Flexibilité et performance dans les routeurs actifs logiciels pour un support efficace des services déployés sur des réseaux gigabits

Jean-Patrick Gelas et Laurent Lefèvre
INRIA RESO / Laboratoire LIP (UMR CNRS, ENS Lyon, INRIA, UCB Lyon 1)
Ecole Normale Supérieure de Lyon
46, allée d'Italie 69364 LYON Cedex 07 - France
jean-patrick.gelas@ens-lyon.fr, laurent.lefevre@inria.fr

Résumé

Proposer des environnement d'executions actifs capables de s'adapter aux contraintes de performances des réseaux actuels demeure un sujet peu exploré. Cet article présente un ensemble de solutions pour la mise en oeuvre d'un noeud actif logiciel hautes performances. Basé sur une architecture multi-niveaux, cet environnement d'exécution a été implémenté sous la forme de l'environnement Tamanoir. Des mesures de performance sur différentes plate-formes expérimentales locales ou longue distance sont présentées[1].

1 Introduction

Les réseaux actifs constituent aujourd'hui un domaine de recherche fortement étudié, notamment aux USA, en Europe et au Japon, dont le but est d'exploiter le réseau de manière plus efficace et plus intelligente.

Un réseau actif contrairement à un réseau traditionnel n'est pas un simple support passif de paquets. Il peut être vu comme un ensemble de nœuds (routeurs) *actifs* qui réalisent des opérations personnalisées sur les flux de données qui le traversent, et qui autorise les utilisateurs, les opérateurs, ou les fournisseurs de services à injecter leurs propres programmes dans les nœuds du réseau, permettant ainsi de modifier, stocker (cacher) ou rediriger le flux de données à travers le réseau.

Le modèle de réseau actif ouvre les portes à de nouvelles applications. De nombreux projets expérimentent la réalisation de nouveaux protocoles (multicast, contrôle de congestion,...) fondés sur l'intelligence du réseau.

Il y a trois composants logiciels qui permettent de construire une architecture active complète pour le support de l'exécution d'une Application Active [Cal99] :

- l'Environnement d'Exécution (EE) ;
- le système d'exploitation du nœud (*node OS*) ;
- l'application utilisateur.

Un Environnement d'Exécution doit être vu comme un cadre logiciel qui offre à une Application Active la possibilité d'être chargée et exécutée. L'EE doit fournir une interface qui permet à l'AA d'interagir de manière contrôlée avec le système d'exploitation. Enfin, une AA est écrite pour pouvoir être exécutée dans un EE donné.

La courbe de progression réalisée dans le domaine des technologies réseaux suit une pente supérieure à celle de la progression realisée dans le domaine des processeurs, même si les progrès de ceux-ci suivent toujours la fameuse loi de Moore énoncée en 1975 (doublement des performances des circuits integrés tous les 18 mois). La loi de Georges Gilder indique que la bande passante des

[1] Une partie de ces travaux a été menée dans le cadre du projet RNRT VTHD++ et du projet RNTL Etoile.

réseaux augmente trois fois plus vite que la puissance des ordinateurs. Les calculs d'Eric Schmidt (CEO de *Google*) et les études de *Probe Research* montrent que la bande passante double en fait tous les ans depuis 1997[Riv02]

Les réseaux actifs mettent en œuvre des services réseaux plus complexes que de la transmission simple de paquets de données, ce qui a pour effet de consommer plus de cycles CPU ou de mémoire. Le développement des technologies de transmissions de données et de commutations ne cesse de s'améliorer. Les débits supportés par les liens ne cessent d'augmenter. On trouve actuellement au coeur des épines dorsales des routeurs multi gigabits. Dans un futur proche, la technologie « réseaux actifs »pourrait se retrouver dans différents types d'équipements réseaux (routeurs, passerelles, proxy...). Mais le concept des réseaux actifs, bien que séduisant, soulève de nombreuses problématiques en termes de performances. Une architecture performante de routeur actif doit être capable de fournir suffisamment de ressources de calcul pour supporter des débits réseaux de plus en plus importants.

Les travaux présentés dans cet article se placent dans le contexte des réseaux actifs logiciels hautes performances. Les contributions de cet article se situent à un double niveau :

- l'apport d'un ensemble de contributions originales à la conception d'une architecture logicielle de nœud actif apte à supporter les contraintes des réseaux actuels ;
- l'étude de la conception d'un nœud actif logiciel performant. Une solution au problème de la performance est de proposer des services et un environnement d'exécution efficace afin de minimiser le temps supplémentaire de transfert pour chaque paquet d'un flux de données sur le réseau. La métrique que nous emploierons est le débit moyen agrégé de flux de données actifs auquels un équipement actif applique un service.

En combinant la haute performance et le déploiement de services portables, nous explorons de nouvelles approches capables d'ouvrir des pistes pour le développement des réseaux actifs. Nous proposons ainsi une nouvelle modélisation des réseaux actifs haute performance et sa mise en œuvre sous la forme d'un environnement et d'une suite logicielle appelée *Tamanoir*. Notre objectif est ainsi de fournir un équipement actif logiciel bénéficiant de solutions de haut niveau pour traiter un flux au niveau applicatif sur le plan données, et de solutions de bas niveau destinées à traiter un flux de façon otpimisée sur le plan contrôle. Enfin nous proposons une validation expérimentale et fonctionnelle de la mise en œuvre de l'architecture active performante décrite dans ce document.

La section 2 présente un bilan exhaustif des travaux qui se sont concentrés sur le problème de la performance. Puis la section 3 présente nos propositions pour l'élaboration d'une architecture de réseau actif haute performance. La section 4 décrit la mise en œuvre de l'architecture sous la forme de l'environnement d'execution Tamanoir. Des mesures de performances sur différentes plate-formes sont décrites dans la section 5. La dernière section conclue cet article et présente diverses perspectives de recherche.

2 État de l'art sur les réseaux actifs hautes performances

Le concept de réseau actif est né en 1996 au MIT aux États-Unis, sur une proposition de David Tennennhouse et David Wetherall [WT96]. Il s'est rapidement étendu à l'Europe et constitue aujourd'hui un sujet de recherche universitaire et industriel très actif [FCF00]. En l'an 2000, plus de 50 projets de recherche travaillent sur ce thème et regroupent plus de 80 laboratoires universitaires et industriels dans le monde. L'apport supposé en dynamicité des réseaux actifs est tel que de nombreux industriels (Alcatel, France Télécom R&D, Nortel Networks[nor], Thales,...) et des organismes gouvernementaux (DARPA [Cal99, hom]) ont jugé utile d'investir dans ce domaine de recherche. Bien que de nombreux progrès aient été réalisés ces quelques dernières années pour rendre la configuration des équipements des réseaux de données plus souple, ceux-ci restent relativement figés et spécifiques à un type d'application. L'extensibilité de ces équipements reste limitée. Ils fournissent uniquement les services pour lesquels ils ont été conçus.

Or les opérateurs et fournisseurs de services ont un besoin crucial de dynamicité pour répondre dans un délai très court aux besoins des usagers. De nouvelles applications comme la téléphonie

sur IP, la diffusion de radio ou de canaux TV, les services de cotations boursières ou encore d'achats aux enchères en ligne sur l'Internet existent déjà. Malheureusement, les services de base fournis par les protocoles réseau sont actuellement mal adaptés à ces applications d'un genre nouveau et avec des exigences particulières. Pour y remédier on voit apparaître une multitude de protocoles de contrôle et de réservation de ressources (MPLS (*MultiProtocol Label Switching*), DiffServ, RSVP (*ReSerVation Protocol*),...) mais les constructeurs d'équipements ne peuvent pas intégrer tous les nouveaux protocoles dans leurs matériels en un temps adapté aux besoins des usagers. Des solutions dynamiques apparurent véritablement à partir de 1995 avec une focalisation sur les réseaux programmables. Un réseau programmable est un réseau de transmission de données ouvert et extensible disposant d'une infrastructure dédiée à l'intégration et à la mise en œuvre rapide de nouveaux services sur l'ensemble de ses composants [FCF00].

Un réseau traditionnel (ou passif) est un réseau de transport de données qui possède un nombre restreint et fixé de services implantés dans les équipements et qui n'offre aucun moyen d'en ajouter. Par conséquent il est impossible de modifier dynamiquement le comportement global du réseau. Au contraire, un réseau programmable est un réseau de transport de données étendu par un environnement de programmation à l'échelle du réseau comportant un modèle de programmation des services, des mécanismes de déploiements et un Environnement d'Exécution (EE). Une initiative est apparue en 1997 au sein de l'IEEE sous la forme du projet P1520 (*Programmable Interface for Networks*) [BHL+99]. En 1996, sur une proposition de David Tennennhouse et David Wetherall apparait le concept de réseaux actifs[WT96]. Un réseau actif repose sur des équipements dont les composants dans les différents plans (signalisation, supervision, données) sont programmables dynamiquement par des entités tierces (opérateurs, fournisseurs de services, applications, usagers)[FCF00]. Cette approche est plutôt issue des travaux sur l'insertion de services applicatifs dans le réseau Internet, par opposition aux réseaux programmables qui sont issus de travaux sur la signalisation pure dans les réseaux de télécommunications de type ATM. Elle étend le concept de programmation en ouvrant les équipements de l'ensemble du réseau et en partant du principe que tout usager peut concevoir et déployer à l'aide d'interface de programmation (API) de nouveaux services. Ces nouveaux services peuvent alors être utilisés de manière dynamique dans le réseau.

