de description des PDUs soit automatique. Pour atteindre ce but, nous avons développé un compilateur qui ajoute ces instructions automatiquement.

Le langage Flavor lui-même présente un inconvénient si l'on veut utiliser pour décrire des PDUs : il a été développé pour décrire des formats de fichiers d'images (JPEG, GIF, TIFF etc.) ou de vidéo (MPEG, H263 etc.). Ces fichiers dits "sans-erreur" ou "*error-free*" présentent rarement des erreurs. Mais une PDU ou un paquet reçu directement du réseau peut contenir des erreurs. Les fichiers d'images ou de vidéo sont souvent prêts à fournir des données à l'analyseur généré par le compilateur de Flavor, mais un analyseur d'une PDU doit parfois attendre les données dans le réseau pour continuer ce travail. Nous avons donc modifié le code de la classe `Bitstream` dans Flavor pour qu'elle prenne en compte toutes ces contraints spécifiques à des PDUs dans le contexte du réseau.

4.8 Génération automatique de l'analyseur et du générateur des PDUs orientés textes

Pour les protocoles textuels comme SIP, HTTP, SMTP etc, nous produisons des analyseurs en employant le générateur de compilateur JavaCC [77]. Nous décrivons un protocole dans un fichier de description de syntaxe approprié à JavaCC.

Voici un extrait du fichier de description du protocole HTTP. Après chaque extraction de la valeur d'une variable, on la met dans une table de hachage qui représente le contexte du protocole. Les attributs d'en-tête qui seront disponibles aux programme *GateScript* sont définis par les variables dont les noms sont identiques aux attributs de HTTP. (En raison des problèmes de compatibilité syntaxique, nous remplaçons le tiret "-" par le soulignement "_"). Par exemple, l'attribut d'en-tête de `Content-Type` est représenté par la variable qui s'appelle `Content_Type`.

```
options { USER_CHAR_STREAM = true; }

PARSER_BEGIN (HTTPResponseParser)

public class HTTPResponseParser{

/* HTTP PDU context */

public Map PDUContext;

} PARSER_END (HTTPResponseParser)

void HTTPParse(): {} { {
 PDUContext = new HashMap();}
 Status_Line() <CRLF>
 ( Header() <CRLF> )* <CRLF>
 Message_Body()
}

void Status_Line() : { String version,reason_phrase;
```

```
int status_code; }
{ version = string()  <SPACE> {
PDUContext.put("version",version); }

status_code = number()<SPACE>
{ PDUContext.put("status_code", new Integer(status_code)); }

reason_phrase = String(); {
PDUContext.put("reason_phrase",reason_phrase); } }

void Header(): { String header,value; } {

header = string() ":" value = string() {

header = header.replace('-','_');
PDUContext.put(header,value); }
}

void Message_Body(): { byte[] data; } {
 data = byte_array()
{ PDUContext.put("content",data); } }
```

Dans ce fichier de JavaCC, nous avons aussi définit une variable représentant le contexte d'une réponse HTTP appelée `PDUContext` de type `Map` qui est une table de hachage en Java. Après chaque définition d'un élément quelconque d'une réponse HTTP, il y a toujours une indication en Java pour créer et ajouter la valeur de cet élément avec son nom dans la table de hachage.

Nous n'avons pas encore la méthode pour générer automatiquement des générateurs de PDUs textuelles. Ces générateurs sont actuellement développés manuellement.

Par exemple dans la définition de la procedure `Status_Line` qui représente la première ligne de la réponse HTTP, après l'extrait de l'élément "`version`", il y a l'indication `PDUContext.put("version",version) ;` en Java pour ajouter cette valeur dans la table de hachage (représentée par la variable `PDUContext`) avec le même nom "`version`".

```
void Status_Line() : {
String version,reason_phrase;
int status_code; }
{
version = string()  <SPACE>
 { PDUContext.put("version",version);}

status_code = number()<SPACE>
{ PDUContext.put("status_code",new Integer(status_code)); }
```

La même principe est appliquée pour les autres éléments d'une réponse HTTP.

4.9 Limites et avantages du langage de GateScript

Malgré sa simplicité, *GateScript* peut présenter encore des difficultés pour des utilisateurs qui n'ont aucune notion de programmation. Nous proposons donc une interface graphique à l'aide de laquelle ils peuvent cocher les options souhaitées qui sont ensuite traduites en un programme *GateScript* pour s'exécuter dans le service de proxy HTTP par exemple. Avec notre approche, de nouvelles options peuvent être facilement ajoutées quand de nouveaux en-têtes sont introduits.

Dans de nombreux cas, le traitement d'une réponse d'un protocole dépend de celui de la requête. Le langage de *GateScript* ne permet pas encore de synchroniser ces deux traitements.

Mais l'environnement d'exécution de *GateScript* permet non seulement d'injecter un programme *GateScript* pour exécuter des actions, mais aussi d'injecter un programme Java au lieu de *GateScript*.

Le programme Java peut aussi accéder aux variables reliées au protocole comme des programmes écrits en *GateScript*. En utilisant Java, on peut écrire des classes représentant des objets pour la synchronisation. Cela s'avère très intéressant pour les applications plus complexes comme un *Multiplexeur X* [106] qui réplique une session X sur différents serveurs X. Le protocole X est un protocole très complexe avec plus de 127 types de PDUs différents. Pour le développement du *Multiplexeur X*, nous avons utilisé le même environnement d'exécution et la même architecture. Au lieu d'utiliser le langage *GateScript* et le moteur d'exécution de *GateScript* pour programmer ce service, nous avons utilisé Java. La section 6.2.3 illustrera ce service.

4.10 Comparaison

Les recherches dans le domaine des réseaux actifs ont débouché sur plusieurs plates-formes soutenant des services actifs. Beaucoup d'entre elles utilisent les langages de programmation tels que Java (*ALAN* [60], *PAN* [65]), le langage C (*Router Plugins* [47], *LARA++* [66]), ou TCL (*AS1* [59]). Cependant nous pensons qu'un langage de script spécialisé avec la création automatique des analyseurs et générateurs de PDU comme *GateScript* est plus adapté pour programmer des services actifs. Quant à Java, nous le considérons comme un excellent langage pour développer des fonctions internes de *GateScript*, mais nous n'avons pas besoin de toute sa complexité pour programmer des services actifs, dans lesquelles par exemple, le programmeur devrait traiter des exceptions et tous les mots-clés de Java.

JACL [108], un nouvel interpréteur de TCL écrit entièrement en Java peut sembler être

un bon candidat pour programmer des services actifs, mais il recompile des programmes TCL à chaque interprétation tandis qu'un programme de *GateScript* est compilé seulement une fois au moment d'activation.

PLAN [69] et *GateScript* ont des objectifs différents : *PLAN* est un langage pour programmer des paquets actifs tandis que *GateScript* est employé pour programmer des services actifs qui traitent des paquets traditionnels (non actifs) de manière transparente.

Netscript [68] est un langage de composition pour composer des services actifs à partir de composants plus petits appelés les boîtes. La différence principale entre *Netscript* et *GateScript* est que *Netscript* convient pour composer des routeurs extensibles avec des piles dynamiques de protocoles, alors que *GateScript* est principalement employé pour personnaliser un service existant pour différents utilisateurs.

Le *Media Gateway Programmable* [79] emploie un langage de script pour programmer un passerelle de média. L'application vise seulement les flots de vidéo et non les structures de PDU d'un protocole quelconque.

4.11 Conclusion

Dans ce chapitre, nous avons présenté l'environnement d'exécution *GateScript* pour des services actifs sur ProAN et son langage de programmation. L'idée principale est de séparer la fonction d'analyse et de formattage des PDUs à celle de traitement des services actifs. Cela permet une flexibilité dans l'évolution des protocoles (PDUs). C'est à dire que, quand il y a une nouvelle options ou un nouvel en-tête, il faut seulement créer le nouvel analyseur et le générateur dans le service. Nous avons aussi proposé d'utiliser un langage de description des PDUs pour spécifier leur structure et, en utilisant un compilateur approprié, pour créer l'analyseur et le générateur automatiquement. Cela évite au développeur des services de s'impliquer dans ces tâches. Le nouveau langage *GateScript* permet aux utilisateurs de personnaliser un service. Pour les services plus compliqués comme le *Multiplexeur* du protocole X, cet environnement d'exécution permet aussi de faire appel aux programmes écrits en Java pour augmenter la performance. Dans le chapitre suivant, nous présentons une classe des services actifs sous ProAN appelés "*services proactifs*". Ces services sont utiles pour les environnements pervasifs.

Chapitre 5

Services proactifs pour les environnements pervasifs

Dans des travaux récents, David Tennenhouse a présenté le concept du "système proactif" ou *"proactive computing"* [72] : avec le développement du nombre de dispositifs de calcul, notre modèle de calcul centré-bureautique traditionnel basé sur l'interaction étroite entre l'être humain et l'ordinateur devra évoluer vers un nouveau mode de fonctionnement dans lequel les systèmes gérés en réseau sont composés d'un grand nombre de processeurs, de sondes et de déclencheurs fonctionnant d'une manière coopérative. Le rôle des être humains est alors réduit à la surveillance. Nous pensons que ces idées peuvent être bien appliquées aux réseaux et services actifs, en particulier pour des applications dans les environnements pervasifs.

5.1 Environnements pervasifs

Les environnements pervasifs sont composés des équipements (ordinateurs, capteurs, actionneurs) qui communiquent entre eux automatiquement lorsque c'est nécessaire. Les équipements sont connectés par différents types de réseaux.

Les équipements peuvent être très variés :

- à la maison : téléphone sans fil, appareils Hi-Fi, ordinateur, caméra et appareil photo numérique, alarme, contrôle de l'énergie, électroménager,
- dans le véhicule : système GPS avec Bluetooth intégré,
- au bureau : ordinateur portable, PDA,
- dans une usine : des thermomètres, des détecteurs de gaz toxique,
- dans les endroits publics comme les aéroports ou les gares : point d'accès au réseau local sans fil, moniteurs de l'environnement etc.

Ces dispositifs (fixes ou portables) peuvent communiquer à travers différents types de réseaux ayant différentes caractéristiques (bande passante, couverture, délai) : réseau téléphonique fixe ou sans fil (DECT, GPRS, UMTS etc.), réseau local filaire (Token Ring, AppleTalk, Ethernet) ou sans fil (802.11a, 802.11b, 802.11g), réseau local personnalisé (Bluetooth) etc.

Les environnements pervasifs peuvent tirer bénéfice d'un traitement adapté aux besoins du client dans des passerelles actives qui se trouvent à la frontière entre les réseaux sans fil et les réseaux fixes. La figure 5.1 illustre un exemple d'un tel environnement comportant une passerelle active.

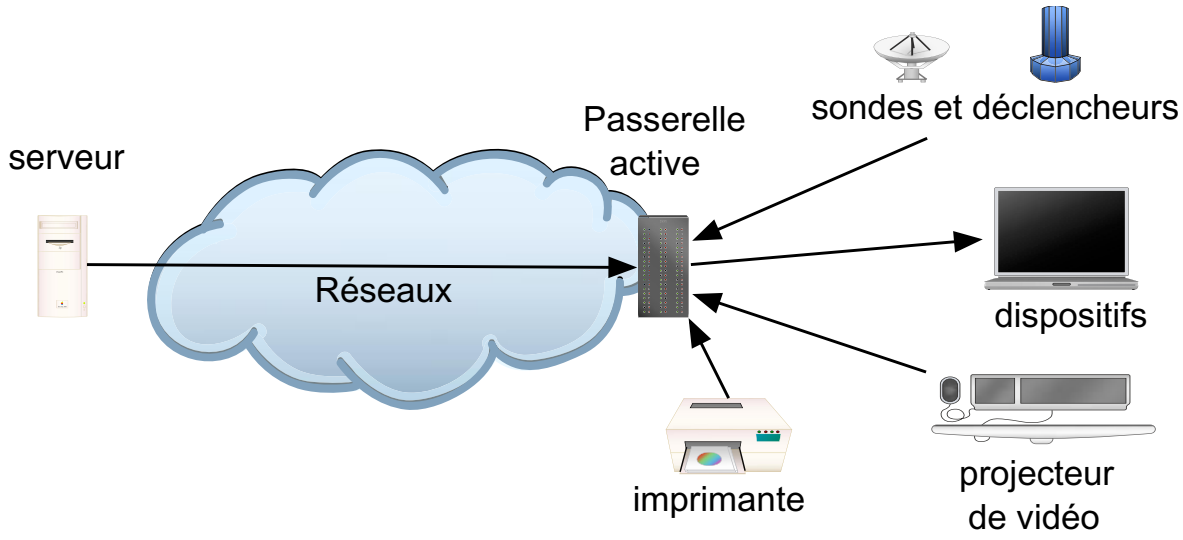

FIG. 5.1 – Environnement pervasif

5.2 Services proactifs dans ProAN

Il existe déjà certains travaux dans lesquels les auteurs suggèrent la nécessité d'une réactivité aux conditions : les applications puissent réagir efficacement aux variations des conditions dans le réseau sans fil grâce à l'aide des services actifs dans les passerelles actives. Par exemple, Boulis et al [46] ont mis en application un service actif placé dans une station de base active. Ce service fragmente les paquets audio de grande taille en plusieurs paquets plus petits. Sans ce service, dans le cas d'une mauvaise connexion sans fil, plusieurs paquets longs sont perdus. La qualité de l'audio est alors médiocre. Si ce service est activé, seule une petite fraction des paquets sont perdus et la qualité sonore est améliorée.

Mais nous pensons qu'inversement les services actifs ont également besoin de l'aide des applications et d'autres équipements du réseau tels que des routeurs et des passerelles pour réagir plus efficacement. En particulier les services actifs doivent tenir compte de l'informa-

tion sur l'état des passerelles, nœuds actifs, réseaux, applications et si possible sur l'état de l'utilisateur afin de réagir au mieux à ce qui se produit avec ces entités et ils doivent avoir la liberté de choisir le bon moment d'intercepter des paquets pour le traitement. La raison est que par exemple, dans le cas du service qui fragmente des paquets d'audio longs en paquets plus courts décrits ci-dessus, si la connexion sans fil est bonne, avec l'activation de ce service la qualité d'audio est aussi dégradée parce que le délai causé par le service augmente la gigue. Cela demande que ce service ne fonctionne que quand l'état de la connexion est mauvais. La figure 5.2 illustre un tel comportement.