On trouve, dans la littérature, peu de projets consacrés à la problématique de la haute performance dans les architectures de réseaux actifs logiciels. Les sections suivantes présentent les travaux les plus représentatifs dans ce domaine :

2.1 PAN : Practical Active Network

Le projet PAN pour *Practical Active Networking* [LJK99a] est né dans les laboratoires du MIT. C'est un projet inspiré de ANTS [Wet99] mais dédié à la conception d'un nœud actif haute performance. Le projet PAN étudie le chemin critique des données et essaie de supprimer les sources potentielles de coûts supplémentaires qui sont : les recopies mémoire, l'interprétation de code, le chargement de code et le passage de données de l'espace noyau à l'espace utilisateur. Écrit en C, PAN obtient de très bonnes performances brutes. Il en existe deux mises en œuvres : l'une s'exécute dans l'espace utilisateur, l'autre dans l'espace noyau, sous forme de modules. Les mesures ont été faite sur UDP pour des paquets dont la taille varie de 128 à 1500 octets. Les meilleures performances sont obtenues pour une exécution exclusivement dans l'espace noyau. Les résultats expérimentaux montrent que pour des petits paquet (128 octets) la recopie coûte peu. Pour des paquets de 1500 octets l'EE natif exécuté dans l'espace utilisateur coûte 5 fois plus que la version noyau. La version exécutée dans le noyau est capable de saturer un lien *Fast Ethernet* avec des paquets de 1500 octets et un sur coût de seulement 13 % pour traiter chaque paquet. Ces résultats sont obtenus grâce à des mécanismes limitant le nombre de copies en mémoire, un traitement des paquets lorsque cela est strictement nécessaire et enfin, par un usage intensif de code natif. L'utilisation de code natif pose un problème pour la portabilité d'un service qui doit pouvoir s'exécuter sur des équipements réseau aux architectures hétérogènes et pose également des problèmes de sûreté et de sécurité.

2.2 ANN : Active Network Node

Le projet ANN[DPC+99] est focalisé sur la conception d'un support de réseau Gigabit ATM. La conception matérielle de ce projet est réalisée à l'université de Washington et le serveur de code ainsi que l'environnement DAN sont développés conjointement à l'ETHZ et à l'Université de Washington.

La mise en œuvre de ANN repose sur une approche matérielle connectée à un réseau ATM, un système d'exploitation performant adapté à l'architecture matérielle choisie, et un EE spécifique nécessaire à l'architecture et à la gestion du téléchargement de code actif sur un nœud. L'approche d'ANN consiste à traiter des flux de paquets actifs et non pas des paquets isolés. Chaque paquet contient une référence à un module actif qui est stocké sur un serveur de codes de confiance *(trusted code server)* appelée DAN pour *Distributed code caching for Active Networks* [DP98]. Les modules sont liés dynamiquement et exécutés en code natif sur le routeur. Ce projet supporte ANTS [Wet99] qui fournit l'environnement souple pour le prototypage de modules. ANN démontre qu'il est important de lier étroitement la capacité de traitement et le réseau ainsi que de distribuer le traitement sur les CPU disponibles. ANN utilise des modules appelés ANPE pour *Active Network Processing Elements* constitué d'un processeur (Pentium), de mémoire et d'un APIC (interface réseau). Un ensemble d'ANPE constitue un Active Network Node (ANN). Un ANN est connecté à un commutateur ATM.

2.3 CLARA : Nœud actif basé sur une grappe

Le projet CLARA [OWM00] a été mené par une équipe de la société NEC, partie du constat que la première cause de dégradation d'un service est la perte de bande passante. CLARA est un prototype d'architecture de nœud de routage basé sur un modèle de grappe issue du projet *Journey* dont l'objectif est de déployer des routeurs auxquels on adjoint des capacités de traitements. Les unités de données sont appelées *media unit*. Elles peuvent contenir par exemple un GoP (*Group of Picture*) d'un flux MPEG. Chaque média unit est considérée comme indépendante. Le modèle employé permet de traiter les média unit en fonction des conditions et disponibilités locales. Les décisions sont prises indépendamment des autres routeurs. Il n'y a pas de message de contrôle entre les routeurs pour traiter un flux particulier. Ce modèle ne garantit pas que chaque *media unit* arrive traité à destination. Il reste dans l'esprit du routage *best effort* du réseau IP. Pour déterminer l'état traité ou non traité d'un *media unit* CLARA utilise l'option *IP Router Alert*. Traité : l'option IP RA est désactivée et le paquet est routé directement ; Non traité : l'option IP RA est activée et le paquet est pris en charge par CLARA pour savoir s'il va pouvoir être traité. L'architecture d'un nœud actif CLARA est constituée d'un PC destiné au routage relié par un réseau local très haut débit à d'autres PC dédiés exclusivement aux traitements. Les *media unit* sont distribuées sur les PC de traitements suivant un simple algorithme de tourniquet *(round-robin)*.

2.4 Services noyau

Plusieurs projets ont étudié la possibilité de déployer des services actifs dans l'espace noyau. La première architecture de nœud actif capable d'accueillir des services actifs directement dans le noyau du système d'exploitation (Solaris) est décrite dans [SKE97]. C'est une approche dédiée au nœud actif où la référence à la fonction à appliquer est passée en paramètre dans les options IP. Cette référence doit être unique pour une fonction donnée. Chaque fonction peut être paramétré sur le nœud ou les paramètres peuvent être fournis dans l'en-tête de chaque paquet. Les nœud permettent d'accéder à certain de leurs états et possèdent une table de fonctions avec des pointeurs sur le code des fonctions disponibles. Le nœud appelle alors la fonction de traitement à appliquer lorsqu'un paquet arrive. L'objectif de ce projet était d'obtenir de bonnes performances pour l'exécution des services. Malgré cela, il manque à cette architecture un mécanisme de déploiement dynamique pour rendre le nœud actif facilement extensible. Les projets PromethOS [KRGP02] et ProActive [ND02] utilisent quand à eux Netfilter qui permet d'étendre l'architec-

ture Linux standard en ajoutant en cours d'exécution des extensions pour le support de nouveaux composants. Même si l'interface est contrainte à celle de Netfilter, les performances de ce projet sont comparables à l'environnement réseau standard de Linux.

Ces différents travaux ont permis de défricher le domaine et ont fait apparaître très nettement les verrous qui demeurent : la sécurité, la performance et le support de l'hétérogénéité. Bien qu'il existe de nombreuses propositions, les prototypes opérationnels proposés jusque là sont peu nombreux, peu transposables lorsqu'il nécessite du matériel dédié, ou peu fonctionnels. D'un point de vue recherche et développement, ces réseaux proposent un lieu formidable d'investigation pour la mise au point grandeur nature de nouveaux protocoles et services avant qu'ils ne soient déployés dans des équipements standard.

3 Propositions pour une architecture de réseau actif haute performance

3.1 Une architecture de nœud actif à quatre niveaux

L'un des objectifs de ce travail est de concevoir un nœud actif, capable de supporter un lien dont le débit total des flux entrant et sortant est d'au moins un gigabit par seconde. Afin de parvenir à ce résultat, nous nous sommes efforcés de minimiser tous les coûts associés à la traversée de l'Environnement d'Exécution par les paquets de données. On rappelle que l'EE est la couche logicielle qui doit, déterminer le service à appliquer à un paquet actif, diriger sa charge utile du paquet vers le service requis, éxecuter et appliquer le service, puis une fois traité, ré-emettre le paquet sur le réseau.

Afin de fournir un EE efficace, nous proposons une architecture de nœuds actifs découpée en quatre niveaux : NIC programmable, espace noyau, espace utilisateur, ressources distribuées (figure 1). Cette proposition est issue du constat suivant. Afin de réduire le temps passé par un paquet dans un équipement actif, il nous parait logique de réduire au maximum le nombre d'obstacles que ce paquet doit traverser. Dans un système, ces obstacles sont matérialisés par les recopies de données d'une zone mémoire à l'autre, notamment lors du passage de l'espace noyau à l'espace utilisateur, par l'allocation dynamique de mémoire, par la génération d'interruptions et l'exécution d'appels systèmes.

La première couche est située dans l'interface réseau (NIC) qui constitue la première zone du noeud susceptible d'embarquer suffisamment de logique pour traiter des paquets. Il existe aujourd'hui des cartes d'interface réseaux programmables équipées d'un (ou plusieurs) processeur(s), de mémoire et moteur DMA (Myrinet, Quadrics...). Cette couche semble donc très bien adaptée dans un contexte haute performance puisque c'est celle qui est située au plus près du « fil ». Cela signifie qu'un paquet n'aura que très peu d'obstacles à traverser pour être traité. Néanmoins cette couche implique quelques contraintes. La puissance de traitement proposée par les cartes réseaux programmables est aujourd'hui encore inférieure à celle du processeur principal d'un système. L'espace mémoire disponible est également relativement contraint et donc insuffisant pour un service qui nécessite de cacher des quantités importantes de données.

La seconde couche correspond à la zone mémoire où s'exécute le noyau d'un système d'exploitation. On y trouve les pilotes des cartes d'interfaces réseaux. L'avantage de traiter un paquet dans cet espace est que l'on bénéficie de la puissance du processeur hôte et de sa mémoire tout en déployant des services près du lien de communication. Mais l'exécution d'un service dans cette couche n'est pas sans contraintes : le développement de services noyau implique que l'on emploie un langage proche du processeur hôte (assembleur ou C compilé pour l'architecture) ce qui limitera la portabilité des services compilés. On peut imaginer un interpréteur embarqué dans le noyau mais bien sûr au détriment des performances. De plus un service mal programmé mettra inévitablement tout le nœud actif en péril. Enfin le système d'exploitation doit supporter le chargement dynamique de nouveaux codes. Notre première proposition consiste donc à limiter la remontée du

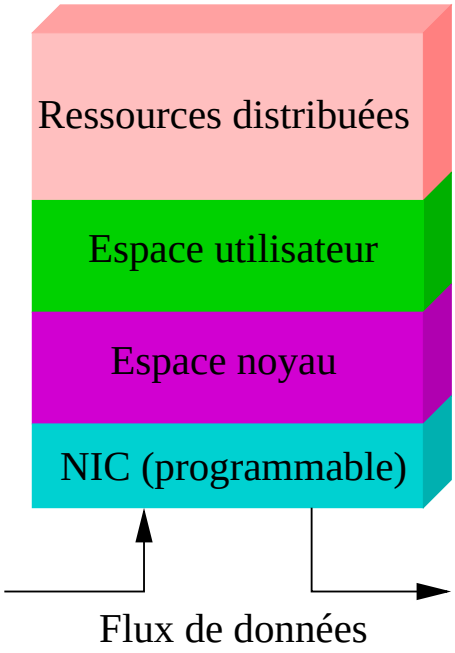

FIG. 1 – Architecture d'un nœud actif en 4 couches

paquet du réseau vers l'espace noyau de l'équipement actif. Les figures 2 et 3 illustrent les parties du système traversées par le paquet actif. Ici le paquet est démultiplexé par un EE allégé appelé μEE et on constate que cette approche permet de nous affranchir des recopies coûteuses entre l'espace noyau et l'espace utilisateur. On retrouve un environnement d'exécution (simplifié), un mécanisme de démultiplexage et des services chargeables dynamiquement.

La troisième couche correspond à la zone mémoire de l'espace utilisateur. Cette zone bénéficie non seulement du processeur et de la mémoire du système hôte mais également de tous les équipements dont le système est pourvu (mémoire de masse (disques durs), cartes dédiées (ex : compression, cryptage)). C'est la zone mémoire où s'exécutent les applications des utilisateurs, dans le langage de leur choix (Java, Perl,...), c'est donc une zone protégée mais qui en contrepartie est située "loin" du lien. Un environnement d'exécution en Java sera déployé dans cet espace utilisateur.

La quatrième couche appelée ressources distribuées repose sur l'intégration de ressources dans l'architecture logicielle de noeud actif. Les flux de données auxquels doivent être appliqués des services lourds (ex : compression, cryptographie, transcodage ou conversion à la volée) mettent en exergue le besoin et l'intérêt d'une architecture parallèle pour les noeuds actifs. Nous proposons dans cette section plusieurs modes de traitements distribués.

La figure 4 présente trois types d'architectures susceptibles de pouvoir accueillir un nœud actif distribué. Dans la figure 4-a, l'EE est exécuté soit sur une machine de type SMP (un système à mémoire partagée) soit sur une grappe de machine avec une bibliothèque de mémoire distribuée partagée (MDVP). Les paquets atteignant ce nœud actif sont placés dans des files d'attente situées dans la mémoire partagée avec une file pour chaque service disponible, mais une file unique pour les services identiques. Les services sont considérés comme les consommateurs de ces files d'attentes. Dans la figure 4-b, l'EE reçoit les paquets des flux actifs. Il analyse l'en-tête active de chaque paquet et les dirige vers le nœud qui possède le service requis. Une bibliothèque de passage de message comme MPI (*Message Passing Interface*) peut être utilisée. Ensuite, le nœud traite le paquet avec le service adapté avant sa retransmission. Chaque service est un processus en attente

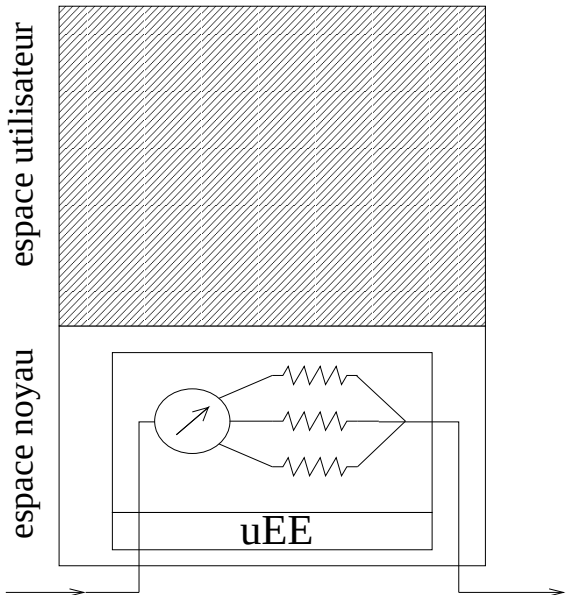

FIG. 2 – Le traitement est réalisé dans l'espace noyau.

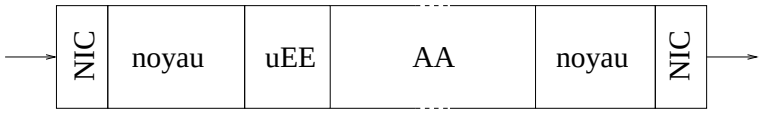

FIG. 3 – Chronogramme d'un paquet actif traité dans l'espace noyau.

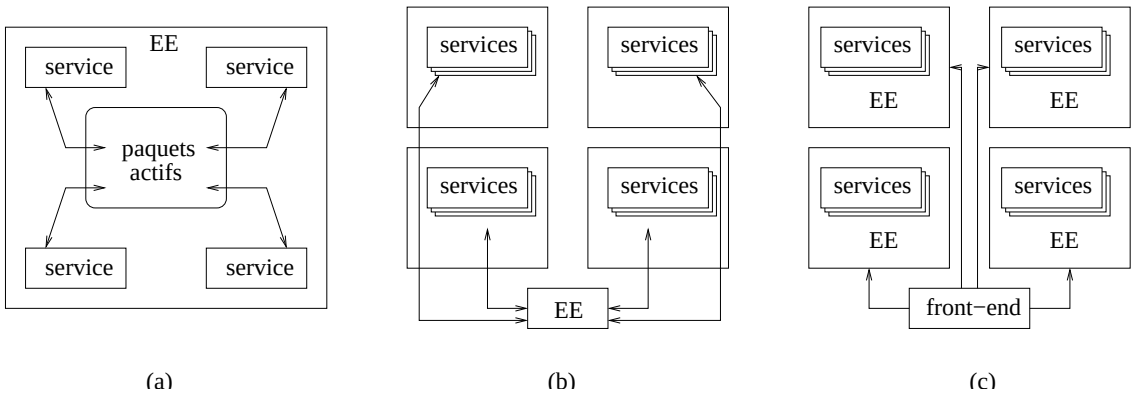

FIG. 4 – Différentes approches de conception d'un nœud actif distribué : (a) mémoire partagée, (b) passage de messages, (c) EE répliqués.

de messages provenant de l'EE. Dans la figure 4-c, l'EE et ses services sont répliqués sur chacun des nœuds de la grappe (appelés *back-ends*). Un nœud frontal (*front-end*) est dédié à la distribution des flux sur les back-ends en mettant en œuvre un algorithme de distribution de flux. On peut utiliser un simple algorithme du tourniquet (*Round-Robin*(RR)) mais cela risque de conduire très rapidement, dans un contexte réel, à un déséquilibre de la charge de calcul entre les différents back-ends. D'autres algorithmes d'équilibrage de charge plus sophistiqués sont disponibles. L'algorithme *Weighted-RR* consiste à ajouter un coefficient correspondant à la puissance de calcul de chaque *back-ends*. Ainsi le back-end ayant reçu le coefficient le plus fort se verra attribuer un nombre de flux plus important. Cet algorithme permet ainsi la construction de grappes hétérogènes. Un autre algorithme appelé *Least Connection RR* consiste à diriger un nouveau flux vers le back-end ayant le moins de connections en cours. Il permet d'équilibrer le nombre de connections sur chaque back-end sans pré-requis sur la durée des connections. On peut ensuite imaginer un algorithme qui combine ces deux derniers afin de tendre vers l'optimal. Nous étudions actuellement une stratégie de distribution différente, de celles proposées ci-dessus, basée sur le concept de boucle de rétroaction.

Notre proposition d'architecture logicielle pour un noeud actif distribué est fondée sur la solution où les EE sont répliqués (4-c). L'équipement frontal destiné à distribuer les paquets ou les flux sur les équipements actifs situés en arrière (*Back End* (BE)), est représenté sur la figure 5. L'équilibrage de charge est donc réalisé par le frontal de préférence dans l'espace noyau. Le frontal pourra également être utilisé pour capturer les paquets actifs dans un mode de fonctionnement bout en bout. Les BE peuvent toujours exploiter l'une ou l'autre des deux approches présentées ci-dessus qui consistent à exécuter un EE dans l'espace utilisateur ou dans l'espace noyau. Dans l'architecture distribuée choisie (fig. 5) le paquet doit traverser un réseau local (LAN) et donc une carte d'interface supplémentaire.

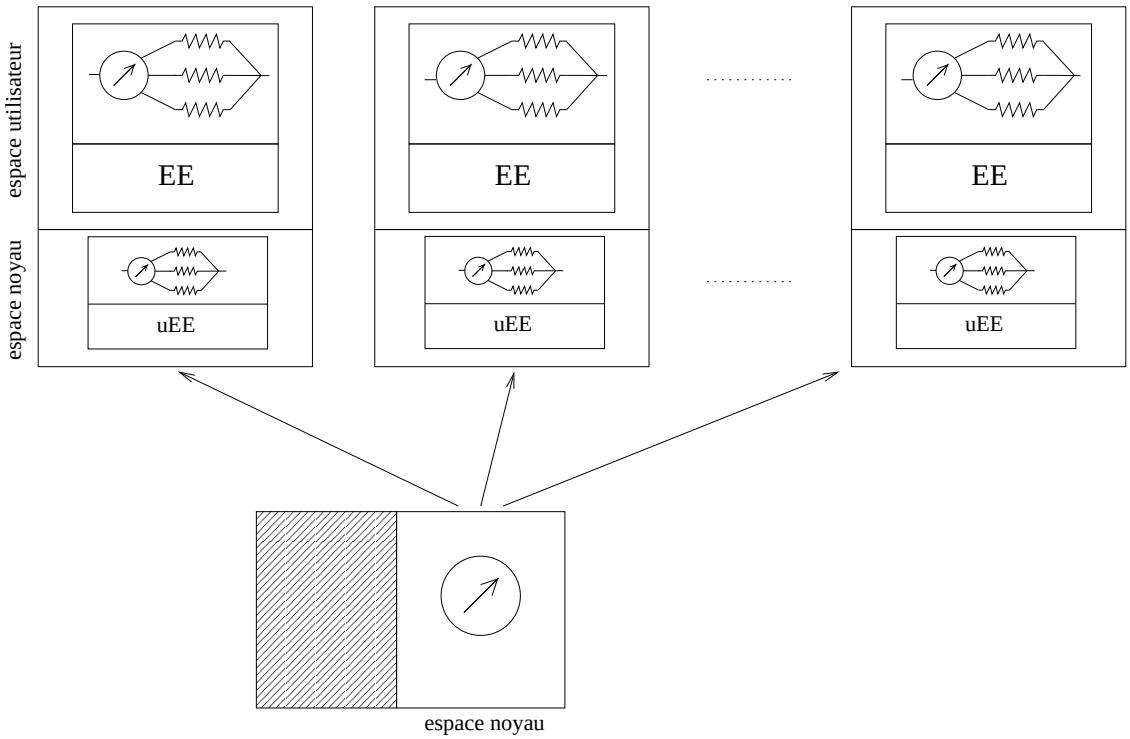

FIG. 5 – Architecture distribuée de noeud actif logiciel avec machine frontale et *back-ends*