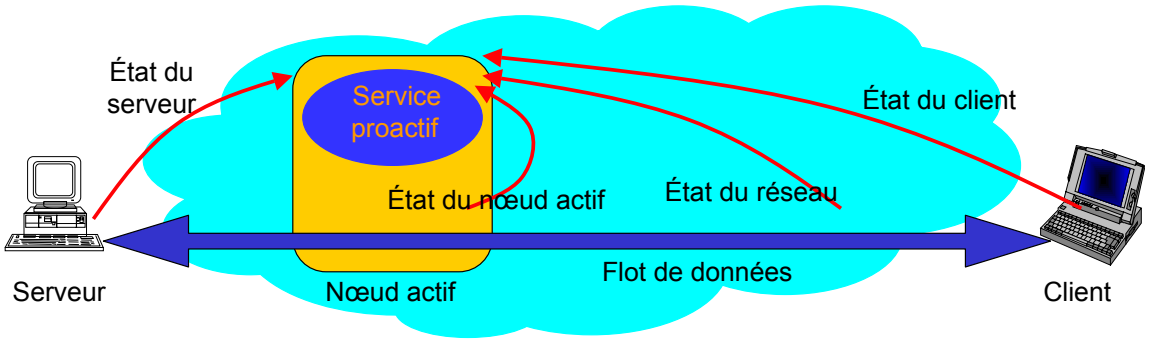

FIG. 5.2 – Service proactif

Le modèle proactif fonctionne différemment du modèle réactif. Nous rencontrons souvent des services réactifs où l'utilisateur doit explicitement demander au service ou au système de faire quelques choses en envoyant des requêtes ou des commandes. Dans le modèle proactif, c'est le service qui détecte et qui suggère les choses que l'utilisateur veut faire et les fait pour lui. L'interaction entre l'utilisateur et le service se réduit au minimum possible, voire à la situation où les services proactifs agissent automatiquement sans l'intervention de l'utilisateur.

La figure 5.3 illustre un exemple d'un service proactif dans une passerelle active. Un utilisateur mobile reçoit un flot de vidéo sur son ordinateur portable (ou son PDA) au travers du réseau local sans fil IEEE 802.11b(WLAN). Son ordinateur portable a aussi une carte Bluetooth [97]. Quand il se déplace dans un bureau où le débit du WLAN se dégrade (loin du point d'accès) il peut arriver que, à côté, se trouve un PC fixe qui a lui aussi une carte Bluetooth, l'ordinateur portable de l'utilisateur peut alors profiter de la connexion via Bluetooth avec le PC fixe pour se connecter au réseau afin de recevoir la vidéo au lieu d'utiliser la connexion WLAN. Le PC devient un routeur temporaire pour ce portable. Le service proactif doit détecter la dégradation du lien sans fil WLAN, configurer le PC comme

un routeur, ainsi que la passerelle active, pour que le flot de vidéo traverse le PC fixe avant l'arrivée au portable.

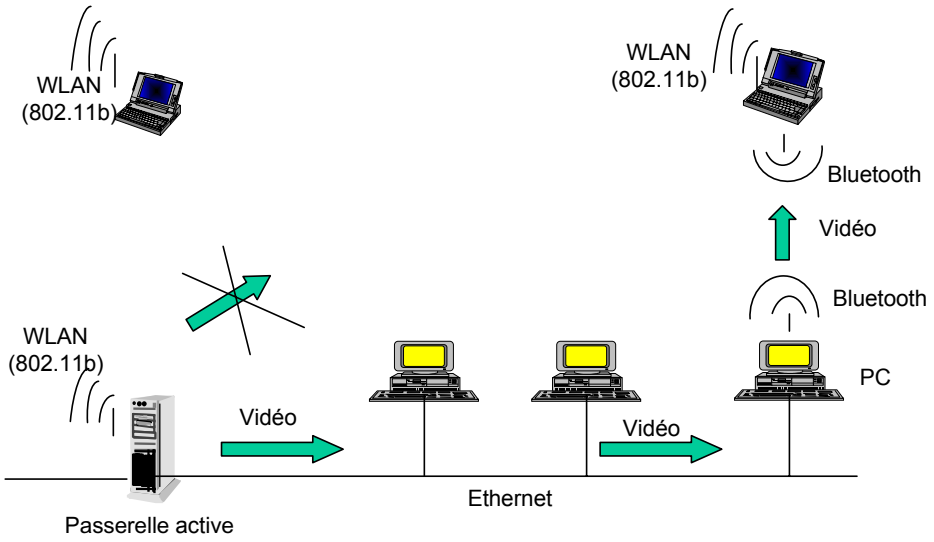

FIG. 5.3 – Utilisation d'un autre dispositif

Afin de permettre à un service proactif de coopérer avec d'autres services, nous pensons qu'il sera plus simple que chaque service fournisse une interface de contrôle et un proxy qui permette à d'autres services proactifs de communiquer avec lui. De cette façon un service proactif ne se soucie plus du protocole de communication avec d'autres services. Par conséquent chaque type de service proactif doit se conformer à une interface commune de contrôle et est libre de fournir le proxy correspondant. Nous proposons par exemple que tous les modules de contrôle des passerelles actives offrent une interface commune de contrôle, car les passerelles actives peuvent être vues comme des services. Et chaque type de passerelle active peut fournir le proxy correspondant.

De même, chaque service de transcodage de vidéo, quelque soit son implémentation et sa plate-forme, peut fournir un proxy représentant l'interface commune de contrôle de tous les types de service de transcodage de vidéo.

La figure 5.4 illustre l'interaction entre les services proactifs par l'intermédiaire de leurs proxies dans ProAN. Le service de découverte joue un rôle important pour les services proactifs. Il leur permet de découvrir les autres services (Ceci inclut aussi les moniteurs de l'environnement et les autres passerelles actives) dans le réseau.

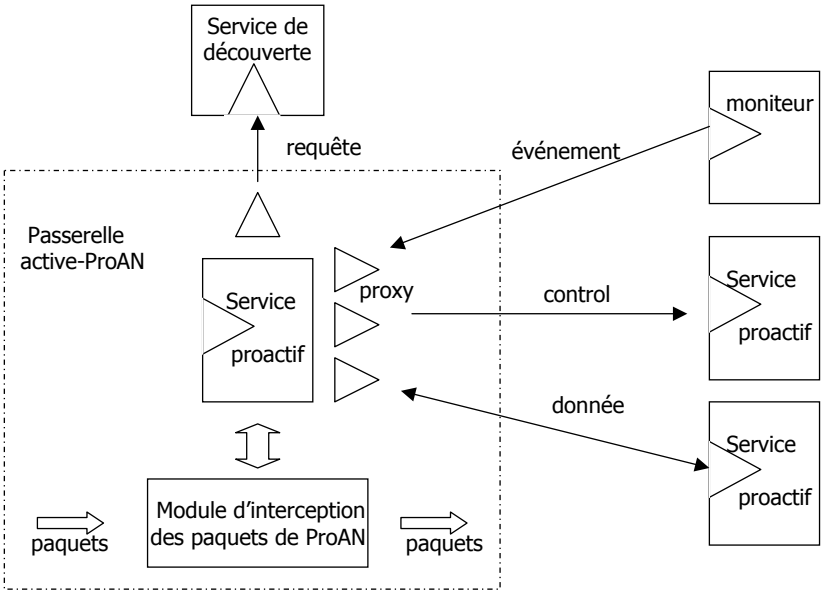

FIG. 5.4 – Interaction entre les services proactifs

Nous avons choisi Jini [91] comme service de découverte dans ProAN pour implémenter des scénarios d'application montrés plus loin dans la section 5.4. Quand chaque service proactif ou chaque module de contrôle d'une passerelle active est activé, il doit enregistrer son proxy auprès du service de découverte pour que les autres services puissent le découvrir. Toutes les entités qui veulent se rendre découvrables fournissent une interface de contrôle et un proxy (stub) qui implémente cette interface et cache le protocole d'accès. Son rôle est d'amorcer la communication entre les services sans qu'il soit nécessaire de connaître un protocole d'accès particulier exigé par un service. Des stubs sont enregistrés auprès du service de découverte, de sorte que d'autres services puissent les découvrir et les télécharger. L'interface de contrôle peut fournir des méthodes pour obtenir la description du service ou son état actuel.

5.3 ProAN dans les terminaux

Bien que les réseaux actifs concernent le plus souvent les éléments réseau (routeurs, passerelles), nous pensons que les terminaux ont aussi intérêt à être actifs. C'est à dire que l'on peut aussi contrôler ces terminaux à distance, ou télécharger des services à exécuter. En effet la notion de nœud actif peut être facilement étendue au terminal qui devient un routeur dans certaines conditions.

Les utilisateurs ont de plus en plus des équipements à leur disposition : ordinateur fixe avec une connexion permanente à l'Internet, ordinateur portable, PDA avec une connexion sans fil de type 802.11. Mais ils ont aussi des appareils ayant seulement une connexion sans fil limitée à une dizaine de mètres comme Bluetooth ou infrarouge : un appareil photo numérique, une montre avec un moniteur de la fréquence cardiaque intégré, des oreillettes numériques etc. Pour que ces appareils puissent être connectés au réseau global, la meilleure solution est de permettre aux terminaux ayant la connexion la plus performante de devenir des passerelles. C'est à dire que, à travers ces terminaux, les plus petits appareils avec une connexion sans fil limitée peuvent être disponibles et détectables par les autres services dans le réseau comme des serveurs, des services actifs dans des passerelles, des proxies ou d'autres applications. Les services actifs et les applications ont intérêt à communiquer avec les petits appareils, sources des informations intéressantes concernant les utilisateurs et l'environnement autour d'eux.

Le téléphone portable est un terminal sans fil que nous jugeons approprié pour être actif. À présent presque chacun possède un téléphone qui est gardé à proximité. Par conséquent, le téléphone portable est une bonne passerelle entre par exemple des moniteurs de la santé de l'utilisateur ou des moniteurs de l'état de l'environnement, et les services proactifs dans le réseau. Par exemple, si quelqu'un avec son portable se trouve à son bureau, le téléphone portable peut détecter le PC fixe de son bureau grâce au Bluetooth. Si un ami lui téléphone, l'appel sera routé vers le téléphone fixe dans son bureau. Cela présente un intérêt car le coût du téléphone fixe est toujours moins élevé que celui du sans fil et la qualité de la communication est supérieure. De même, l'appel pourra être routé à son PC de bureau pour utiliser VoIP. Si son ami utilise lui aussi un PC, l'appel sera gratuit. Dans cet exemple, c'est le téléphone portable qui détecte que l'utilisateur se trouve à côté de son PC fixe au bureau. Le service de routage actif des appels de l'opérateur téléphonique doit savoir si l'utilisateur se trouve à côté d'un autre téléphone ou un PC fixe ou non pour décider d'une action appropriée.

Comme l'environnement autour de l'utilisateur change au cours du temps, on ne sait pas combien il y a de sondes ou de dispositifs qui s'inscrivent auprès des terminaux autour de l'utilisateur. Avec notre architecture de ProAN, les services proactifs peuvent :

- contrôler, télécharger ou activer un service dans les terminaux actifs
- déployer un moniteur actif sur un terminal pour surveiller l'état du terminal et les autres dispositifs (sondes) autour du terminal

– consulter une base de donnée des dispositifs autour du terminal et ainsi ses états pour les tâches de gestion. (Un service intéressant serait un mini service de gestion de base de données qui permet de traiter des requêtes SQL envoyés par plusieurs entités).
– intercepter des flots de données brut provenant des sondes autour du terminal pour filtrer et extraire seulement des informations utiles.

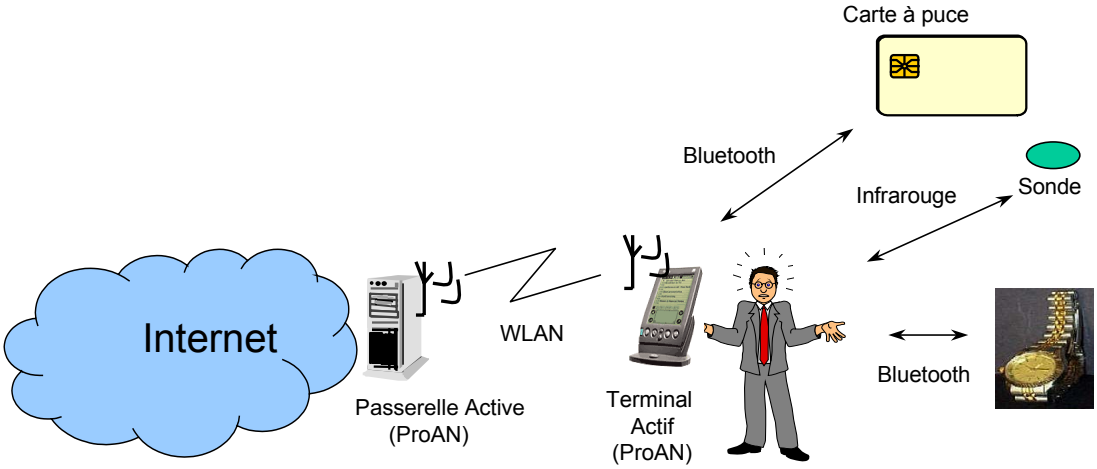

FIG. 5.5 – ProAN dans le terminal

Ces visions de l'utilisation du terminal actif dépassent celles, classiques, qui ont été déjà invoquées dans plusieurs projets existants, où un terminal actif permet seulement de télécharger des nouvelles applications actives et de les activer au besoin.

La figure 5.5 illustre un exemple d'un terminal actif avec ProAN. Grâce au service de découverte intégré, les autres sondes et dispositifs autour du terminal peuvent l'utiliser comme un service d'annuaire et une passerelle avec l'Internet. Ils peuvent inscrire leur proxies auprès du terminal avec un nom et des propriétés permettant aux autres services de les découvrir et les interroger, ou éventuellement les contrôler.

5.3.1 Portage de ProAN sur un terminal

Nous avons porté ProAN sur un terminal - un PDA iPAQ.

Les iPAQs sont livrés avec Windows CE par défaut. La première étape est de remplacer Windows CE par une version du Linux déjà compilé pour ce modèle de l'iPAQ. Ce processus a été bien développé et détaillé dans de multiples documents [86]. Mais le processus pour recompiler le noyau avec des modifications dans les modules et le remplacer sur un iPAQ s'est heurté à plusieurs problèmes. Nous avons eu de nombreuses surprises : des erreurs dans

les codes, des options qu'il faut configurer etc. Nous présentons en détail ce processus en annexe 8.1.

5.4 Scénarios d'applications

Nous décrivons ci-après quelques exemples des services proactifs pour les environnements pervasifs. L'implémentation de certains de ces services est présentée dans la section 6.2.

5.4.1 Service d'affichage proactif du contenu

Ce service montre qu'un service proactif peut profiter d'autres dispositifs autour de l'utilisateurs pour augmenter la qualité de service. Un utilisateur avec un PDA est en train de voir une vidéo grâce à un service de transcodage qui transcode la vidéo du format original MPEG1 au H263. L'utilisateur entre dans une salle où il y a un grand écran disponible (ou un ordinateur) qui peut afficher le MPEG1. Un événement est envoyé au service de transcodage pour l'informer de la présence de cet écran. Le service de transcodage s'arrête de transcoder et envoie directement la vidéo MPEG1 à l'écran pour afficher. Dans ce cas non seulement l'utilisateur peut profiter du grand écran et de la qualité de la vidéo, mais le nœud actif sur lequel le service de transcodage s'exécute peut aussi économiser des ressources. Quand l'utilisateur sort de la salle, un autre événement est envoyé au service de transcodage qui reprend le travail et envoie la vidéo transcodée en format H263 au PDA.