Finalement, la figure 6 résume les différents cheminements possibles d'un paquet actif dans notre proposition d'architecture. Dans le cas d'un traitement distribué, les flux de données tra-

versent un frontal (NIC (frontal)) qui distribue les flux ou les paquets aux BE embarquant un EE dans l'espace noyau ou/et dans l'espace utilisateur. On pourrait imaginer exécuter les services léger directement sur le frontal, mais il serait probablement insuffisant pour supporter de nombreux flux. Dans le cas d'un nœud actif qui n'est pas distribué son fonctionnement est identique à celui d'un BE qui ne serait pas précédé d'un frontal et qui serait directement connecté au réseau. Le paquet actif passe à travers une carte (NIC) puis s'arrête dans l'espace noyau pour être traité par le uEE (μEE) ou le cas échéant remonte jusque dans l'espace utilisateur pour être traité par un EE de plus haut niveau On remarque que tous les éléments traversés par le paquet ajoutent une latence. La somme de ces latences plus celle induite par le service (AA) donne la latence totale pour la traversée d'un paquet. La latence induite par le service (AA) n'est « maîtrisée »que par son développeur. Si il conçoit un algorithme coûteux en cycle CPU, il augmente d'autant plus le temps de passage dans l'équipement actif.

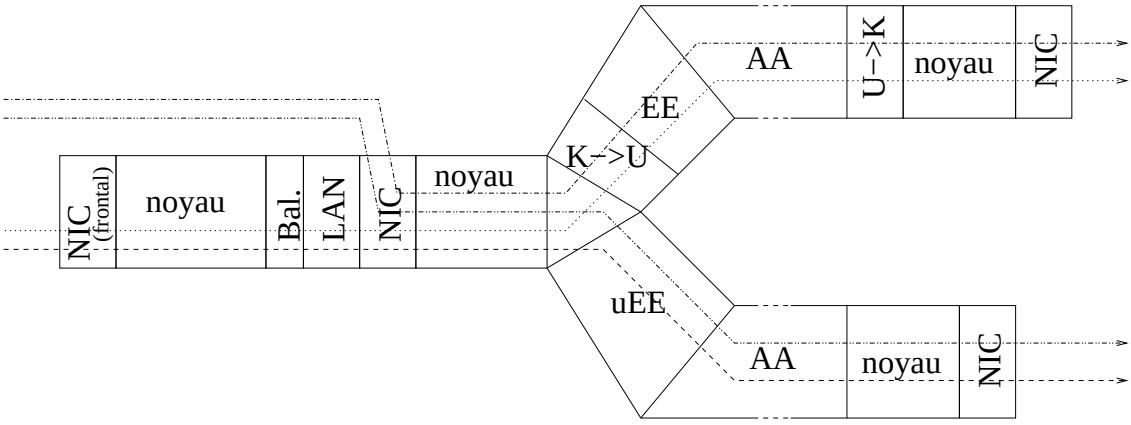

FIG. 6 – Chrono-gramme global pour la traversée de l'équipement actif

Notre proposition de découpage d'un nœud actif permet de nous rendre compte que chaque couche possède ses propres caractéristiques, avantages et inconvénients. Plus les services dynamiques sont proches du lien de communication, plus les contraintes sont fortes et la mise en œuvre difficile. C'est pourtant à ce prix que le gain en performance semble le plus évident. On prévoira donc un mécanisme de communications inter couches afin qu'un même service puisse être distribué sur les différents niveaux. Les couches hautes fourniront la souplesse nécessaire (par exemple au déploiement) et la force de calcul brute (routeur-cluster) et les couches basses autoriseront des services performants. Bref, il est nécessaire de classifier les services qui en fonction de leur caractéristiques (consommation CPU et/ou mémoire, complexité, fonctionnalités, langage d'écriture) seront exécutés dans la couche la mieux adaptée.

3.2 Classifier les services

Les travaux de thèse de V. Galtier [GMC+00] ont montré qu'il est extrêmement complexe de réaliser des prévisions sur la quantité de ressources nécessaires au bon fonctionnement d'un service. L'usage des ressources est représenté par la consommation de cycles CPU et la quantité de mémoire nécessaire au bon fonctionnement du service. Néanmoins nous pouvons tenter de classifier les services en quatre grandes familles. Un paquet actif peut se voir appliquer des services :

- **hyper légers** : peu consommateurs de CPU, peu consommateurs de mémoire. Ce sont des services sans états, qui applique un traitement extrêmement léger sur les paquets actifs tel qu'un service de marquage ou de filtrage. Ces services peuvent être exécutés directement sur une carte d'interface programmable, au plus près du lien.
- **légers** : toujours peu consommateurs de cycles CPU, mais demandant un minimum de mémoire pour pouvoir y stocker des états associés aux flux. Ces services peuvent être exécutés dans l'espace noyau du système qui fournit un accès total à la mémoire du système. Ces

services légers bénéficient du fait que les paquets de données ne traversent pas la barrière entre espace noyau et espace utilisateur (réduction du nombre de copies) pour conserver d'excellentes performances.

- **moyens** : demandent un environnement riche pour pouvoir effectuer des traitements complexes mais pas forcément très lourd en calcul. Ces services s'exécutent dans l'espace utilisateur, donc dans un espace protégé, ne risquant pas de mettre le nœud actif en péril. Le service peut accéder à toutes les ressources matérielles mises à sa disposition sur le nœud (mémoire, disques, cartes spécialisées, réseau...).
- **lourds** : très consommateurs de cycle CPU et/ou très consommateurs de mémoire. Ces services ont besoin d'être distribués et parallélisés. Cette parallélisation peut intervenir à deux niveaux de granularité différents : au niveau paquet ou au niveau flux. Dans le premier cas les paquets sont distribués sur les unités de traitement qui leur appliquent le service. Dans le second cas, un flux complet est associé à une unité de traitement. Les flux sont alors traités en parallèle. Il est ensuite important d'employer un algorithme bien adapté de distribution des paquets ou des flux, en fonction de la granularité choisie.

On peut également faire une distinction simplificatrice. La plupart des services agissant sur le *plan contrôle* semble peu consommateurs en cycle CPU et espace mémoire, ils pourront donc être mis en œuvre dans un service léger ou hyper léger. En contre partie les services agissant sur le *plan données* nécessitent plus de ressources et pourront donc être mis en œuvre dans un service moyen ou lourd.

4 L'environnement logiciel Tamanoir

Nos travaux consacrés au domaine des *réseaux actifs* orientés haute performance ont été développés sous la forme d'un environnement logiciel actif appelé *Tamanoir*[2], qui se propose de répondre aux problèmes de performance, d'hétérogénéité et de déploiement dynamique de services[GL00, GL02, GEHL03]. L'environnement *Tamanoir* fournit aux utilisateurs la possibilité de déployer et de maintenir dynamiquement des noeuds actifs, appelés TAN pour *Tamanoir Active Node*, distribués sur un réseau à grande échelle.

4.1 Déploiement des nœuds Tamanoir dans le réseau

Nous pensons que le gain en performance est étroitement lié à la localisation des composantes de notre architecture. On parlera :

- d'urbanisation ou de déploiement horizontal pour traiter du positionnement d'un équipement actif ou d'un service dans le réseau ;
- et de classification de services ou déploiement vertical pour trouver la couche la mieux adaptée pour l'exécution d'un service au cœur d'un nœud actif.

4.2 Urbanisation d'équipements actifs

Nous faisons la distinction entre les routeurs de bords et les routeurs de cœur du réseau. Dans notre architecture, seuls les routeurs de bordure sont dotés de capacités de traitements (figure 7).

Les routeurs situés au cœur du réseau doivent garantir de très hauts débits (de l'ordre de plusieurs Gbits/s). Ils doivent rester le plus simple et le plus passif possible. En y plaçant de l'*intelligence* et donc du traitement, on risque de réduire les performances de l'épine dorsale. Les nœuds actifs trouvent leur place dans un contexte réseau de performance plus modeste dans le réseau d'accès, où l'on rencontre des points de connections entre réseaux hétérogènes. On peut également imaginer une structuration hiérarchique des équipements actifs et ne pas limiter leur déploiement uniquement en entrée d'un domaine. Une disposition arborescente s'avérera probablement très utile pour des services de cache par exemple. Enfin on peut imaginer employer un

[2] Le tamanoir aussi connu sous le nom de grand fourmilier se nourrit exclusivement de fourmis (30,000 par jour). Ce nom a été choisi en référence au projet ANTS [Wet99] (fourmis).

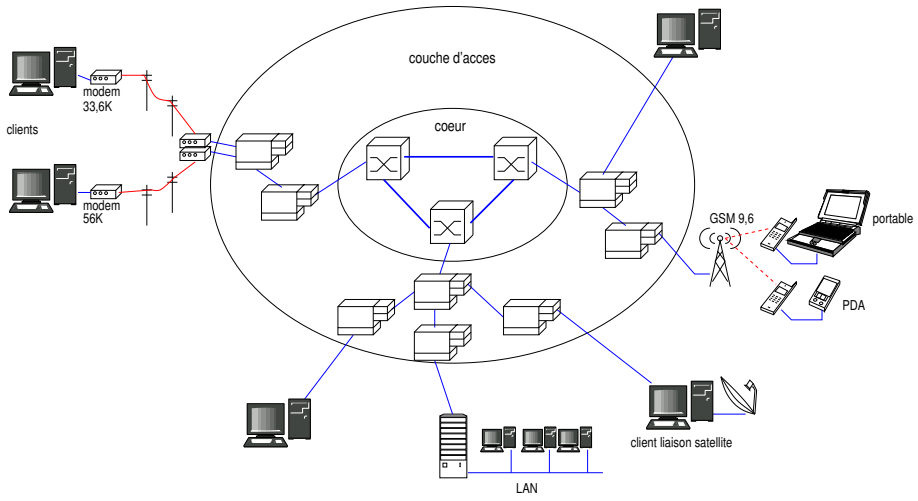

FIG. 7 – Équipements actifs haute performance déployés dans la couche d'accès.

nœud actif haute performance à l'intersection de deux réseaux d'opérateurs afin de déployer des services dynamiques adaptés aux accords de *peering* des opérateurs.