Les services proactifs doivent aussi s'assurer que l'utilisateur veuille l'action proposée. Par exemple dans le cas où il y a beaucoup d'autres personnes dans la salle, il ne serait pas souhaitable d'afficher la vidéo sur le grand écran car cela peut gêner les autres. Dans un tel cas, le service proactif doit demander à l'utilisateur à l'avance, en proposant un petit menu pour savoir s'il veut que sa vidéo soit affichée sur le grand écran ou pas. L'implémentation de ce scénario d'application est détaillée dans la section 6.2.2.

5.4.2 Continuation d'une session

Une personne A équipé d'un PDA se trouve dans un aéroport pour prendre son avion. Il y a plusieurs autres passagers autour de lui attendant l'embarquement. Parmi eux il y a une personne B avec ordinateur portable connecté au réseau Wi-Fi de l'aéroport. Cette personne B a un film dans son ordinateur portable, et est disposée à le partager. Grâce au service de découverte, l'utilisateur A le sait, et est intéressé par le film mais son PDA n'a pas suffisamment d'espace disque pour le télécharger totalement. (Chaque fichier de film demande environ un gigaoctets). Le PDA peut activer un service de diffusion (*streaming*) de vidéo sur une passerelle active de l'aéroport. Le service de diffusion de vidéo télécharge

le film depuis l'ordinateur portable de la personne B dans le serveur de stockage temporaire dans le réseau, et envoie en temps réel le film vers le PDA. Le service de streaming envoie un moniteur au PDA pour collecter l'information concernant l'heure de départ et le numéro de vol. Quand celui-ci doit embarquer, et s'il n'a pas encore terminé de voir le film, le service de streaming peut l'avertir et télécharger le reste du film au même service dans l'avion sur lequel cette personne A va embarquer. La personne A peut alors continuer voir le film dans l'avion.

5.4.3 Utilisation d'un autre réseau d'accès

Dans un réseau local, quand la connexion avec l'Internet à travers le réseau fixe est interrompue (à cause d'une panne du réseau du campus ou du réseau d'accès), la connexion avec l'Internet peut être maintenue grâce à un ordinateur de bureau (un PC) qui a une connexion Bluetooth [97] avec un téléphone mobile GPRS, comme indiqué dans la figure 5.6. Le service proactif se trouvant dans une passerelle doit détecter tout de suite la déconnexion avec l'extérieur, et configurer le PC comme un routeur principal avec l'extérieur. Le service proactif doit d'abord chercher quels sont les PCs dans le réseau ayant une connexion Bluetooth avec un téléphone portable qui permette de se connecter à l'extérieur à travers le réseau GPRS. Ensuite le service proactif met en place le table de routage sur ces PCs.

Il doit aussi activer le service NAT (*Network Address Translation*) [80] dans ces PCs pour que les paquets entrants puissent passer par le téléphone portable. Le service proactif utilise le protocole DHCP [101] pour envoyer l'information concernant les nouveaux routeurs (les PCs ayant l'interface Bluetooth) aux ordinateurs portables sans fil ou aux autres PCs pour changer cette information automatiquement.

Le protocole DHCP demande que le client DHCP envoie explicitement une requête vers le serveur DHCP pour obtenir des informations de configuration. Mais nous avons ici un exemple un comportement proactif : dans ce cas c'est le service proactif dans la passerelle active qui pousse cette information vers la machine où se trouve le serveur DNS et aux autres ordinateurs du réseau.

Puisque certains systèmes d'exploitation ne permettent pas de changer dynamiquement la configuration du réseau (e.g l'adresse du routeur par défaut), le service proactif dans la passerelle active doit aussi reconfigurer la passerelle pour qu'elle route tous les trafic au nouveau routeur - le PC avec le Bluetooth. S'il y a plusieurs PCs avec une connexion Bluetooth, et si chaque PC peut se connecter à l'Internet à travers un téléphone portable, la passerelle active peut jouer aussi un rôle de distributeur de charge pour répartir les trafic sur plusieurs PCs, car les connexions avec Bluetooth sont faibles en débit - environ 1 Mbits/s.

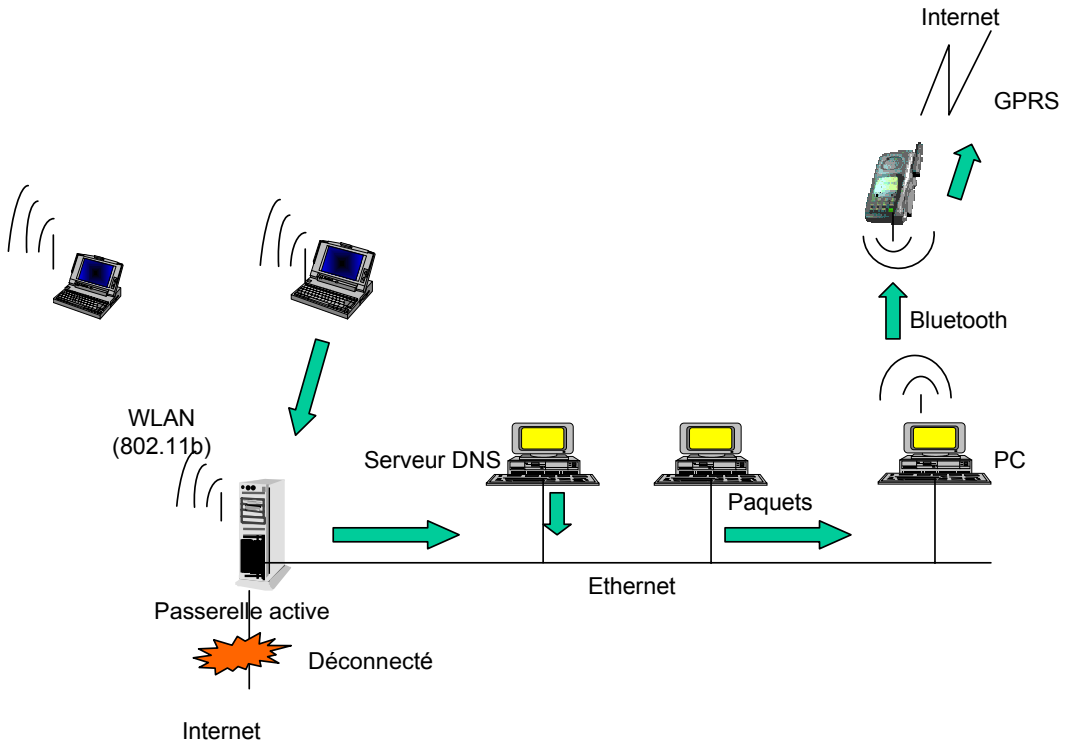

FIG. 5.6 – Utilisation d'un autre réseau d'accès en cas de déconnexion

5.4.4 Service d'aide imprimante

Nous avons vu qu'il y a beaucoup de services qui font de la conversion des données pour les dispositifs de petite taille (PDA, téléphone portable etc.). Par exemple s'il y a un PDA qui ne peut pas afficher un fichier de type Microsoft Word, un service de conversion extrait le texte du fichier Word et l'envoie au PDA. Si l'utilisateur du PDA veut imprimer ce texte, il ne peut pas l'imprimer un formattage correct. Un service de conversion peut stocker le document Word, et permettre à l'utilisateur d'appeler l'application Word sur une autre machine et de diriger la sortie vers l'imprimante de son choix. De cette façon, l'utilisateur peut avoir une bonne version imprimée du texte. Le même service avec d'autres formats comme PowerPoint, Word Perfect, OpenOffice, Postscript ou PDF serait intéressant.

5.4.5 Service d'appel intelligent

Imaginons un utilisateur qui arrive à son bureau. Un ami l'appelle sur son téléphone portable. Autour de lui il y a un téléphone fixe sur le bureau et son ordinateur de bureau est allumé. L'appel sur son portable pourrait être routé sur le téléphone fixe grâce à un service proactif qui sait traiter les appels et qui peut découvrir un téléphone fixe à côté du portable. Par conséquent, la communication serait moins chère. Le service proactif sur son portable peut même activer une application VoIP sur son ordinateur de bureau et si son ami utilise lui aussi VoIP, l'appel serait gratuit.

Un autre type de service d'appel intelligent peut avoir lieu selon le scénario suivant : On oublie parfois son téléphone portable à la maison, ou oublie de vérifier, ou estime mal le niveau de la batterie du téléphone portable. Dans ce cas on peut se trouver dans la situation où autour de nous il y a des amis ou des gens avec un téléphone fonctionnant mais le notre ne l'est pas. Par conséquent nos proches ne peuvent plus nous joindre. Si sur les téléphones portables il y a un service proactif qui nous permet d'enregistrer notre numéro de téléphone sur un autre téléphone de quelqu'un qui se trouve actuellement proche de nous, le service proactif va enregistrer cette information au serveur de l'opérateur, et quand il y a un appel sur notre numéro, cet appel serait routé vers le téléphone portable de la personne qui nous a permis d'enregistrer notre numéro. Comme ça nous pouvons profiter de son téléphone quand le notre n'est plus disponible, et nous sommes toujours joignable. Dans le cas où nous sommes avec notre téléphone mais il est presque à court de batterie, le service proactif sur notre téléphone peut le détecter et contacter un autre téléphone autour de nous pour enregistrer notre numéro automatiquement (avec l'accord préalable du propriétaire du téléphone en question).

5.4.6 Service d'information

Un utilisateur avec un WAP-phone arrive dans une ville. Son téléphone portable peut contacter les systèmes GPS dans des voitures de passage autour de lui par Bluetooth pour avoir l'information précise de l'endroit où il se trouve. Avec cette information, l'utilisateur peut connaître tous les services disponibles autour de lui, comme un centre commercial, un site historique, un bon restaurant, une station d'essence, une banque, un point argent, un bureau de tabac, un hotel bon marché, des WC publics etc, à travers le téléphone portable.

5.4.7 Service de téléchargement intelligent

Nous constatons tous que les serveurs FTP et les clients FTP ne supportent pas encore la déconnexion. Si un client FTP se déconnecte au milieu du processus de téléchargement, il doit souvent recommencer le processus dès le début. Nous avons développé un service de proxy FTP proactif qui permet des clients FTP conscients des déconnexions de pouvoir continuer

le téléchargement quand la nouvelle connexion serait établie. Ce service proxy FTP proactif peut aussi détecter si le client FTP a suffisamment d'espace de stockage, pour chercher, le cas échéant, un autre endroit pour stocker le contenu désiré (au serveur à la maison par exemple).

5.5 Conclusion

Dans ce chapitre nous avons présenté une classe des services actifs appelés "services proactifs" pour des environnements pervasifs. L'idée principale est que les services actifs doivent changer leur manière de fonctionner afin de devenir des services proactifs, pour être plus utiles dans ces environnements. Nous voulons réduire au maximum les interactions entre les utilisateurs et les services dans le réseau. Dans un tel environnement, la notion de terminal s'atténue car les terminaux peuvent devenir des routeurs quand c'est nécessaire pour fournir une meilleure qualité de service. Le service de découverte permet aux services proactifs de trouver les autres services dans leur environnement, ainsi que de moniteurs fournissant des informations concernant l'état du réseau et l'utilisateur. Nous avons aussi présenté quelques scénarios utiles d'applications du réseau actif dans les environnements pervasifs et nous croyons il y en aura beaucoup d'autres intéressants à investiguer dans l'avenir. Dans le chapitre suivant, nous présenterons l'évaluation des performances de ProAN, de l'environnement d'exécution de *GateScript* ainsi que nos expériences sur l'implémentation de divers services proactifs présentés dans ce chapitre.

Chapitre 6

Évaluation et Expérience

Nous présentons dans ce chapitre les résultats de l'évaluation des prototypes développés ainsi que le retour d'expériences effectuées avec des services proactifs.

6.1 Évaluation

6.1.1 Évaluation de ProAN

Pour évaluer les performances de l'implémentation du module d'interception des paquets de ProAN sous Linux, nous avons mesuré le temps d'expédition des paquets et le temps nécessaire pour les passer à un service actif dans l'espace utilisateur sur un PC Pentium III de 800 mégahertz, 128 MB RAM exécutant Linux RedHat 7.2 avec la version 2.4.16 du noyau.

La figure 6.1 présente le temps d'expédition des paquets en fonction de leur taille pour deux cas :

- dans le premier cas, les paquets entrent dans le noyau et ils sont simplement expédiés à la destination (aucun service actif n'est installé) ;
- dans le deuxième cas, un service actif est installé et activé. Celui-ci injecte un filtre qui intercepte des paquets. Le service n'effectue aucun traitement et réinjecte simplement ces paquets dans le noyau pour expédition.

Nous avons mesuré le délai de passage par le nœud ProAN. La différence entre les deux courbes représente le délai ajouté par le fait de passer un paquet à un service actif dans l'espace utilisateur. Dans ce cas (service actif installé), les paquets rencontrent un délai augmenté en fonction de la taille des paquets.

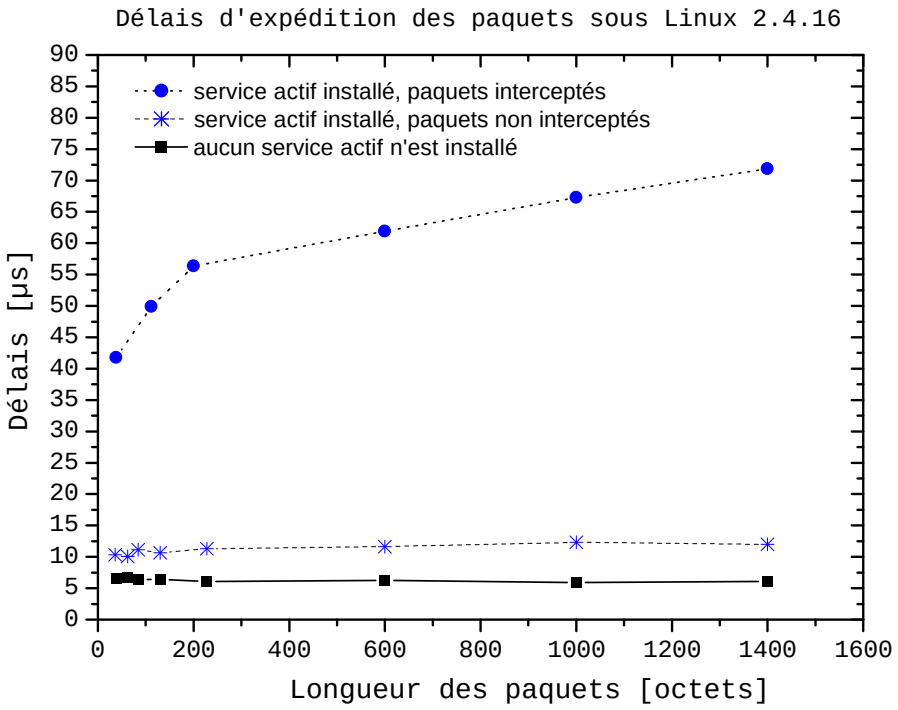

FIG. 6.1 – Délais d'expédition des paquets du ProAN

En laissant le service actif intercepter des paquets, nous avons fait passer un deuxième flot des paquets qui ne sont pas interceptés par aucun service actif. Quand il y a un service actif qui est en train d'intercepter des paquets du premier flot, le délai du deuxième est à peu près le même que dans le cas où aucun service actif n'est installé.

Ces résultats prouvent que les retards apparaissent seulement sur les flots de données sur lesquels les services actifs doivent effectuer un traitement utile : le retard pour les flots qui ne sont pas interceptés reste limité même si un service actif est en train de traiter des paquets.

6.1.2 Évaluation du moteur d'exécution de GateScript

Nous avons fait des expériences avec le langage *GateScript* en mettant en application un proxy HTTP actif qui analyse le trafic de HTTP au nom d'un utilisateur et effectue la personnalisation des données (filtrage des bannières de publicité, réduction de la taille d'image etc.). Pour évaluer notre implémentation, nous avons mesuré les performances de notre proxy HTTP programmé en *GateScript* sur un PC Pentium III 1.06 GHz avec 148 MB

RAM sous Windows XP et comparé avec la performance d'un proxy HTTP public appelé *Muffin* [105]. Les deux proxy sont exécutés avec Java 2 SDK 1.4.1.

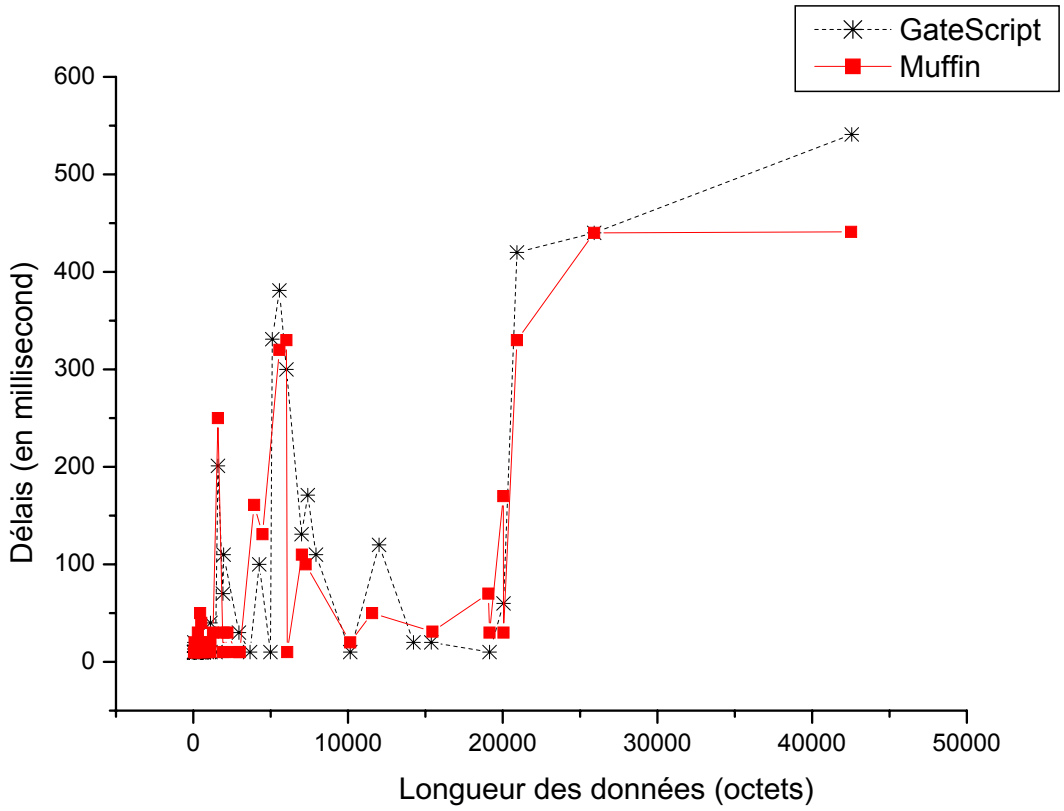

FIG. 6.2 – Performance du GateScript vs. Muffin, aucun traitement

Dans le premier test, nous avons téléchargé les pages du site *www.cnn.com* en passant par les deux proxy qui ne font aucun traitement sur les réponses HTTP. La figure 6.2 compare le délai de notre proxy avec *GateScript* et Muffin. Les performances sont comparables.

Dans le deuxième test, des traitements sont associés aux réponses HTTP : chaque page est analysée et toutes les images sont éliminées. La figure 6.3 présente la comparaison du délai des deux proxy. De nouveau on constate que notre passerelle offre des performances comparables avec une passerelle spécialisée.

6.2 Expérience

Dans cette section, nous décrivons nos expériences sur l'implémentation des services proactifs décrits en section 3.3.11 et 5.4.

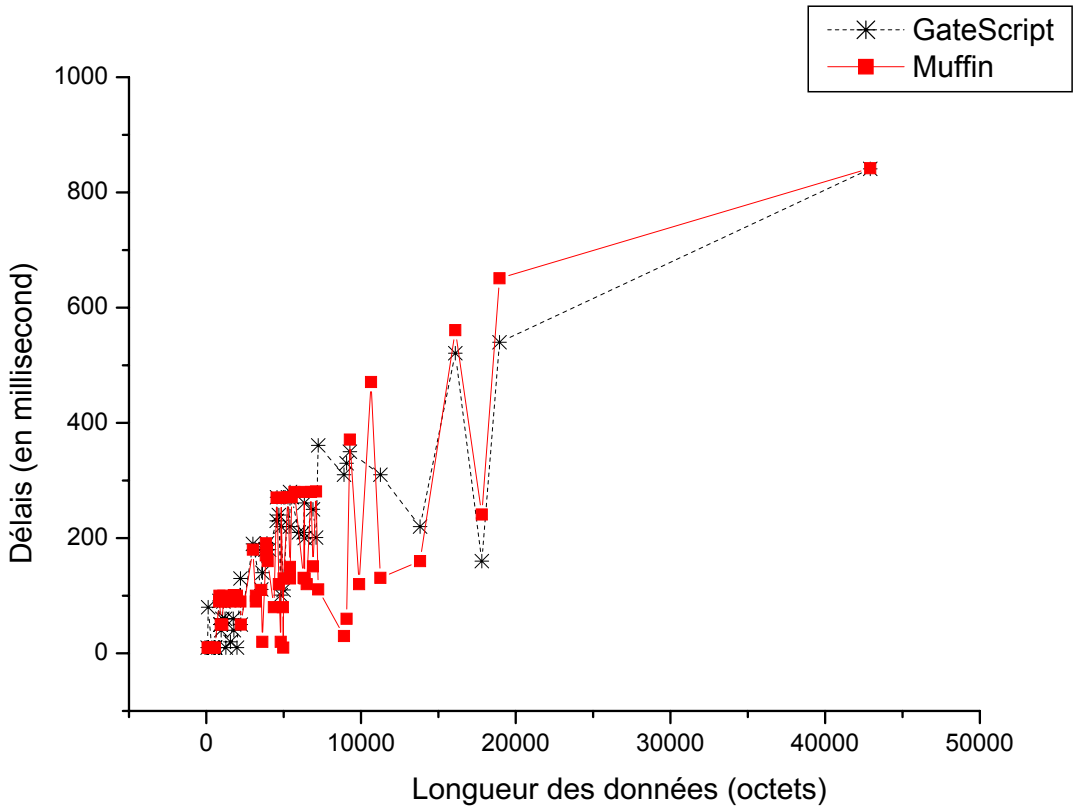

FIG. 6.3 – Performance du GateScript vs. Muffin, élimination des images

6.2.1 Service de transcodage de vidéo

Nous avons utilisé le framework JMF - *Java Média Framework* [104] pour réaliser le service de transcodage de vidéo décrit en section 3.3.11. Nous avons fourni deux plug-ins (plugin d'entrée et plugin de sortie) au JMF. Le plugin d'entrée (*input plugin*) a pour but de lire un flot IP/UDP et d'en extraire les paquets RTP (ou RTCP) pour passer au module `InputDataSource` - une interface qui représente la source de média du JMF. Rappelons que les paquets reçus par les services proactifs dans l'espace utilisateur de ProAN sont des paquets IP entiers avec l'en-tête d'IP. Pour que les paquets de la vidéo déjà transcodée puissent re-entrer dans le noyau pour continuer la route, il faut leur ajouter les options de l'en-tête d'IP/UDP. Cela est assuré par le plugin de sortie qui fait l'inverse : lire les flots de RTP du module `OutputDataSource` du JMF et créer un flot IP/UDP.

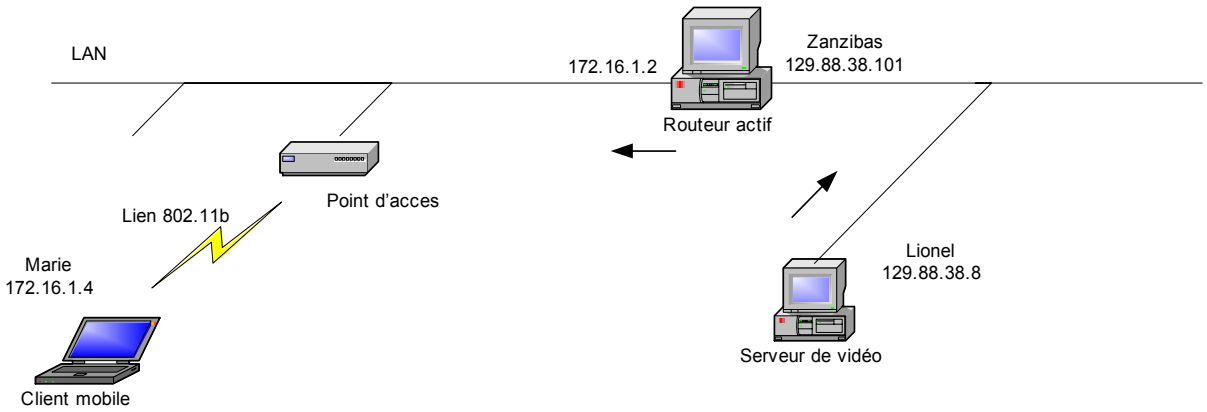

FIG. 6.4 – Environnement de test

La figure 6.4 illustre notre environnement de test qui a les composants suivants :

- Client mobile : un ordinateur portable HP OmniBook sous Windows 98, Pentium III 500 MHz, 128 MB RAM avec une carte sans fils Lucent WaveLan 802.11b. Le client utilise l'outil JMStudio de JMF pour afficher la vidéo.
- Routeur actif : un PC Pentium III 800MHz, 128MB RAM avec notre passerelle active ProAN installée qui permet de passer dynamiquement des flots de paquets au service de transcodage.
- Serveur de vidéo : un ordinateur portable avec un processeur AMD Duron 1GHz, 256 MB RAM, dans lequel on installe JMF, JMStudio sur Windows XP pour transmettre la vidéo.

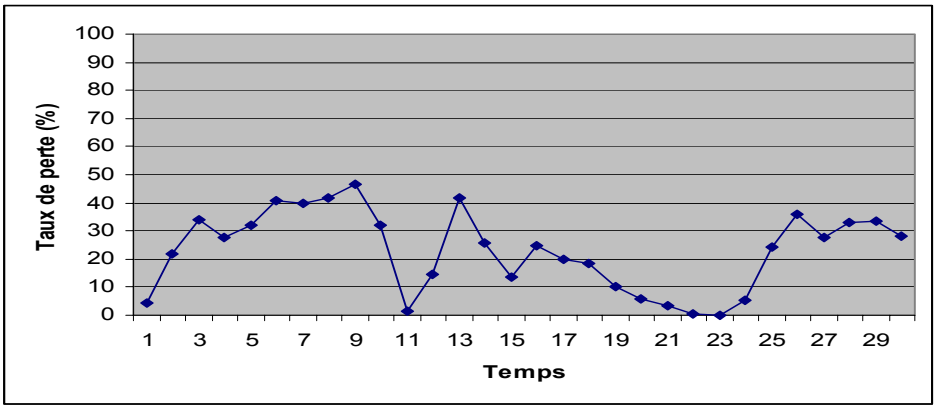

FIG. 6.5 – Variation du taux de perte au cours du temps

La figure 6.5 représente la variation du taux de perte de données donné par le client mobile à travers le protocole RTCP. Si on s'éloigne encore du point d'accès, le client ne peut plus afficher correctement la vidéo.

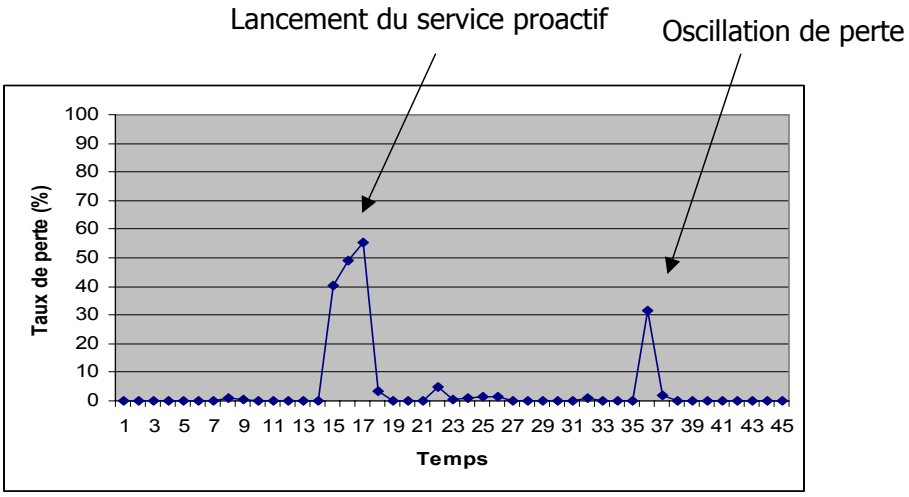

FIG. 6.6 – Lancement du service proactif

La figure 6.6 nous montre le moment où le service proactif de transcodage de vidéo est réveillé par le moniteur RTCP. Quand la bande de passante diminue, le taux de pertes augmente très vite. Le service de transcodage est alors réveillé pour intercepter les flots RTP et réduire le débit en transformant le flot de vidéo original (MPEG1) en un flot de bas débit au format H263. Grâce à ceci, le client peut afficher la vidéo normalement. Parfois, il y a une oscillation de perte comme le deuxième pic de la courbe : dans ce cas, le service ne change pas son état.