4.3 Urbanisation de services

Afin d'expliciter les différents déploiements possibles d'un service au sein d'une infrastructure active nous nous appuierons sur la figure 8. On appelle service composé un service constitué de plusieurs services, c'est donc un service de plus haut niveau. La figure 8-I présente un service composé, déployé uniquement sur le premier nœud actif traversé. Celui-ci est donc en charge d'appliquer les trois services à chacun des paquets. Afin d'optimiser l'application des services sur les paquets on peut utiliser une version de nœud actif distribué. Le flux passe ensuite à travers les autres équipements de manière transparente jusqu'à atteindre le récepteur. Le déploiement du service est transparent pour l'application. On parle de déploiement opportuniste de service car s'il n'y aucun équipement actif sur le chemin, le flux ne sera pas traité. La figure 8-II présente une distribution des services sur plusieurs nœuds actifs. Ce type de distribution permet de diviser la charge de calcul (ou de stockage) des nœuds. Cette topologie des services permet un traitement en mode *pipeliné* des paquets. L'inconvénient est qu'il est nécessaire de maîtriser l'infrastructure pour connaître le nombre d'équipements traversés et si ils sont opérationnels. La figure 8-III présente un service recyclé. Le même service est déployé sur tous les équipements actifs situés sur le passage du flux. Les paquets se voient donc appliquer le même service à chaque fois qu'ils passent dans un équipement actif. Le déploiement est également opportuniste.

On peut optimiser le placement d'un service sur le meilleur nœud actif, en entrée ou en sortie d'un domaine par exemple (mode proxy). On peut aussi concevoir un nœud actif embarquant des cartes matérielles spécialisées (matériels dédiés) destinées à appliquer des services codés dans le matériel. On force alors le chemin des données vers ce nœud actif particulier pour l'application efficace d'un service.

Pour le déploiement des nœuds actifs Tamanoir dans le réseau nous proposons deux scénarios.

4.3.1 Connections par morceau : mode proxy

Un flux de données émis par une machine A vers une machine B située à une autre extrémité du réseau peut traverser un routeur non actif sans problème ; bien sur le flux ne se verra alors appliquer aucune optimisation ou adaptation sur ce type de nœud (Fig. 9). Le premier nœud actif traversé par le flux sera situé en entrée de la couche d'accès du réseau. Ce premier nœud actif

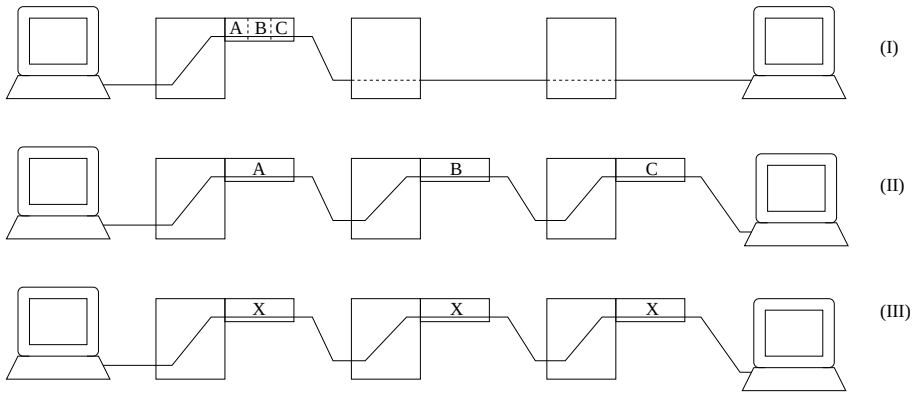

FIG. 8 – Urbanisation de services dans un réseau de nœuds actifs

a donc un statut particulier puisqu'il interconnecte un réseau classique avec l'Internet. Ce noeud est appelé *proxyTAN* : une passerelle entre les clients et l'infrastructure active. Étant donné le caractère expérimental (simulation à grande échelle) de notre plate-forme, dans un premier temps, seuls les TAN ont connaissance de la localisation des autres TAN. Pour cela chaque TAN possède une table de routage qui lui permet d'atteindre les autres TAN déployés. On parle d'un *overlay* de nœuds actifs. La connexion est « morcelée »en multiples connections point à point entre les différents TAN. Cette technique nous permet de faire remonter un paquet actif du réseau vers la couche application de notre nœud actif, car du point de vue de la couche IP le paquet semble être arrivé à destination. Cette solution par morceau est bien adaptée à certains services comme*ActiveWeb*[LG03] qui permet de modifier ou réagir à un flux `http`, non actif.

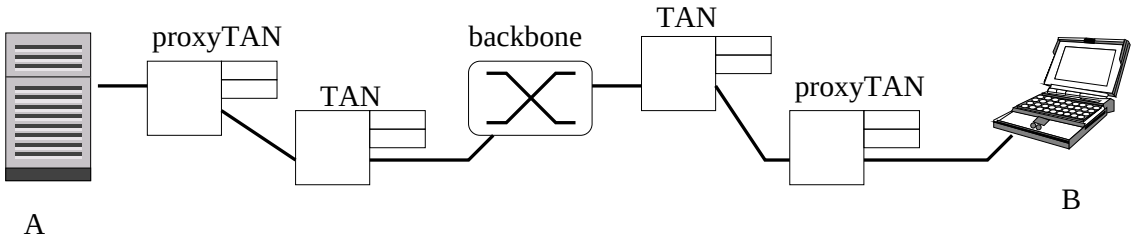

FIG. 9 – Topologie d'une architecture active par morceau. Les proxy Tan permettent d'accéder à l'infrastructure active déployée. Les routeurs classiques du backbone n'interfèrent pas sur le flux.

4.3.2 Connections de bout en bout

Fournir une connexion de bout en bout signifie que les équipements actifs traversés par un flux peuvent intercepter un flux de données, reconnaître que le flux est actif, appliquer un traitement sur le flux et le réinjecter dans le réseau en direction du poste destinataire. Afin de distinguer les paquets à traiter des paquets à transmettre directement, nous avons besoin d'un mécanisme de différenciation mis en œuvre sur les équipements réseaux (routeurs) existants. Une solution déjà mise en œuvre pour les protocoles de signalisation qui nécessitent une telle caractéristique est l'option *IP Router Alert* (RFC 2113[Kat97]). Cette option indique au routeur qu'il doit examiner le paquet pour déterminer si un traitement complémentaire est nécessaire. Pour IPv6, Router Alert (RFC 2711)[PJ99] est une option "point à point" (*Hop-by-Hop*) proposant la même sémantique. Cependant contrairement à IPv4 où seule la valeur 0 est fixée (les autres sont réservés), IPv6 à trois valeurs réservées dont l'une pour les *active network messages*. Les autres ont été assignés à RSVP et au message multicast de ICMPv6.

Un filtre est placé en entrée de l'équipement actif pour analyser les options de l'en-tête IP. Lorsqu'un paquet actif est détecté l'adresse du champ IP destination est remplacée par l'adresse locale (celle du TAN). Il recalcule ensuite le *checksum*. Puis, le paquet passe dans le module de routage et puisqu'il est destiné à l'adresse locale il est dirigé vers l'EE local en attente de paquet actif sur une socket. Le paquet est ensuite traité par le service puis transmis vers le destinataire. On a vu que l'IP destination est écrasée au profit de l'IP locale afin de faire remonter naturellement le paquet du noyau vers l'EE. Cette IP destination doit donc avoir été sauvegardée dans l'en-tête de paquet actif ANEP. L'adresse IP source du paquet transmis est maintenant celle du TAN local. D'un point de vue IP, le récepteur suivant, qu'il soit l'application réceptrice ou un autre TAN, croit que le paquet a été originellement émis depuis ce TAN. Pour certaines applications il est nécessaire de modifier à la volée lors de la sortie du paquet l'adresse IP source du paquet (et recalculer le *checksum*). Pour le protocole UDP toutes ces manipulations posent peu de difficultés.

Pour le protocole TCP, donc connecté de bout en bout, entre l'application émettrice et réceptrice, la mise en œuvre d'un tel dispositif de capture et modification de paquets sur le chemin des données devient plus difficile. Les services déployés dans ce cadre doivent avoir un impact limité sur les paquets (destruction, duplication interdites).

D'un point de vue expérimental sur un réseau longue distance, nos expériences ont montré que certains équipements réseaux opérationnels écrasent les options IP (projet RNRT VTHD++). Donc un paquet étant marqué actif par l'application grâce à l'option IP Router Alert peut se retrouver impossible à détecter efficacement par un équipement actif effectuant une discrimination des paquets à l'aide de cette seule information.

4.4 Déploiement de services actifs dans Tamanoir

Deux stratégies de déploiement de services sont disponibles. La première consiste à utiliser un serveur de services Tamanoir que l'on appelle *service repository*. La seconde repose sur le déploiement de services de proche en proche.

4.4.1 Déploiement de services par Service Repository

Le *Service Repository* est un serveur dédié à la distribution de services Tamanoir. Des applications de même classe (vidéo à la demande, téléphonie, broadcast radio,...) utiliseront probablement des services standards (transport de la voix, filtrage, multicast,...). Ces services pourront être proposés par les opérateurs, les ASP (*Application Service Provider*), les ISP (*Internet Service Provider*) ou les fournisseurs de services spécialisés. Ainsi, les routeurs actifs seront plus facilement maitrisables et pourront appliquer des services génériques sur différents flux. Dans notre mise en œuvre actuelle, nous utilisons une adresse *URL* dans le nom du service, les applications déploient ainsi des services connus que les nœuds actifs iront directement télécharger sur le serveur de services (*service repository*). En guise de serveur de services nous utilisons actuellement un serveur Web *Apache*[apa] configuré pour délivrer des services actifs Java (`.class` ou `.jar`). Afin de sécuriser le transport du service entre le service repository et le TAN requérant le service on utilise le protocole sécurisé `https`. Le téléchargement d'un service à partir du service repository est illustré sur la figure 10.