6.2.2 Service d'affichage proactif du contenu

Nous avons utilisé Jini [91] comme protocole de découverte de service pour implémenter le scénario de service décrit en section 5.4.1. Nous utilisons également JMF pour implémenter le transcodeur. Dans cette version de l'implémentation, l'écran (un PC) est enregistré au service d'annuaire de JINI pour que le service de transcodage puisse le découvrir.

Quand l'afficheur de vidéo dans le PDA est activé, il recherche à l'aide du service de découverte Jini une passerelle active ProAN à laquelle il peut demander d'exécuter un service de transcodage. Une fois obtenu le proxy du module de contrôle de la passerelle, il envoie une commande pour activer le transcodeur avec les paramètres tels que l'adresse de

la source de vidéo, l'adresse de destination (son adresse), et également les numéros des ports correspondants.

Le transcodeur, à son tour, contacte la source de vidéo pour la vidéo et envoie également une commande au service d'annuaire Jini afin de recevoir un événement quand un nouveau grand écran apparaît. Le grand écran dans notre test est un PC portable, avec une carte sans fil 802.11b sur lequel il y a aussi une passerelle active ProAN.

Le service de transcodage effectue son rôle en transcodant la vidéo MPEG au format H263, l'audio MPEG au format GSM et en les transmettant au PDA. Le PC portable simulant le grand écran est éteint.

Quand l'écran apparaît (le PC portable est allumé), le module de contrôle de la passerelle sur cet écran est activé et enregistre ensuite son proxy auprès du service d'annuaire Jini avec la propriété "*grand écran*". Un événement est alors envoyé par Jini au service de transcodage. Le service de transcodage s'arrête de transcoder et envoie une commande au module de contrôle de la passerelle sur l'écran (PC portable) afin d'activer le service d'affichage de vidéo sur le PC portable. Le service de transcodage envoie ensuite la vidéo et l'audio au format original MPEG à l'écran.

Dans l'interface du proxy du module de contrôle de passerelle active, il y a une méthode permettant de connaître l'adresse du nœud. Par conséquent, le service de transcodage connaît l'adresse de l'écran pour envoyer la vidéo.

Quand le PC portable est éteint, (l'utilisateur est loin de l'écran), un autre événement est envoyé par Jini au service de transcodage qui reprendra le travail et envoie la vidéo et l'audio transcodés au PDA.

6.2.3 Service de multiplexeur du protocole X

Nous avons implémenté ce service pour démontrer les propriétés de l'architecture générique des services actifs avec *GateScript*.

Le service de multiplexeur du protocole X permet de répliquer une session sur plusieurs écrans. Il permet à un client X d'afficher une fenêtre d'application sur plusieurs serveurs. Par conséquent, plusieurs utilisateurs peuvent voir et interagir avec la même application en même temps de différents endroits. On peut envisager l'application d'un tel service dans le domaine de simulation, travail collaboratif, ou télé-enseignement.

Ce service actif de multiplexeur joue le rôle intermédiaire (passerelle) entre un client et différents serveurs X. Il intercepte et traite les requêtes et les réponses entre le client et les serveurs pour que le client croie qu'il affiche sur un seul serveur. La figure 6.7 illustre ce processus.

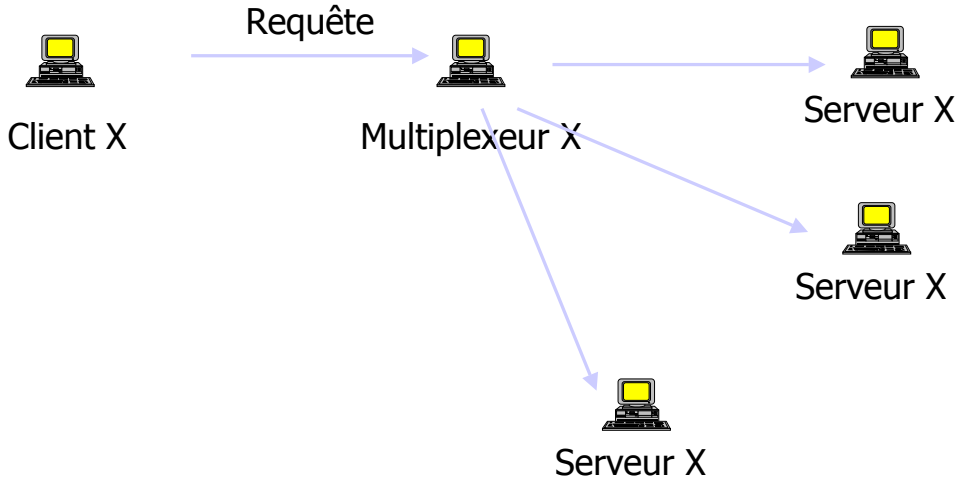

FIG. 6.7 – Service de multiplexeur du protocole X

L'architecture générale du système X, de type client-serveur, est représentée dans la figure 6.8 :

- le client, est une application qui a besoin d'afficher des informations sur un écran.
- le serveur est un programme, exécuté sur une machine ("machine hôte") connectée habituellement à un écran, un clavier et une souris. Il offre des services d'affichage aux clients qui en ont besoin.

Le client envoie des requêtes au serveur et reçoit éventuellement des réponses ou des erreurs. Le serveur peut aussi envoyer des événements au client, pour lui signaler des modifications ou des actions de l'utilisateur. La communication entre le client et le serveur se fait grâce au protocole X basé sur TCP/IP.

Parmi les requêtes qu'un client peut envoyer à un serveur, nous trouvons principalement :
- des requêtes de manipulation de fenêtres : création, destruction, déplacement, redimensionnement,... ;
- des requêtes de dessin : lignes, rectangles, arcs de cercles, images,... ;
- des requêtes d'administration : utilisées pour la gestion des ressources du serveur.

Les événements quant à eux peuvent être classés en deux catégories :

- des événements de manipulation de fenêtres : le serveur X, en réponse aux requêtes

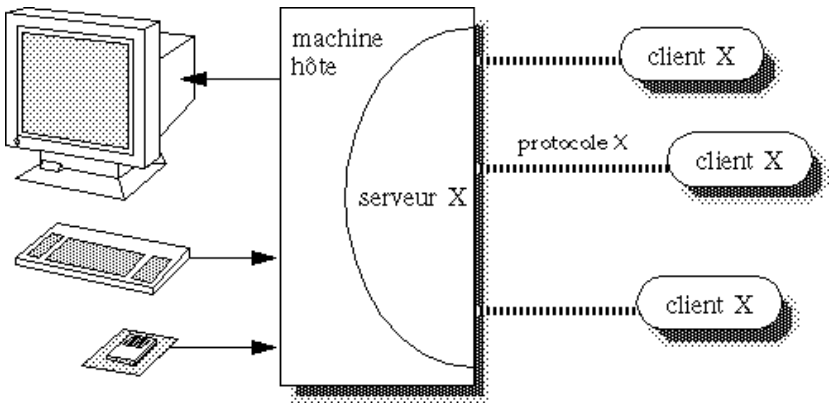

FIG. 6.8 – Architecture du système X

ou à des actions externes (déplacement d'une fenêtre par l'utilisateur, par exemple), envoie des événements de notification sur la vie des fenêtres (création, destruction, etc...) ;
– des événements concernant l'utilisateur : lorsque l'utilisateur déplace la souris, appuie ou la relâche ou appuie sur une touche de clavier ou la relâche, le serveur X envoie un événement pour le signaler.

Dans le protocole X, chaque requête, réponse, événement ou erreur est encapsulé dans une PDU. Il y a au total plus de 127 types de PDUs différentes.

Chaque composant sur le serveur X comme une fenêtre ou une police de caractère, doit avoir un identificateur (ID) unique. Par conséquent, avant d'envoyer une requête dupliquée à plusieurs serveurs, le service actif de multiplexeur doit analyser la requête et changer les identificateurs correspondants. Il fait la même chose pour les événements et les réponses.

En appliquant l'architecture générique des services actifs, nous avons utilisé le langage

Flavor pour spécifier le protocole X : les requêtes, les réponses, les événements et les erreurs. Voici un extrait des PDUs de requêtes spécifiés en Flavor.

```
class X_PDU {
int(8) opcode;

if (opcode < 0) { int(8) request[31]; }
if (opcode == 1) {

    int(8) unusedByte;
    int(16) length;
    int(32) wid;
    int(32) parent;
    int(16) x;
    int(16) y;
    int(16) width;
    int(16) height;
    int(16) border_width;
    int(16) ClassS;
    int(32) visual_id;
    int(8)  value_mask[4];
    int(8)  value_list[4*(length-8)];
}

if (opcode == 2) {

    int(8) unusedByte;
    int(16) length;
    int(32) window;
    int(8) value_mask[4];
    int(8) value_list[4*(length-3)];
}}
```

La variable `"opcode"` spécifie le type de PDU (requête) et occupe toujours le premier octet dans toutes les PDUs.

Voici une partie de la classe générée (l'analyseur de PDU) par le compilateur du Flavor, nous montrons seulement une partie de cet analyseur du protocole X.

```
import flavor.*;
import java.io.*;

public abstract class X_PDU {
    int opcode;
    int request[] = new int[64];
    int unusedByte;
```

```
int length;
int wid;
int parent;
int x;
int y;
int width;
int height;
int border_width;
int ClassS;
int visual_id;
int value_mask[] = new int[64];
int value_list[] = new int[64];
int window;
int unusedByteArray[] = new int[64];


public abstract int execute();


public int get(Bitstream _F_bs) throws IOException {
    int _F_ret = 0;
    int _F_dim0, _F_dim0_end;

    opcode = _F_bs.sgetbits(8);

    if ((opcode<0))
    {
        _F_dim0_end = 31;
        for (_F_dim0 = 0; _F_dim0 < _F_dim0_end; _F_dim0++) {
            request[_F_dim0] = _F_bs.sgetbits(8);
        }
    }

    if ((opcode==1))
    {
        unusedByte = _F_bs.sgetbits(8);
        length = _F_bs.sgetbits(16);
        wid = _F_bs.sgetbits(32);
        parent = _F_bs.sgetbits(32);
        x = _F_bs.sgetbits(16);
        y = _F_bs.sgetbits(16);
        width = _F_bs.sgetbits(16);
        height = _F_bs.sgetbits(16);
        border_width = _F_bs.sgetbits(16);
        ClassS = _F_bs.sgetbits(16);
        visual_id = _F_bs.sgetbits(32);
        _F_dim0_end = 4;
        for (_F_dim0 = 0; _F_dim0 < _F_dim0_end; _F_dim0++) {
            value_mask[_F_dim0] = _F_bs.sgetbits(8);
        }
        _F_dim0_end = (4*(length-8));
        for (_F_dim0 = 0; _F_dim0 < _F_dim0_end; _F_dim0++) {
            value_list[_F_dim0] = _F_bs.sgetbits(8);
```

```
            }
        }

        if ((opcode==2))
        {
            unusedByte = _F_bs.sgetbits(8);
            length = _F_bs.sgetbits(16);
            window = _F_bs.sgetbits(32);
            _F_dim0_end = 4;
            for (_F_dim0 = 0; _F_dim0 < _F_dim0_end; _F_dim0++) {
                value_mask[_F_dim0] = _F_bs.sgetbits(8);
            }
            _F_dim0_end = (4*(length-3));
            for (_F_dim0 = 0; _F_dim0 < _F_dim0_end; _F_dim0++) {
                value_list[_F_dim0] = _F_bs.sgetbits(8);
            }
        }
}}
```

Nous ajoutons la méthode `public abstract int execute() ;` qui est une méthode abstraite dans cette classe pour obtenir une classe abstraite représentant l'analyseur de PDU du protocole X. Ensuite, en étendant cette classe et en implémentant cette méthode abstraite avec les codes appropriés du *Multiplexeur X*, on obtient ce service, car tous les codes de Java dans la méthode `public int execute() ;` peuvent accéder aux variables de PDU définies dans la classe originale.

Voici un exemple du code de la méthode de contrôle des requêtes de ce protocole X écrit en Java.

```
public class X_PDU_Requete_Control extends X_PDU{

public int execute() {
   put(server1);  // envoie la requête au serveur 1;
   /* préparation la requête pour envoyer au serveur 2 */

    if (opcode ==1)
    {
      winid = CreateNewID(2,winid); //  créer un nouveau
                                    //identificateur
    }

    if (opcode ==2)
    {
      window = CreateNewID(2,window);
    }
   ...
   ...
   put(server2);  // envoie la même requête avec des changement au serveur 2;
```

```
} }
```

Grâce à cet exemple, nous démontrons comment on peut utiliser soit le langage *GateScript*, soit le langage Java pour écrire les codes de contrôle dans l'architecture générique des services actifs proposée dans le chapitre 4.

6.3 Conclusion

Dans ce chapitre nous avons évalué la performance de l'implémentation du module d'interception des paquets de ProAN sous Linux. Les résultats obtenus prouvent que les retards apparaissent seulement sur les flots de données sur lesquels les services actifs doivent effectuer un traitement utile : le retard pour les flots pour lesquels les paquets ne sont pas interceptés par le filtre de paquet reste petit même si les paquets d'un flot intercepté sont traités par un service actif.

La performance du moteur d'exécution de *GateScript* est comparable à celle d'un service écrit en langage Java. Dans cet environnement d'exécution, on peut aussi utiliser le langage Java pour écrire les codes de contrôle des services actifs en gardant la même propriété générique de l'architecture.

Chapitre 7

Conclusion

7.1 Bilan du travail réalisé

Dans cette thèse nous avons construit une passerelle active générique appelée ProAN qui peut supporter plusieurs environnements d'exécution des services actifs. ProAN laisse la liberté aux services actifs de choisir le bon moment pour injecter des filtres de paquets pour intercepter le flot des paquets ayant besoin de traitement. Les filtres de paquets peuvent impliquer des données de n'importe quelle portion du paquet IP - du niveau réseau au niveau application. ProAN est implémenté sous Linux et supporte pour l'instant trois environnements d'exécution dont *GateScript*. L'environnement de *GateScript* offre une architecture générique pour des services actifs. À partir de cette architecture, un service traitant un protocole peut être instancié. Cette architecture permet de séparer la fonction d'analyse et de formattage des PDUs de la fonction de traitement. Un langage de script appelé *GateScript* a été développé pour programmer les services actifs. Grâce à ce langage un utilisateur peut composer un service sans avoir à se préoccuper du code de l'analyseur et du générateur de PDUs. Nous avons aussi proposé d'utiliser le langage Flavor pour décrire les protocoles orientés bits, et JavaCC pour décrire les protocoles textuels. À partir du fichier de description, le générateur et l'analyseur du protocole sont créés automatiquement.

ProAN supporte aussi les services proactifs pour des environnements pervasifs. Ces services prennent en compte des changements dans l'environnement. Le service de découverte permet aux services proactifs de découvrir les autres services dans le réseau pour en bénéficier. Plusieurs scénarios d'applications ont été également présentés dans les différents contextes : la mobilité de l'utilisateur, le changement d'un périphérique, le travail coopératif. Tous ces exemples montrent le grand avantage de notre passerelle active : la possibilité de facilement étendre la fonctionnalité d'un protocole existant. L'avantage de généricité n'est pas obtenu au prix des performances diminuées - en effet nos expérimentations montrent que notre passerelle active obtient des performances comparables avec des passerelles spécialisées.

7.2 Perspectives

Au delà des contributions de ce travail, d'autres domaines de recherche doivent encore être explorés.

Par exemple dans le contrôle de ressource : pour l'instant, l'ordonnanceur de CPU de ProAN est celui du noyau Linux 2.4 original. Cet ordonnanceur, dit au mieux, ne garantit pas de qualité de service pour les services s'exécutant sur ProAN. On peut remplacer cet ordonnanceur par un autre qui garantit la qualité de service. Un parmi ceux-ci est celui *HFSQ* [109] qui peut partager le CPU entre les services selon des pourcentages prédéfinis.

Le portage de IPv6 sur ProAN devrait aussi être réalisé dans un futur proche. Nous essaierons d'intégrer le support de IPv6 sur la même architecture, pour que ProAN puisse supporter les paquets IPv6 et IPv4 en même temps.

On peut aussi utiliser les machines virtuelles Java avec la possibilité de contrôle de ressource comme Janos [54] pour implémenter l'environnement d'exécution *GateScript*. Un compilateur JIT (*Just in Time*) peut également être intégré avec *GateScript* pour obtenir une meilleure performance. Avec le langage *GateScript*, nous devons aussi tester et implémenter d'autres applications pour augmenter sa capacité comme la synchronisation entre des services.

L'application des réseaux actifs dans les environnements pervasifs ouvre encore de larges perspectives. D'autres scénarios d'application restent à imaginer et implémenter.

Chapitre 8

Annexe

8.1 Portage du ProAN sur un iPAQ

Nous avons déjà un iPAQ avec Linux et nous voulons compiler le ProAN (noyau avec des modifications, surtout avec un nouveau module *ip_queue*) et l'installer ensuite sur cet iPAQ.

Nous disposons aussi d'un PC avec Linux qui est utilisé comme plate-forme de compilation et passerelle entre l'iPAQ et l'Internet.

(`note : le mot "nguyenhb" dans le text suivant est mon login`).

8.1.1 Préparation des outils

A. Connecter l'iPAQ au PC sous Linux

La première chose est d'établir une connexion entre l'iPAQ et le PC par le protocole ppp [87].

A.1 Paramétrer l'iPAQ

1. Ajouter un nouvel utilisateur appelé "ppp" avec son shell /sbin/pppd

```
echo "ppp::101:101:user ppp:/home/ppp:/sbin/pppd" >> /etc/passwd
```

2. Créer un fichier de configuration /etc/ppp/options :

```
mkdir /ect/ppp
```

```
echo "-detach defaultroute noauth nocrtscts lock lcp-echo-interval
5 lcp-echo-failure 3 /dev/ttySA0 115200"  >> /etc/ppp/options
```

3. Ajouter les modules de ppp au fichier /etc/modules afin qu'ils soient chargés dans le noyau au moment de démarrage :

```
echo "slhc ppp_generic ppp_async" >> /etc/modules
```

Les commandes suivantes sont à exécuter si l'on ne veut pas redémarrer l'iPAQ :

```
insmod slhc insmod ppp_generic insmod ppp_async
```

A.2 Paramétrer le PC

Se connecter en tant que super-utilisateur (root) et exécuter la commande ci-dessous :

```
/sbin/pppd -detach debug /dev/ttyS0 115200 192.168.0.1:192.168.0.2
local noauth nocrtscts lock user ppp connect "/usr/sbin/chat -V
-t3 ogin--ogin: ppp"
```

Attention, il y a une petite espace entre ogin–ogin : et ppp

Si tout va bien, on a l'adresse de l'iPAQ est 192.168.0.2 et l'adresse du PC est 192.168.0.1.

Pour que l'iPAQ puisse connecter à l'Internet, il faut exécuter les commandes suivante sur le PC :

```
modprobe ip_tables iptables -t nat -A POSTROUTING -s 192.168.0.2
-j MASQUERADE echo 1> /proc/sys/net/ipv4/ip_forward
```

Sur l'iPAQ on doit paramètrer le routeur par défaut, celui étant le PC :

```
route add default gw 192.168.0.1
```

Configurer le service DNS sur l'iPAQ. Vous pouvez copier le fichier /etc/resolv.conf du PC à l'iPAQ. Modifier le fichier /etc/hosts sur l'iPAQ et sur le PC si vous voulez utiliser les noms plutôt l'adresse de l'iPAQ et le PC.

Faire un ping pour vérifier si tout marche bien !

B. Installer l'outil cross-compile toolchain [85] sur le PC

L'outil toolchain est un compilateur dit "cross-platform" qui permet la compilation sur la plate-forme Intel i386 des codes C écrits pour une autre plate-forme dans ce cas l'iPAQ avec le processeur Intel StrongARM SA-1110 [88].

Copier l'outil toolchain déjà compilé pour exécuter sur l'i386 au
ftp ://ftp.handhelds.org/pub/linux/arm/toolchain/arm-linux-toolchain-post-2.2.13.tar.gz

et le mettre sous le répertoire /skiff/local/ car celui-ci ne marche que dans ce répertoire.

Copier aussi la librairie libz au ftp ://ftp.handhelds.org/pub/linux/arm/toolchain/arm-linux-libz.tar.gz

C. Installer l'ipkg sur PC

L'outil ipkq [89] permet de compresser et de gérer les modules du noyau déjà compilés sur le PC pour ensuite les télécharger sur l'iPAQ.

Copier l'outil ipkg sur le PC au : ftp ://lorien.handhelds.org/pub/linux/feeds/demo/ipkg-tools.tgz Décompresser en utilisant :

```
tar -xvfz ipkg-tools.tgz
```

On obtient les outils suivants : "ipkg" , "ipkg-build", "ipkg-make-index" et "ipkg-make-kernel_packages". Mais on va utiliser principalement "ipkg-build" pour construire les packages des modules du noyau pour l'iPAQ. Ajouter le chemin de ces outils dans le variable de l'environnement $PATH pour pouvoir les accéder plus facilement.

8.1.2 Préparation du ProAN pour l'iPAQ

Nous utilisons le noyau Linux 2.4.18 pour construire un noyau pour l'iPAQ. Ensuite nous remplaçons le module *ip_queue* par notre nouveau module *ip_queue* pour obtenir le ProAN pour l'iPAQ.

Un noyau Linux pour l'iPAQ est composé de trois sources :

1) le noyau original de Linux

2) le Patch de ftp.arm.linux.org.uk

3) le Patch de ftp.handhelds.org.

Donc pour avoir le noyau version 2.4.18 pour l'iPAQ, on doit au préalable télécharger ces trois sources et ensuite appliquer les Patches.

De manière plus précise, sur le PC appliquer les commandes suivantes :

```
bash$ wget
ftp://ftp.kernel.org/pub/linux/kernel/v2.4/linux-2.4.18.tar.gz

bash$ wget
ftp://ftp.arm.linux.org.uk/pub/armlinux/kernel/v2.4/patch-2.4.18-rmk3.gz

bash$ wget
ftp://ftp.handhelds.org/pub/linux/kernel/patch-2.4.18-rmk3-hh20.gz

bash$ tar -zxvf linux-2.4.18.tar.gz

bash$ mv linux linux-2.4.18-iPAQ bash$ cd linux-2.4.18-iPAQ

bash$ zcat ../patch-2.4.18-rmk3.gz | patch -p1

bash$ zcat ../patch-2.4.18-rmk3-hh20.gz | patch -p1
```

Fixer le bug du noyau linux 2.4.18 en modifiant tous les codes (il y en a deux à la ligne 109 et la ligne 209) hashfn(pos) en HASHFN(pos) dans le fichier ghash.h dans le répertoire

linux-2.4.18-iPAQ/include/linux

Ajouter un # devant la variable ARCH à la ligne 9 dans le fichier linux-2.4.18-iPAQ/Makefile

À la ligne 24, dans le fichier linux-2.4.18-iPAQ/Makefile, il faut refixer la valeur du variable CROSS_COMPILE en /skiff/local/bin/arm-linux-

On obtient : CROSS_COMPILE = /skiff/local/bin/arm-linux-

Maintenant dans le répertoire linux-2.4.18-iPAQ, on doit remplacer le module *ip_queue* par le nouveau module *ip_queue* du ProAN et on obtient le noyau ProAN pour l'iPAQ.

8.1.3 Compilation et installation du ProAN sur l'iPAQ

Copier le fichier linux-2.4.18-iPAQ/arch/arm/def-configs/h3600 au répertoire linux-2.4.18-iPAQ/. et changer son nom en ".config" (attention c'est le point config (.config)).

Faire la commande : make oldconfig

Ensuite la commande : make menuconfig

Entrer dans "*IP :Netfilter Configuration*" dans "*Networking options*" dans le menu principal. Garder les options pré-choisies mais choisir de plus les options suivantes :

```
<M> Packet type match support
<M> REJECT target support
```

Sortir et choisir "*save*" pour enregistrer les options de configuration.

Ensuite faire la commande : make dep

Ensuite la commande : make zImage

Ensuite la commande : make modules

On a l'image du noyau au linux-2.4.18-iPAQ/arch/arm/boot/zImage.

Par précaution, on renomme cette image en zImage-ProAN et ensuite la copie dans le répertoire /boot/ de l'iPAQ. Pour copier un fichier du PC à l'iPAQ, on place le fichier dans un FTP server et ensuite sur l'iPAQ utiliser l'outil sftp pour connecter au FTP server et le télécharger.

Pour installer les modules, les indications suivantes utilisent l'outil ipkg.

8.1.4 Installation des modules du ProAN sur l'iPAQ

Créer un répertoire par exemple ProAN-modules. J'utilise mon home répertoire c'est à dire j'ai le répertoire /home/nguyenhb/ProAN-modules pour installer temporairement les modules avant les télécharger sur l'iPAQ.

Faire la commande suivante dans le répertoire linux-2.4.18-iPAQ/

```
make INSTALL_MOD_PATH=/home/nguyenhb/ProAN-modules modules_install
```

Cela va copier les modules pour l'iPAQ dans le répertoire : /home/nguyenhb/ProAN-modules

Attention, il faut donner le chemin complet de type /home/... à la variable INSTALL_MOD_PATH comme dessus, il ne faut pas donner un chemin relatif de type ~nguyenhb/ProAN-modules.

Les fichiers des modules sont copiés au sous-répertoire lib au répertoire ProAN-modules. Ensuite, il faut changer le propriétaire de ces modules en root car ils vont être transférés à l'iPAQ.

Changer au mode de super-utilisateur (root) :

```
bash$ su
```

Changer le propriétaire de ces modules au super-utilisateur par :

```
bash$ cd /home/nguyenhb/ProAN-modules
bash$ chown -R root:root lib
```

Ensuite, il faut faire les tâches suivantes :

1)Entrer dans le répertoire ProAN-modules/lib/modules/2.4.18-ProAN/

2)Effacer le fichier modules.dep

3)Créer un lien entre /var/run/modules.dep et modules.dep dans ce répertoire

```
bash$ cd ProAN-modules/lib/modules

bash$ rm modules.dep

bash$ ln -s/var/run/modules.dep modules.dep
```

Si les tâches ci-dessus ne sont pas faites - le noyau ne marchera pas et ne démarrera pas sur l'iPAQ !

On a fini de faire du package des modules du ProAN avec l'ipkg. Quitter le mode super-utilisateur et entrer dans le répertoire ProAN-modules

Créer un sous-répertoire CONTROL

```
bash$ cd ProAN-modules bash$ mkdir CONTROL
```

Dans le répertoire CONTROL créer le fichier qui s'appelle : "control" dont le contenu est le suivant :

```
Package: ProAN
Version: 2.4.18
Architecture: Arm
Maintainer : nguyenhb
Description : ProAN pour l'iPAQ
```

Ensuite sortir du répertoire ProAN-modules et executer la commande ipkg-build pour enveloper les modules du ProAN :

```
bash$ ipkg-build  ProAN-modules
```

on obtient le fichier ProAN_2.4.18_Arm.ipk

Télécharger ce fichier à l'iPAQ et ensuite sur l'iPAQ. Executer la commande suivante pour décompresser ce fichier et installer les modules :

```
ipkg install ProAN_2.4.18_Arm.ipk
```

(Attention, cette commande s'exécute sur l'iPAQ)

Sur l'iPAQ, entrer au répertoire /boot/, créer un lien entre zImage et zImage-ProAN

```
rm zImage
ln -s zImage-ProAN zImage
```

Redémarrer et on obtient le ProAN sur l'iPAQ.