4.4.2 Déploiement de services de proche en proche

La seconde solution est d'autoriser un déploiement, à la volée des services au fur et à mesure de la traversée des routeurs actifs. Dans chaque paquet ANEP transporté, un champ est dynamiquement mis à jour avec le nom du dernier TAN traversé. C'est le champ `lastId`. Ainsi, si un TAN ne possède pas le service approprié, une demande de chargement est émise vers le dernier routeur actif TAN traversé qui lui renvoie le code du service (`.class` ou `.jar`). Les services sont transportés dans un canal de contrôle fiable dédié. Après cette opération de déploiement, le service est chargé dans la mémoire du TAN et est alors prêt à traiter les paquets.

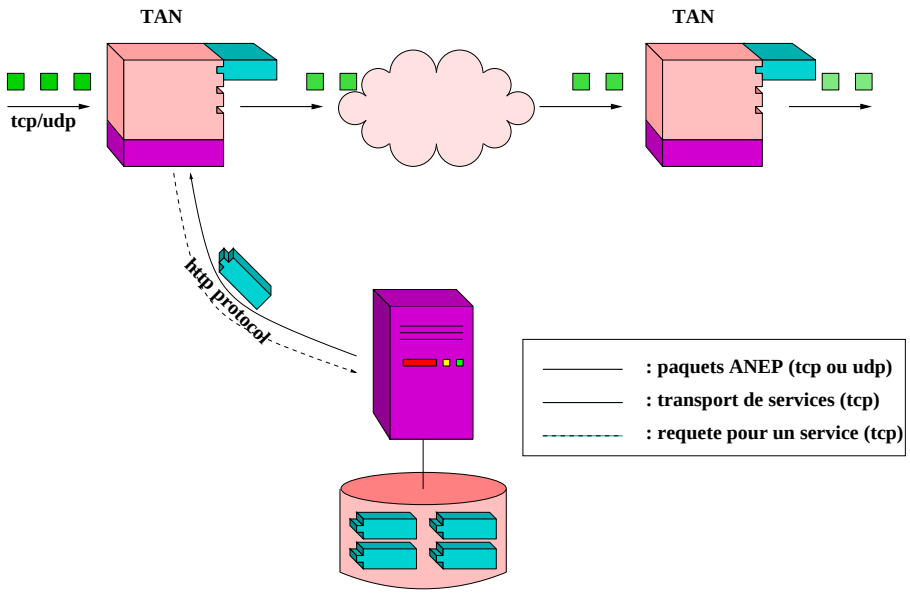

FIG. 10 – Déploiement de services à partir d'un service repository.

4.5 Tamanoir : noeud actif hautes performances

4.5.1 Dans l'espace utilisateur

Un nœud actif Tamanoir simple ou *Tamanoir Active Node* (TAN) (figure 11) est constitué de trois éléments : un Environnement d'Exécution (EE), un gestionnaire local appelé *Active Node Manager* (ANM) et un ensemble de services.

L'environnement d'exécution **EE** peut recevoir des flux de données au format ANEP ou sous forme de données brutes (*raw data*) transportés avec TCP ou UDP. L'EE agit comme un serveur parallélisé (multi-threads) persistant en attente de connection. Il y a donc un thread en attente pour chaque canal de communication possible : ANEP/UDP, ANEP/TCP, raw/UDP, raw/TCP. Les en-têtes des paquets au format ANEP sont lus par un démultiplexeur. Celui-ci extrait, de l'en-tête ANEP, l'identificateur du service requis par la charge utile du paquet et redirige le paquet ANEP vers le service adéquat. Si le service requis est indisponible le démultiplexeur émet une requête au gestionnaire local (ANM).

Le gestionnaire local **ANM** se charge de déployer les services manquants. Tout d'abord, l'ANM interroge le système de fichier local dans un répertoire convenu pour vérifier si le service est déjà disponible dans le cache local. Si non, en fonction de la position du TAN dans l'infrastructure active, c'est à dire si il est le premier TAN traversé par le flux ou un TAN intermédiaire, l'ANM émet une requête au *Service Repository* ou au dernier nœud actif traversé. Le service requis est alors téléchargé et stocké dans le cache local. Il est ensuite chargé en mémoire grâce à un mécanisme de chargement dynamique. Finalement le service est instancié et prêt à traiter les paquets ANEP qui exigent ce service. Le déploiement d'un service est réalisé sur un canal `TCP` distinct.

Les **services** sont chargés et branchés à la demande dynamiquement dans l'EE. On appelle un service Tamanoir du code Java qui hérite d'une classe générique *Service* elle même dérivée de la classe standard *Thread*. La classe générique *Service* fournit quelques méthodes indispensables à tout service. Ces méthodes peuvent être surchargées par le développeur du service.

L'architecture multi-threadée de Tamanoir permet la gestion simultanée de flux UDP et TCP. En fonction du mode employé, les paquets actifs ne sont pas injectés de la même manière dans le TAN. En mode proxy les flux sont émis sur un port défini du TAN. En mode bout en bout un

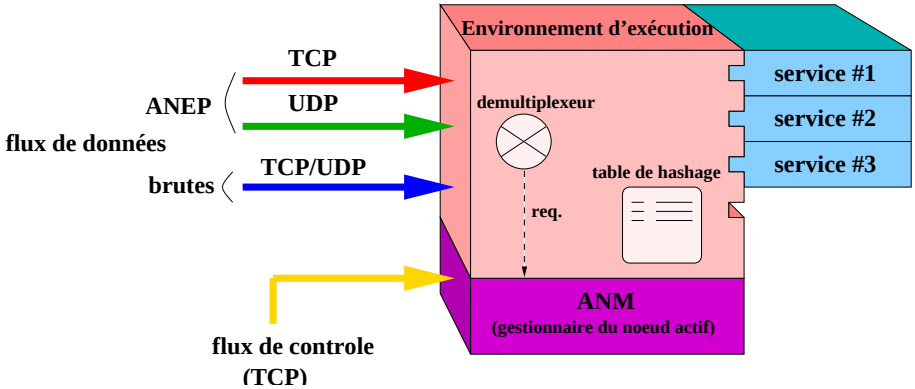

FIG. 11 – Un nœud actif Tamanoir.

filtre sur le bit router alert exécuté dans le noyau capture les paquets actif pour les injecter dans le TAN. Quel que soit le mode employé les flux peuvent être de type ANEP ou brute. Dans le second cas on associe aujourd'hui un service à un numéro de port. Le TAN est constitué de deux threads de démultiplexage. Ces deux threads agissent comme des serveurs en attente de messages sur les deux principaux protocoles de transport disponibles au-dessus d'IP : les protocoles TCP et UDP. Lorsqu'un paquet ANEP atteint le nœud Tamanoir, l'en-tête du paquet est analysé par le thread de démultiplexage afin de déterminer le service qui doit appliquer un traitement sur la charge utile du paquet. Dans le cas d'un flux transporté par UDP le service est instancié une unique fois en mémoire. Il est ensuite chargé de traiter tous les datagrammes qui requièrent ce service sans pouvoir tenir compte (facilement) du fait qu'un datagramme appartient à un flux précis. On devra limiter les informations statistiques par exemple au nombre de datagrammes traités par ce service sans pouvoir caractériser le comportement d'un flux particulier. Les datagrammes de tous les flux doivent passer par le démultiplexeur.

Lorsque le premier paquet d'un flux transporté par le protocole TCP atteint son thread de démultiplexage, le principe reste, dans un premier temps, le même. Le service threadé est chargé en mémoire. Mais, contrairement au cas d'un flux UDP, à chaque arrivée d'un nouveau flux TCP, l'instanciation d'un nouveau service threadé en mémoire est déclenchée (même si ce nouveau flux nécessite un service identique aux précédents). On profite du mode connecté du protocole TCP car les paquets ANEP suivants appartenant au même flux sont directement dirigés vers le service instancié sans avoir à passer par le démultiplexeur. Le grand avantage de cette approche ne réside pourtant pas dans ce dernier point mais plus au niveau du service lui-même. Il nous est maintenant possible d'associer des variables d'états à chaque flux. Ces variables nous permettront alors dans un second temps d'analyser avec une fine granularité le comportement de chaque flux.

Ainsi, le démultiplexeur UDP est utilisé pour orienter chaque nouveau datagramme UDP vers l'unique instance du service alors que le démultiplexeur TCP est utilisé une unique fois, lors de l'établissement d'un nouveau flux. Les paquets sont alors ensuite à la charge de l'instance du service associé à ce flux. Le service pourra alors analyser le comportement du flux entre deux nœuds actifs.

4.5.2 Dans l'espace noyau

Dès la version 2.4, le noyau Linux s'est enrichi de fonctionnalités dans le domaine du filtrage des paquets, de la translation d'adresse (NAT ou *Network Address Translation*) et de la modification des paquets à la volée. Les outils disponibles rendent possible la réalisation de passerelles et de pare-feu performants. Afin de pouvoir réaliser de telles opérations, le canevas Netfilter a été développé. Ce canevas est situé dans le noyau Linux soit en tant que module (chargé au moment souhaité) ou bien lié statiquement au noyau. Pour un utilisateur avancé, le canevas permet de réaliser les trois opérations suivantes :

– Filtrer des paquets, principalement pour assurer des fonctions de pare feu ;
– Réaliser des translations d'adresses (NAT) : cette fonction est utile lorsque l'on veut faire communiquer un réseau privé avec l'Internet ;
– Marquer des paquets, pour leur appliquer un traitement spécial : cette fonction permet d'appliquer des priorités différentes aux paquets dans les files d'attentes d'un équipement réseau.

Le canevas Netfilter n'a pas la même structure en fonction du protocole employé (IPv4, IPv6). Chaque protocole définit des *hooks* (accroches) qui sont des points précis dans le trajet du paquet dans la pile du protocole. IPv4 en définit 5 (voir Fig. 12).