8.2 Code du nouveau module ip_queue

Ce code est très long - veuillez le consulter à l'adresse :
http ://drakkar.imag.fr/~nguyenhb/Netfilter/ip_queue.c
http ://drakkar.imag.fr/~nguyenhb/Netfilter/ip_queue.h

8.3 Code de la librairie JavaLipipq

8.3.1 Code de l'interface JNI JavaLipipq

```
/*
 JavaLibipq.java  :
 Author : Nguyen Hoa Binh        May 2002
          hoa-binh.nguyen@imag.fr
*/

import java.io.*; import java.lang.*;


public class JavaLibipq{

    public static int IPQ_COPY_NONE = 0;
    public static int IPQ_COPY_META = 1;
    public static int IPQ_COPY_PACKET = 2;
    public static int IPQ_PASS_PACKET = 3;

    public static int NF_DROP   = 0;
    public static int NF_ACCEPT = 1;
    public static int NF_STOLEN = 2;
    public static int NF_QUEUE = 3;
    public static int NF_REPEAT = 4;
    public static int NF_INJECT = 5;



 public static ipq_packet_msg ipq_get_packet(byte[] buf) throws LibipqException
    {
    boolean LSB = true;
    int buf_len,len;
    String tmp;

    buf_len = buf.length;

    ipq_packet_msg  msg = new ipq_packet_msg();

    if (buf_len < 69) throw new LibipqException("1: kernel message is bad !");
    msg.packet_id = build_int(buf[0],buf[1],buf[2],buf[3],LSB);

    msg.mark      = build_int(buf[4],buf[5],buf[6],buf[7],LSB);

    msg.timestamp_sec  = build_int(buf[8],buf[9],buf[10],buf[11],LSB);
    msg.timestamp_usec = build_int(buf[12],buf[13],buf[14],buf[15],LSB);
    msg.hook           = build_int(buf[16],buf[17],buf[18],buf[19],LSB);
    msg.indev_name = new byte[16];
        System.arraycopy(buf,20,msg.indev_name,0,16);
```

```
    msg.outdev_name = new byte[16];
        System.arraycopy(buf,36,msg.outdev_name,0,16);
    msg.hw_protocol = (short)build_short(buf[52],buf[53],LSB);

    msg.hw_type     = (short)build_short(buf[54],buf[55],LSB);
    msg.hw_addrlen  = buf[56];
    msg.hw_addr = new byte[8];
    System.arraycopy(buf,57,msg.hw_addr,0,8);


    msg.data_len = build_int(buf[65],buf[66],buf[67],buf[68],LSB);

    if (msg.data_len > 0)
        {

        if (msg.data_len > buf_len - 69)
        throw new  LibipqException("2 :kernel message is bad !");
        msg.payload = new byte[msg.data_len];
        System.arraycopy(buf,69,msg.payload,0,msg.data_len);
        }
    return msg;



    }


 public static   void setIPchecksum(ipq_packet_msg msg){
        byte[] b = msg.payload;
        int IHL = (int)(msg.payload[0] & 0x0f) * 4; // IP header length

        // mark checksum field zero
        b[10] = (byte) 0x00;
        b[11] = (byte) 0x00;

        // add up header's words
        long sum = 0;
        for (int i = 0; i < IHL; i += 2 ){
            long s = ((b[i] << 8) & 0xFF00) + (b[i+1] & 0xFF);
            sum += s;
        }

        //
        while ((sum >> 16) != 0){
            sum = (sum & 0xFFFF) + (sum >> 16);
        }
        // make complement
        sum = ~sum;

        // mark checksum field
        b[11] = (byte) (sum & 0xFF);
        b[10] = (byte) ((sum >> 8) & 0xFF);
    }
```

```
   public static int build_int(byte b1, byte b2, byte b3, byte b4, boolean LSB)
    {
    int s = 0;

    if (LSB==true)
        {
                s=b1&0xff;
        s|=((b2&0xff)<<8);
        s|=((b3&0xff)<<16);
        s|=((b4&0xff)<<24);
        }
        else   // (LSB==false)
        {
                s=b1&0xff;
        s=((s<<8)&0xffff)|(b2&0xff);
        s=((s<<8)&0xffffff)|(b3&0xff);
        s=(s<<8)|(b4&0xff);
        }
    return s;
    }

    public static int build_short(byte b1, byte b2, boolean LSB)
    {
        int s = 0;

        if (LSB)
        {
         s=b1&0xff;
         s |=((b2&0xff)<<8);
        }
        else
         {
            s = b1&0xff;
            s =((s<<8)&0xffff)|(b2&0xff);
        }

        return s;
    }

}
```

8.3.2 Code de l'implémentation de l'interface JNI JavaLibipq en C

```
/*
 JavaLibipq.c  : Java Native Interface for libipq
 Author : Nguyen Hoa Binh        May 2002
```

```
          hoa-binh.nguyen@imag.fr
 compile :
 gcc -I/usr/java/j2sdk1.4.0/include/linux JavaLibipq.c
 /home/nguyenhb/iptables-new/libipq/libipq.a -shared -o libJavaLibipq.so */


#include "QueueHandlerJava.h"
#include <stdio.h>
#include<sys/types.h>
#include <limits.h>
#include <net/if.h>
#include <netinet/ip.h>
#include <stdio.h>

#include <linux/netfilter_ipv4.h>

#include "/home/nguyenhb/linux-2.4.16/include/linux/netfilter.h"
#include"/home/nguyenhb/linux-2.4.16/include/linux/netfilter_ipv4/ip_queue.h"
#include "/home/nguyenhb/iptables-new/include/libipq/libipq.h"



struct ipq_handle *h; unsigned char packet[65536];

JNIEXPORT void JNICALL Java_QueueHandlerJava_ipqHelloWorld(JNIEnv
*env, jobject obj)
         {
             printf("Hello world!\n");

             return;
         }


JNIEXPORT jint JNICALL
Java_QueueHandlerJava_ipqCreateHandle(JNIEnv *env, jobject obj,
jint mode) { if ((h = ipq_create_handle(mode)) == NULL)
  return -1;
 else return 1;
}

JNIEXPORT jint JNICALL Java_QueueHandlerJava_ipqSetMode(JNIEnv
*env, jobject obj, jint handle
                   , jint mode, jint bufsize)
{ return ipq_set_mode(h,mode,bufsize); }

JNIEXPORT  jbyteArray JNICALL Java_QueueHandlerJava_ipqRead(JNIEnv
*env, jobject obj) {

  jbyteArray jb;
  int ptr;
  unsigned char *buf;

  read :
```

```
 ipq_read(h,packet,sizeof(packet),0);
 if (ipq_message_type(packet) == IPQM_PACKET)
   {
     ipq_packet_msg_t *msg = ipq_get_packet(packet);

     buf = (unsigned char *)malloc(65355);
     memcpy(buf,&(msg->packet_id),sizeof(msg->packet_id));
     ptr = sizeof(msg->packet_id);
     memcpy(buf+ptr,&(msg->mark),sizeof(msg->mark));
     ptr += sizeof(msg->mark);
     memcpy(buf+ptr,&(msg->timestamp_sec),sizeof(msg->timestamp_sec));
     ptr += sizeof(msg->timestamp_sec);
     memcpy(buf+ptr,&(msg->timestamp_usec),sizeof(msg->timestamp_usec));
     ptr += sizeof(msg->timestamp_usec);
     memcpy(buf+ptr,&(msg->hook),sizeof(msg->hook));
     ptr += sizeof(msg->hook);
     memcpy(buf+ptr,&(msg->indev_name),16);
     ptr += 16;
     memcpy(buf+ptr,&(msg->outdev_name),16);
     ptr += 16;
     memcpy(buf+ptr,&(msg->hw_protocol),sizeof(msg->hw_protocol));
     ptr += sizeof(msg->hw_protocol);
     memcpy(buf+ptr,&(msg->hw_type),sizeof(msg->hw_type));
     ptr += sizeof(msg->hw_type);
     memcpy(buf+ptr,&(msg->hw_addrlen),sizeof(msg->hw_addrlen));
     ptr += sizeof(msg->hw_addrlen);
     memcpy(buf+ptr,&(msg->hw_addr),8);
     ptr += 8;
     memcpy(buf+ptr,&(msg->data_len),sizeof(msg->data_len));
     ptr += sizeof(msg->data_len);
     if (msg->data_len >0)
     memcpy(buf+ptr,&(msg->payload),msg->data_len);
     ptr += msg->data_len;


     jb = (*env)->NewByteArray(env,ptr);
    (*env)->SetByteArrayRegion(env,jb,0,ptr,(jbyte *)buf);
    return jb;
   }
 else
   goto read;

}

JNIEXPORT void JNICALL Java_QueueHandlerJava_ipqSetVerdict (JNIEnv
*env, jobject obj, jint handle, jint id, jint verdict, jint
data_len, jbyteArray packet) { jsize len; jbyte *payload;

len = (*env)->GetArrayLength(env, packet); payload =
(*env)->GetByteArrayElements(env,packet,0);
ipq_set_verdict(h,id,verdict,len,payload);
 (*env)->ReleaseByteArrayElements(env,packet,payload,0);
 return;
```

```
}

JNIEXPORT void JNICALL Java_QueueHandlerJava_ipqDestroyHandle
  (JNIEnv *env, jobject obj, jint handle)
{ if (h) {
        close(h->fd);
        free(h);
    }
    return ;
}
```

8.4 Grammaire du GateScript

```
GateScriptProgram  = Statements

Statements = ([static] Statement ‘‘;’’)*

Statement = AssignStatement
        | IfStatement
        | FunctionStatement
        | OnEventStatement

AssignStatement = "set" Variable  Expression

IfStatement = ‘‘if’’ Expression ‘‘then’’ Statements
[ElseStatement] ‘‘endif’’

ElseStatement = ‘‘else’’ Statements

OnEventStatement =‘‘onEvent’’ Expression ‘‘then’’ Statements
‘‘endEvent’’

FunctionStatement = FunctionName (Expression)*

Expression  = ConstantValue
              | ‘‘$‘‘Variable
              | Expression BinOp Expression
              | UnOp Expression
              | ‘‘[‘‘ FunctionStatement ‘‘]‘‘
              | ‘‘(‘‘ Expression ‘‘)’’

Variable     = Identifier
FunctionName = Identifier

Identifier = Letter (Letter | Digit)*

ConstantValue   = Boolean | String | Integer |
```

```
                Real | Character
BinOp       = ``+'' | ``-'' | ``*'' | ``/''
              | ``<'' | ``<='' | ``='' | ``!=''
              | ``>'' | ``>=''
              | ``and'' | ``or'' | ``contains''
UnOp        = ``-'' | ``+'' | ``!''

Boolean     = ``true'' | ``false''

String      = ``"''  (~[``"'',``\'',``\n'',
              ``\r'',"[","]"])* ``"''
Integer     = Digit (Digit)*

Real        = Integer [Fraction] [Exponent]

Fraction    = ``.'' Integer

Exponent    = (``e'' | ``E'') [``+'' | ``-''] Integer

Digit       = ["0"-"9"]

Letter      = ["a"-"z", "A"-"Z"] | "_"
```

8.5 Structure du paquet IP/TCP décrite en Flavor

```
class TCP_IP {

unsigned int(4) version;
unsigned int(4) hdr_length;

unsigned int(8) service_type;
unsigned int(16) total_length;
unsigned int(16) identification;
unsigned int(3)  flags;

unsigned int(13) fragment_offset;
unsigned int(8)  ttl;

unsigned int(8) protocol;
unsigned int(16) header_checksum;
unsigned int(32) source_address;

unsigned int(32) destination_address;

if (hdr_length>5) { unsigned int(8) options[(hdr_length*4-20)]; }

unsigned int(16) source_port;
unsigned int(16) destination_port;
unsigned int(32) sequence_number;
unsigned int(32) ack_number;
```

```
unsigned int(4) data_offset;
unsigned int(6) reserved;

unsigned int(1) URG;
unsigned int(1) ACK;
unsigned int(1) PSH;
unsigned int(1) RST;
unsigned int(1) SYN;
unsigned int(1) FIN;
unsigned int(16) window;
unsigned int(16) TCP_Checksum;

unsigned int(16) urgent_pointer;

if (Data_Offset>5)
  {
    unsigned int(8) TCP_options[(Data_Offset-5)*4];
  }

unsigned int(8) data[total_length-(hdr_length*4)
                     -(Data_Offset*4)];
};
```

Bibliographie

[1] Internet Protocol Version 4 (IPv4) Specification. Internet Standard RFC 791.

[2] Internet Protocol Version 6 (IPv6) Specification. Internet Standard RFC 2460.

[3] Integrated Services (intserv).
Disponible sur : http ://www.ietf.org/html.charters/intserv-charter.html

[4] BLAKE S., BLACK D., CARLSON M. et al. An Architecture for Differentiated Services. Internet RFC 2475, 1998.

[5] KINDBERG T., FOX A. System Software for Ubiquitous Computing. IEEE Pervasive Computing, 1(1) :70-81, 2002.

[6] WETHERALL D., LEGEDZA U., GUTTAG J. Introducing New Internet Services : Why and How. IEEE Network Magazine, July 1998.

[7] CAMPBELL A.T., DE MEER H.G., KOUNAVIS M.E. Kounavis et al. A Survey of Programmable Networks. ACM Computer Communications Review, April 1999.

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

[9] OPENSIG : Open Signalling and Service Creation.
Disponible sur : http ://comet.columbia.edu/opensig/

[10] BISWAS J. et al. The IEEE P1520 Standards Initiative for Programmable Network Interfaces. IEEE Communications Magazine, Vol. 36, pp. 64-72, IEEE, October 1998.

[11] ALEXANDER S., SHAW M., NETTLES S. M., SMITH J. M. Active Bridging. In Proceedings of SIGCOMM97, September 1997.

[12] DARPA Active Network Research Groups. Disponible sur : http ://www.darpa.mil/ito/ResearchAreas/ActiveNetsList.html

[13] WETHERALL D., GUTTAG J., TENNENHOUSE D. ANTS : A Toolkit for Building and Dynamically Deploying Network Protocols. IEEE Openarch'98, San Francisco, CA, April 1998.

[14] SCHMID S., SHEPHERD D. Ease of Network Programmability Based on Active LARA++ Components. In IEE Workshop on Application Layer Active Networks : Techniques and Deployment, London, UK, November 2000.

[15] IEEE. Media Access Protocol (MAC) Bridges. Technical Report ISO/IEC 10038, ISO/IEC, 1993.

[16] McCLOGHRIE K., ROSE M. Management Information Base for Network Management of TCP/IP-based Internets. RFC 1066, TWG, August 1988.

[17] RIVEST R. The MD5 Message-Digest Algorithm. Internet RFC 1321, April 1992.

[18] POSTEL J. Internet Protocol - DARPA Internet Program Protocol Specification. Internet RFC 791, September 1981.

[19] POSTEL J. Transmission Control Protocol. Internet RFC 793. September 1981.

[20] BERNERS-LEE T., FIELDING R., FRYSTYK H. Hypertext Transfer Protocol - HTTP/1.0. Internet RFC 1945 MIT/LCS, UC Irvine, May 1996.

[21] BERNERS-LEE T., MASINTER L., McCAHILL M. Uniform Resource Locators (URL). RFC 1738, CERN, Xerox Corporation, University of Minnesota, December 1994.

[22] TCP SYN Flooding and IP Spoofing Attacks. CERT Advisory CA-1996-21 Original issue date : September 19, 1996.
Disponible sur : http ://www.cert.org/advisories/CA-1996-21.html

[23] LEMON J. Resisting SYN Flooding DoS Attacks with a SYN Cache. Proceedings of USENIX BSDCon2002, February, 2002.

[24] BERNSTEIN Daniel. SYN cookies. Disponible sur : http ://cr.yp.to/syncookies.html

[25] FERGUSON P., SENIE D. Network Ingress Filtering : Defeating Denial of Service Attacks which employ IP Source Address Spoofing. Internet RFC 2827, May 2000.

[26] ANEP - Active Network Encapsulation Protocol. Disponible sur : http ://www.cis.upenn.edu/ switchware/ANEP/

[27] CALVERT K.L. Architectural Framework for Active Networks. University of Kentucky, July 1999.
Disponible sur : http ://www.cc.gatech.edu/projects/canes/papers/arch-1-0.ps.gz

[28] PETERSON Larry. Node OS and Interface Specification. January 2000. Disponible sur : http ://www.cs.princeton.edu/nsg/papers/nodeos.ps

[29] YEMINI Y. Active Network Management. Disponible sur : http ://www.cs.columbia.edu/dcc/anm

[30] MERUGU S., BHATTACHARJEE S. et al. Bowman : A Node OS for Active Networks. IEEE Infocom 2000, Tel Aviv, March 2000.

[31] SCHWARTZ B. et al. Smart Packets : Applying Active Networks to Network Management. ACM Transactions on Computer Systems, Vol. 18, No. 1, February 2000.

[32] FABER T. Acc : Using Active Networking to Enhance Feedback Congestion Control Mechanisms. May 1998.

[33] BHATTACHARJEE S. On Active Networking and Congestion. Tech. Rep. GIT-CC-96-02, Georgia Institute of Technology, Atlanta, Georgia, 1996.

[34] BHATTACHARJEE S., McKINNON M. W. Performance of Application-Specific Buffering Schemes for Active Networks. Georgia Institute of Technology, Atlanta, Georgia, 1998.

[35] HARBAUM T., SPEER A., WITTMANN R., ZITTERBART M. AMnet : Efficient Heterogeneous Group Communication Through Rapid Service Creation. Proceedings of The Active Middelware Workshop AMS'00, Pittsburg, USA, August, 2000.

[36] KASERA K., BHATTACHARJEE S. et al. Scalable Fair Reliable Multicast Using Active Services. IEEE Network Magazine (Special on Multicast), Jan./Feb. 2000.

[37] LEGEDZA U., GUTTAG J. Using Network Level Support to Improve Cache Routing. in the Proceedings of the 3rd International WWW Caching Workshop June, 1998.

[38] BHATTACHARJEE S., CALVERT K., ZEGURA E. Congestion Control and Caching in CANES. Proceedings of ICC '98, Atlanta, GA, 1998.

[39] BHATACHARJEE S., CALVERT K. L., ZEGURA E. Self-organizing wide-area network caches. Georgia Institute of Technology GIT-CC-97/31, 1997.

[40] Content Delivery and Distribution Services.
Disponible sur : http ://www.web-caching.com/cdns.html

[41] LEGEDZA U., WETHERALL D., GUTTAG J. Improving the Performance of Distributed Applications Using Active Networks. IEEE INFOCOM, San Francisco, April 1998.

[42] MAXEMCHUK N. F. Active Networks in Telephony. OPENARCH'99, NY, USA, March 26-27, 1999.

[43] Next Generation Networks Initiative. Disponible sur : http ://www.ngni.org/

[44] VAN C. V. A defense Against Address Spoofing Using Active Netwoks. Master Thesis of EEE & Computer Sciences, MIT, May 1997.

[45] PREHOFER C., WEI Q. Active Networks for 4G Mobile Communication : Motivation, Architecture, and Application Scenarios. in Proceedings of the 4th International Working Conference, IWAN2002, Zurich, Switzerland, December 2002.

[46] BOULIS A., LETTIERI P., SRIVASTAVA M.B. Active Base Stations and Nodes for a Mobile Environment. ACM 1998.

[47] DECASPER Dan et al. Router Plugins : A Software Architecture for Next Generation Routers. in Proceedings of the ACM SIGCOMM '98, Vancouver Canada, September 1998.

[48] MARESCA R., ARIENZO M. et al. An Active Network approach to Virtual Private Networks. in Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02), Taormina/Giardini Naxos, Italy, July 1 - 4, 2002.

[49] TSCHUDIN C., LUNDGREN H., GULBRANDSEN H. Active Routing for Ad Hoc Networks. IEEE Communications Magazine, April 2000.

[50] FOSTER I., KESSELMAN C., TUECKE S. The Anatomy of the Grid : Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 2001.

[51] LEFEVRE L., PHAM C.D., PRIMET P., et al. Active Networking Support for The Grid. in Proceedings of the third International Working Conference on Active Networks (IWAN'01), September 30 and October 1-2 2001, Philadelphia, USA, pp16-33.

[52] HARTMAN J., PETERSON L. et al. Experiences Building a Communication-Oriented JavaOS. Software-Practice and Experience, 2000.

[53] MOSBERGER D. Scout : A Path-based Operating System. Ph.D. Dissertation, Department of Computer Science, University of Arizona, July 1997.

[54] TULLMANN P., HIBLER M., LEPREAU J. Janos : A Java-oriented OS for Active Networks. in IEEE Journal on Selected Areas of Communication. Volume 19, Number 3, March 2001.

[55] SHALABY N., GOTTLIED Y. et al. Snow on Silk : A NodeOS in the Linux Kernel. in Proceedings of the 4th International Working Conference, IWAN2002, Zurich, Switzerland, December 2002.

[56] KELLER R., RUF L. et al. PromethOS : A Dynamically Extensible Router Architecture Supporting Explicit Routing. in Proceedings of the 4th International Working Conference, IWAN2002, Zurich, Switzerland, December 2002.

[57] LARRABEITI D., CALDERON M., AZCORRA A., URUENA M. A Practical Approach to Network-based Processing. IEEE 4th Internacional Workshop on Active Middleware Services, Edinburgh, Scotland, 23 July 2002.

[58] GELAS J.P., HADRI S.E, LEFEVRE L. Tamanoir : a Software Active Node Supporting Gigabit Networks. in Proceedings of the Second International Workshop on Active Network technologies and Applications, ANTA2003, May 28-30, 2003, Osaka, Japan.

[59] AMIR E., McCANNE S., KATZ R. An Active Service Framework and its Application to Real-time Multimedia Transcoding. ACM Communication Review, vol. 28, no. 4, pp. 178-189, Sep. 1998.

[60] FRY M., GHOSH A. Application Level Active Networking. Computer Networks 1999.

[61] Network Processing Forum. Disponible sur : http ://www.npforum.org/

[62] PEYRAVIAN M., CALVIGNAC J. Fundamental Architectural Considerations for Network Processors. Computer Networks, col. 41, no.5, pp.587-600, April 5, 2003.

[63] ADILETTA M., ROSENBLUTH M. et al. The Next Generation of Intel IXP Processors. Intel Technology Journal, August 2002.

[64] ALLEN J., BASS B. et al. PowerNP Network Processor : Hardware, Software and Applications. IBM Journal of Research and Development, (to appear 2003).

[65] NYGREN E., GARLAND S., KAASHOEK M. F. PAN : A High-Performance Active Network Node Supporting Multiple Mobile Code Systems. the second IEEE Conference on Open Architectures and Network Programming, OpenArch99, New York, New York, March 1999.

[66] SCHMID S., CHART T., SIFALAKIS M., SCOTT A. C. Flexible, Dynamic and Scalable Service Composition for Active Routers. IWAN02 - Fourth Annual International Working Conference on Active Networks, ETH Zrich, Switzerland, December 4-6, 2002.

[67] YEMINI Y., SILVA S. Towards Programmable Networks. IFIP/IEEE International Workshop on Distributed Systems : Operations and Management, L'Aquila, Italy, October, 1996.

[68] SILVA S. Programming in the Netscript Toolkit. DCC Laboratory, Columbia University, September 1998.

[69] HICKS M. et al. PLAN : A Programming Language for Active Networks. In Proc. ICFP'98, 1998. Disponible sur : http ://www.cis.upenn.edu/ switchware/PLAN/

[70] NGUYEN Hoa-Binh, DUDA Andrzej. ProAN : an Active Node for Proactive Services in Pervasive Environments. in Proc. 2nd International Workshop on Active Network Technologies and Applications (ANTA 2003), May 2003, Osaka, Japan.

[71] Autonomic Computing : IBM's perspective on the State of Information Technology. Disponible sur : http ://researchweb.watson.ibm.com/autonomic/

[72] TENNENHOUSE D. Proactive Computing. CACM, May 2000.

[73] The netfilter-iptables project. Disponible sur : http ://netfilter.samba.org

[74] Iptables - U32 Match Tutorial.
Disponible sur : http ://www.stearns.org/doc/iptables-u32.v0.1.html

[75] BALAKRISHNAN H., SESHAN S., KATZ R. H. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. ACM Wireless Networks, V 1, N 4,December 1995.

[76] ELEFTHERIADIS A., HONG D. Flavor : a language for media representation. in Proc. of the fifth ACM international conference on Multimedia 1997, Seattle, Washington.

[77] Java Compiler Compiler (JavaCC) - The Java Parser Generator. Disponible sur : http ://www.webgain.com/products/java_cc/

[78] MOGUL Jeffrey C. Server-Directed Transcoding. in Proc. 5th International Web Caching and Content Delivery Workshop, 2000.

[79] WEI Tsang Ooi et al. Design And Implementation Of Programmable Media Gateways. NOSSDAV 2000 - The 10th International Workshop on Network and Operating System Support for Digital Audio and Video. June 26-28 2000 Chapel Hill, North Carolina.

[80] The IP Network Address Translator (NAT) - Internet standard RFC 1631.

[81] Man pages of LIBIPQ. Disponible sur : http ://www.cs.princeton.edu/ nakao/libipq.htm

[82] ip_queue module for IPv6.
Disponible sur : http ://www.iglu.org.il/lxr/source/net/ipv6/netfilter/ip6_queue.c

[83] Linux kernel capabilities FAQ. Disponible sur : http ://www.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.2/capfaq-0.2.txt

[84] IP Queue Multiplex Daemon (ipqmpd).
Disponible sur : http ://gnumonks.org/projects/

[85] README-Toolchain.
Disponible sur : ftp ://ftp.handhelds.org/pub/linux/arm/toolchain/README

[86] The Linux iPAQ HOWTO Homepage.
Disponible sur : http ://mstempin.free.fr/linux-ipaq/

[87] The Point-to-Point Protocol (PPP) - Internet Standard RFC 1661. Disponible sur : http ://www.ietf.org/rfc/rfc1661.txt ?number=1661

[88] Intel StrongARM processor.
Disponible sur : http ://www.intel.com/design/pca/applicationsprocessors/1110_brf.htm

[89] Buildings ipkgs. Disponible sur : http ://www.handhelds.org/z/wiki/BuildingIpkgs

[90] Netlink Socket Introduction.
Disponible sur : http ://qos.ittc.ukans.edu/netlink/html/node3.html

[91] Jini Network Technology. Disponible sur : http ://wwws.sun.com/software/jini/

[92] A JINI Protocol Library for Non-Java Capable Service Providers. Disponible sur : http ://user-hanno.jini.org/

[93] UPnP Forum. Disponible sur : http ://www.upnp.org/

[94] Salutation Consortium. Disponible sur : http ://www.salutation.org/

[95] DNS Service Discovery (dns-sd). Disponible sur : http ://www.dns-sd.org/

[96] Service Location Protocol - Internet RFC 2165. Disponible sur : http ://www.ietf.org/rfc/rfc2165.txt

[97] Bluetooth Wireless Technology. Disponible sur : https ://www.bluetooth.com/

[98] Zero Configuration Networking (Zeroconf). Disponible sur : http ://www.zeroconf.org/

[99] Java Naming and Directory Interface (JNDI). Disponible sur : http ://java.sun.com/products/jndi/

[100] Java Remote Method Invocation. Disponible sur : http ://java.sun.com/products/jdk/rmi/

[101] The Dynamic Host Configuration Protocol (DHCP). Disponible sur : http ://www.dhcp.org/

[102] Infrared Data Association (IrDA). Disponible sur : http ://www.irda.org/

[103] JRendezvous. Disponible sur : http ://www.strangeberry.com/java_rendevous.htm

[104] Java Media Framework API.
Disponible sur : http ://java.sun.com/products/java-media/jmf/

[105] Muffin : a World Wide Web Filtering System. Disponible sur : http ://muffin.doit.org/

[106] The X Protocol. Disponible sur : http ://www.x.org/X11_protocol.html

[107] Tcl (Tool Command Language). Disponible sur :http ://www.tcl.tk/software/tcltk/

[108] Tcl Java Integration. Disponible sur : http ://www.tcl.tk/software/java/

[109] QLinux 2.4.x : A QoS enhanced Linux Kernel for Multimedia Computing. Disponible sur : http ://lass.cs.umass.edu/software/qlinux/

Communications produites

– NGUYEN Hoa-Binh, DUDA Andrzej. An Active Node Architecture for Proactive Services. paru à la session de poster au IWAN 2002 - Fourth Annual International Working Conference on Active Networks, December 4-6, 2002, ETH Zürich, Switzerland.

– NGUYEN Hoa-Binh, DUDA Andrzej. ProAN : an Active Node for Proactive Services in Pervasive Environments. paru au procedings of the 2nd International Workshop on Active Network Technologies and Applications (ANTA 2003), May 2003, Osaka, Japan

– NGUYEN Hoa-Binh, DUDA Andrzej. Generic Active Gateways for Pervasive Communications. proposé à la conference PerCom 2004 - second IEEE international conference on pervasive computing and communications -March 14-17 2004, Orlando, Florida.

– NGUYEN Hoa-Binh, DUDA Andrzej. GateScript : A Scripting Language for Generic Active Gateways. paru à la session de poster au IWAN2003 - Fifth Annual International Working Conference on Active Networks, December 10-12, 2003, Kyoto, Japan.

Services Actifs et Passerelles Programmables

Résumé

Nous avons développé une passerelle active générique appelée *ProAN* supportant plusieurs environnements d'exécution. L'implémentation de ProAN se fait sous Linux. Trois environnements d'exécution sont disponibles dans ProAN : Linux, pour les services écrits en C, Java et GateScript. L'environnement GateScript offre un langage de script pour programmer les services actifs. Nous proposons une architecture générique pour les services actifs qui peut être instanciée pour un service traitant un protocole donné de n'importe quel niveau : réseau, transport ou application. Cet environnement propose aussi de générer automatiquement l'analyseur et le générateur de PDU du protocole en question en utilisant un langage de description de protocole comme Flavor ou JavaCC. ProAN est également adapté aux services proactifs pour les environnements pervasifs. Ces services proactifs peuvent réagir aux changements d'état de l'environnement sans l'intervention de l'utilisateur. Le service de découverte permet aux services proactifs de trouver des moniteurs de l'environnement ainsi que d'autres services dans le réseau pour donner une meilleure qualité de service aux utilisateurs.

Mots clés : Réseaux Actifs, Passerelle Générique Programmables, GateScript, Services Proactifs

---

Active Services and Programmable Gateways

Abstract

We developed a generic active gateway called *ProAN* supporting different execution environnements for active serices. Its implementation is realized under Linux. Three execution environnements are available on ProAN : Linux for active services written in C, Java and GateScript. GateScript environnement offers a script language for programmer active services. We also propose a generic architecture for active services in GateScript. This architecture can be instanciated for an active service treating a given protocol of whichever level. PDU parser and generator are also generated automatically from a PDU description file by using Flavor language or JavaCC. ProAN is also adapted for proactive services for pervasive environments. Proactive services can react to the changes of the state of environment without the user intervention. The lookup service allows proactive services to find environment monitors or other services in the networks in order to give a better quality of service to users.

Keywords : Active Networks, Generic Active Gateway, GateScript, Proactive Services

---

Discipline : Informatique, Systèmes et Communications

Laboratoire : LSR-IMAG – Équipe Drakkar
BP 72, 38402 Saint Martin d'Hères Cedex, France