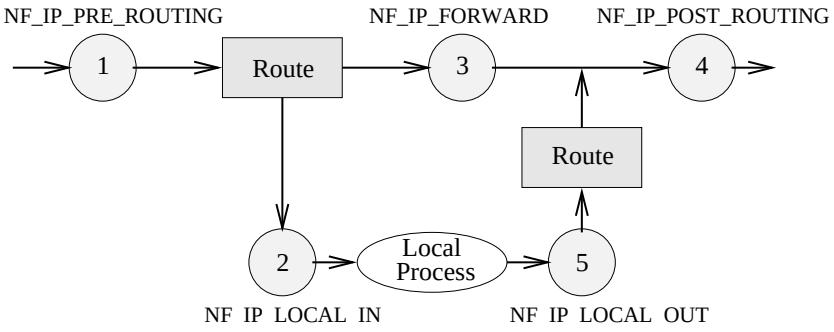

FIG. 12 – Hooks de Netfilter

Comme nous l'avons décrit dans le chapitre précédent, nous classons les services en fonction des ressources qu'ils consomment et de ce fait plaçons leur exécution dans l'espace le mieux adapté. Nous appelons services noyau des services actifs légers exécutés dans le noyau du système d'exploitation *Linux* , sous forme de modules écrit en C et chargés dynamiquement dans l'espace noyau grâce à Netfilter. Netfilter nous permet ainsi l'accroche de services actifs pour le traitement de paquet actifs dans l'espace noyau.

Nous mettons en oeuvre des mécanismes de communications entre un service Tamanoir écrit en Java et s'exécutant dans l'espace utilisateur, et un service écrit en C exécuté dans l'espace noyau (Figure 13). Cette solution autorise la mise en oeuvre de services actifs multi-couches et est utile pour alléger la charge de traitement de Tamanoir dans l'espace utilisateur. Un service noyau peut ainsi recevoir des instructions du service Java pour éviter la remontée de certains paquets dans l'espace utilisateur. Avec une politique *best effort* les paquets non traités dans un TAN pourront être traités dans le noeud actif suivant.

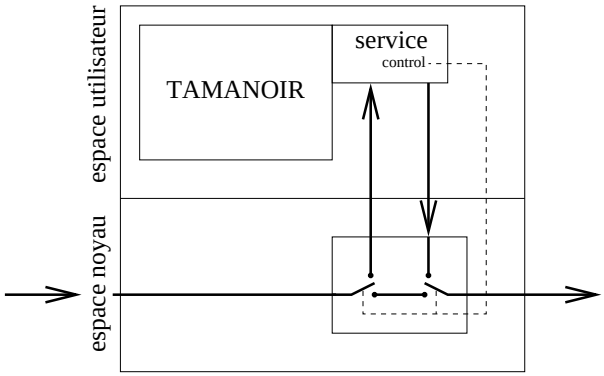

FIG. 13 – Mécanisme de communication entre le service exécuté dans l'espace utilisateur et son homologue exécuté dans l'espace noyau

La figure 14 illustre ce mécanisme facultatif de remontée des paquets où le service exécuté dans l'espace utilisateur traite 500 paquets puis demande au service noyau de ne pas faire remonter les 500 paquets suivants. Sur cette figure on voit également que la taille du paquet influence peu le temps de traversée du noyau mais de manière significative le temps de traversée de l'espace utilisateur (entre 5 et 9 ms).

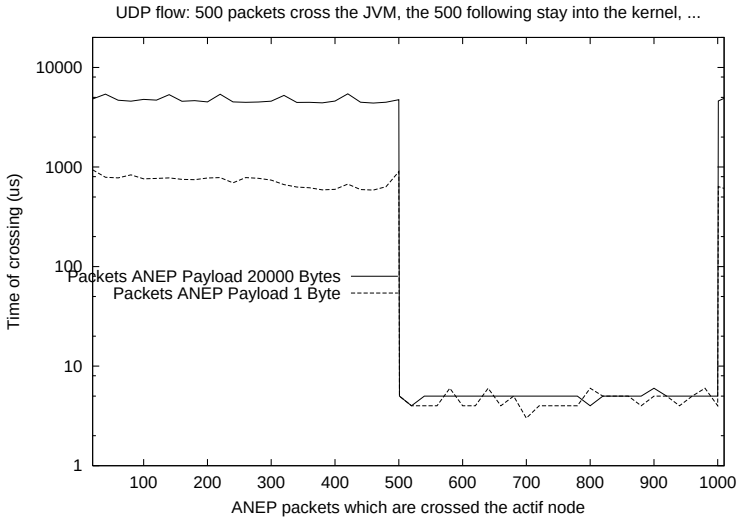

FIG. 14 – 500 paquets traversent la JVM, les 500 suivants sont transmis par le noyau directement sur l'interface de sortie.

Le TAN exécuté dans l'espace utilisateur est utilisé pour déployer et charger dynamiquement de nouveaux services actifs dans l'espace noyau, par un appel à des commandes du système (*insmod*).

4.5.3 Dans l'espace des ressources distribuées

Nos objectifs sont d'accélérer l'exécution de services sur les flux actifs en parallélisant le traitement et d'augmenter la capacité de stockage d'un nœud actif. Pour cela, un nœud actif Tamanoir s'appuie sur système distribué de type grappe de PC (cluster). Pour réaliser cette architecture répliquée, nous avons détourné et adapté la fonctionnalité originelle du projet LVS (Linux Virtual Server)[Zha00] qui est de distribuer des requêtes sur une batterie de serveurs, pour distribuer la charge de traitement des paquets actifs.

Un serveur virtuel Linux (*Linux Virtual Server* : LVS) [Zha00] est constitué d'un groupe de serveurs, appelés *realservers* et d'un directeur (ou *director*). Cet ensemble de machines forme le serveur virtuel qui apparaît comme une machine unique pour les clients. On appelle directeur (ou *load balancer*, *frontend* (FE) ou frontal) le nœud qui exécute le code `ipvs` qui se présente sous la forme d'un module Netfilter. Les clients se connectent au directeur. Le directeur transmet les paquets au realservers. Le directeur n'est autre qu'un routeur avec des règles spécifiques qui font fonctionner le LVS. Les *realservers* ou *backends* (BE) permettent le déploiement des services. Ils gèrent les requêtes des clients. Un LVS offre des services de plus grande capacité en terme de débits et de disponibilités par rapport à un serveur unique.

Chaque client croit être connecté directement au *realserver*. Ni les clients, ni les *realservers* n'ont le moyen de détecter qu'un *director* est intervenu dans la connexion.
Un LVS n'est à la base pas une grappe de machines destinée à calculer des petites parties d'un grand problème de façon coopérative. Les *realservers* ne coopèrent pas, ils n'ont pas connaissance de la présence de leurs voisins et sont connectés à un client. LVS propose de transmettre les paquets de trois manières différentes :

– **LVS-NAT**, basé sur la translation d'adresse (NAT) ;

– **LVS-TUN** (TUNnelling) où le paquet est IPIP encapsulé et transmis au *realserver* ;
– **LVS-DR** (Direct Routing) où les adresses MAC des paquets sont changées et le paquet transmis au *realserver*.

L'usage principal de LVS est de distribuer la charge d'un serveur Web, DNS ou Proxy Squid en distribuant les connexions sur un ensemble de realservers. Dans le cas d'un serveur Web par exemple, chaque realserver est un miroir du serveur web. Le director se charge de distribuer les requêtes `GET` provenant des navigateurs des clients sur l'un des realservers qui honorera alors la requête. Dans le projet Tamanoir nous détournons l'usage de LVS pour distribuer non plus des requêtes Web mais des flux TCP/UDP de paquets actif. Le director est employé à rediriger chaque nouveau flux vers un realserver qui exécute un EE Tamanoir indépendant. La figure 15 illustre l'architecture d'un Tamanoir cluster où l'EE Tamanoir exécuté dans l'espace utilisateur est dupliqué sur chaque realserver. Parmi les trois modes de traitements distribué que nous proposions dans le chapitre précédent, c'est donc le troisième cas décrit figure 4-c que nous avons choisi de mettre en œuvre à l'aide du projet LVS.

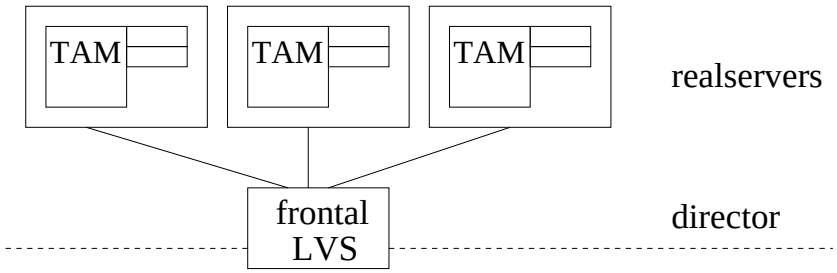

FIG. 15 – Un nœud actif Tamanoir cluster : l'EE Tamanoir de l'espace utilisateur est dupliqué sur plusieurs nœuds (realservers ou backends) auxquels on accède à travers un frontal (director) qui distribue les connections.

Lorsqu'un flux actif est dirigé par le frontal vers l'un des BE, le même scénario de déploiement de service que pour un EE Tamanoir classique exécuté dans l'espace utilisateur s'applique. Cela signifie qu'un BE pourra appliquer un service que son voisin ne possède peut-être pas encore. Nous n'avons pas encore prévu à ce jour de mécanismes de déploiement de services intra-nœud.

Le director exécute un module de filtrage sur les paquets TCP et un port destination qu'on lui a spécifié. Lorsqu'une nouvelle connexion est établie, l'algorithme de distribution du module lui associe un realserver à qui il transmet ensuite les données. Le module tient donc à jour une table des couples identificateurs de flux, realserver.

5 Expérimentations

Dans ce chapitre nous commençons par décrire les différentes plateformes de tests utilisées tout au long des expérimentations de Tamanoir. Ensuite nous présentons les performances en latence et débit brut obtenus avec un nœud Tamanoir simple. Nous verrons l'apport d'une architecture distribuée sur deux types de services. Enfin nous terminerons sur des mesures réalisées avec une plateforme Tamanoir longue distance.

5.1 Description des plateformes matérielles et logicielles de tests de Tamanoir

5.1.1 Plateformes matérielles

Différentes plateformes matérielles ont été utilisées :
– **Plateforme 1** : Cette première plateforme de test, financée par l'ANVAR, est constituée de trois PC bi-processeur Pentium III cadencés à 1GHz qui sont utilisés comme nœuds actifs

Tamanoir. Les applications clientes sont déployées sur des PC AMD Athlon 1GHz. Cette plate-forme est interconnectée par un réseau FastEthernet .

- **Plateforme 2** : Cette seconde plateforme de test, déployée dans le cadre du projet RNRT VTHD++ [Pro], est constituée de sept Compaq DL360 (G2) Proliant, bi-PIII à 1.4GHz avec un bus PCI à 66MHz. Ils sont interconnectés par un réseau Gigabit Ethernet cuivre sur un commutateur Gigabit *Foundry*.
- **Plateforme 3** : Cette troisième plateforme, déployée dans le cadre du projet RNTL EToile [Eto], est une grappe constituée de 16 SUN LX50, bi-PIII à 1.4GHz avec un bus PCI à 66MHz. Les nœuds de calculs sont interconnectés par deux réseaux : FastEthernet et Myrinet [BCF+95] (réseau Gbit de type (*System Area Network*) très haut débit, très faible latence).
- **Plateforme 4** : Cette plate-forme est basée sur une connexion directe à la plate-forme VTHD (Vraiment Très Haut Débit) avec un lien Gbit. Ce réseau relie de nombreux sites de recherche français comme l'INRIA Rocquencourt et Sophia, l'ENS de Lyon, les SUN labs/Europe à Grenoble, l'ENST, Eurecom ainsi que les différents centres de recherche de France Telecom (FT R&D). Les machines utilisées dans cette plate-forme dépendent des sites interconnectés pendant les expérimentations.

5.1.2 Plateformes logicielles

Les premières expériences ont été réalisées sur différentes machines virtuelles Java. Celle de SUN[Mic], d'IBM[IBM], la version libre Kaffe[Kaf] et celle proposée par l'organisation Blackdown[tol]. Nous avons également mené des expérimentations sur une version Tamanoir toujours écrite en Java mais compilée en code natif à l'aide de GCJ (GNU Compiler for Java)[GCJ]. Les mesures les plus récentes ont été réalisées à l'aide de la JVM J2RE 1.4.2 fournie par SUN et incluant la technologie Hotspot [hot]. Certaines mesures ont été réalisées avec un prototype de Tamanoir (version 0.1) ne bénéficiant pas de toutes les optimisations disponibles dans la version actuelle (Tamanoir version 1.0). Toutes les mesures présentées dans cete section ont été réalisées sur des plateformes animées par le système d'exploitation GNU/Linux (distribution *Debian*).

Nous avons focalisé nos expérimentations sur deux types de services (léger et lourd) caractéristiques des fonctionnalités dynamiques déployables sur les noeuds Tamanoir :

- Un **service léger** consiste à compter les paquets pour chacun des flux. Il ne consomme que très peu de mémoire puisqu'il ne possède qu'une variable d'état, le compteur.
- Un **service lourd** se propose de compresser la charge utile des paquets actifs. C'est un service très exigeant en ressource CPU. Sa consommation mémoire se limite à allouer une nouvelle zone mémoire de la taille de la charge utile pour chaque nouveau paquet. Une fois celle-ci compressée la zone mémoire allouée est libérée.

Pour le besoin des tests nous avons créé deux outils efficaces, entièrement écrits en Java. Le premier outil est un récepteur de flux de données `udp` ou `tcp`. Le flux de données reçu est redirigé par défaut sur la sortie standard (`stdout`). Le second outil permet d'émettre un flux de données au format ANEP. La charge des paquets ANEP émis peut être fixée ou peut transporter le contenu d'un fichier. Dans le premier cas un paquet ANEP unique est créé et ensuite expédié autant de fois que nécessaire, dans le second cas il suffit de passer en paramètre le nom du fichier (texte ou binaire) à émettre. La taille de la charge utile des paquets ANEP, le protocole de transport à employer, le nom du service à appliquer aux paquets actifs sont paramétrables dans l'outil.

5.2 Mesures de latence

Pour mesurer l'efficacité de notre EE nous déployons un service qui applique un traitement extrêmement léger, en terme de consommation CPU et mémoire, sur la charge utile du paquet actif. On appelle **latence d'un nœud actif** le temps total mis par un paquet actif pour le traverser.

Les mesures de latence présentées dans les figures 16 et 17 ont été réalisées sur la plateforme 1 avec deux JVM (IBM et SUN 1.3) ayant le compilateur *Just-In-Time* activé ainsi que sur une version compilée d'un TAN avec GCJ. En abscisse nous faisons varier la taille de la charge utile

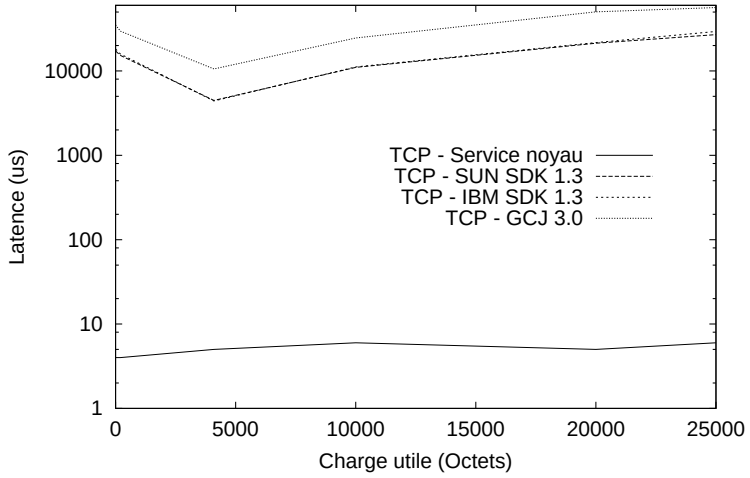

FIG. 16 – Temps de traversée d'un nœud actif Tamanoir par un paquet ANEP sur `TCP`.

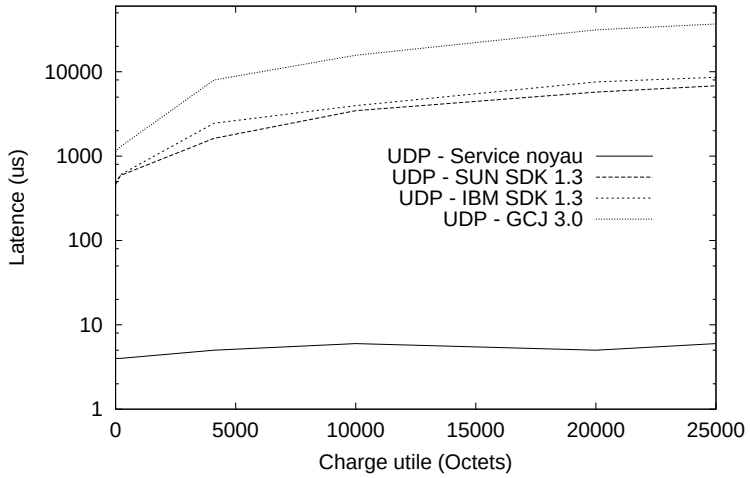

FIG. 17 – Temps de traversée d'un nœud actif Tamanoir par un paquet ANEP sur `UDP`.

jusqu'à 25ko, en ordonnée, les temps de traversée (latence) sont données en μs. Le service appliqué est extrêmement léger puisqu'il se contente d'estampiller et de compter les paquets ANEP. La courbe *TCP-Service noyau* représente la latence induite par un routage noyau classique. Ici un service actif noyau est appliqué. Les paquets sont simplement comptabilisés et transmis sur l'interface de sortie le plus rapidement possible. Les valeurs obtenues oscillent autour de 5 à 7 μs et semblent indépendantes de la taille du paquet. Quelle que soit la JVM employée les résultats obtenus restent similaires. Seule la version compilée par GCJ se démarque nettement en offrant des résultats plus faibles. Ces faibles résultats obtenus avec GCJ pour une exécution en mode natif (compilé) s'expliquent par le manque de maturité du compilateur qui produit un code natif peu optimisé en performances. Pour un flux `UDP`, la latence obtenue avec les deux JVM reste inférieure à 10 ms et atteint de bons résultats pour des paquets de petite taille : 7ms pour des paquets de 128 octets. Pour un flux `TCP` on constate des performances plus faibles pour des paquets de petite taille. Cela s'explique par le fait que l'implémentation de la couche `TCP` du noyau Linux tente d'agréger les paquets de petite taille avant de les émettre. Dans ce cas, la première partie de la courbe reflète les temps d'expiration *(timeout)* avant émission. Ensuite la courbe augmente de manière régulière.

Ainsi, bien que la latence de traversée d'un paquet `TCP` soit sensiblement supérieure à celle d'un paquet `UDP` on constate que cela reste dans le même ordre de grandeur. Par contre, on constate un facteur d'environ 1000 entre un routage noyau simple et un routage dans l'espace utilisateur avec l'application d'un service même très simple. Ces expérimentations ont été menées avec la version 0.1 de Tamanoir qui n'inclut pas toutes les optimisations. Néanmoins elles illustrent bien le fait que la remontée et le traitement d'un paquet dans l'espace utilisateur ont un impact sur la latence.

5.3 Mesures de débits

Nous avons mené de nombreuses campagnes d'expérimentations afin de mesurer le débit d'un noeud actif Tamanoir dans différentes conditions (matérielles et logicielles). Les résultats sont donnés en Mbps. Ils représentent la vitesse moyenne de traitement des données dans un équipement Tamanoir déployant un service actif. Jusqu'à la section 5.3.3 incluse nous utilisons pour les mesures un nœud actif Tamanoir mono machine et plusieurs machines clientes pour émettre ou recevoir des flux actifs. Nous limitons l'émetteur à un maximum de trois flux car au delà le débit brut par flux s'écroule rapidement. Cela s'explique par les conflits qui interviennent lorsque des processus émettent en parallèle des flux à travers un bus système et une carte d'interface réseau unique. Toutes les mesures présentées dans cette section ont été réalisées avec la version 1.0 de Tamanoir.

5.3.1 Mesures de débits sur réseau FastEthernet

Les mesures de débits présentées sur la figure 18 ont été obtenues sur le réseau FastEthernet de la plateforme 3 (grappe SUN). Nous avons déployé un service léger, en faisant varier le nombre de flux. Sur TCP, dès que la charge utile des paquets atteint 1 kB on obtient un débit de plus de 70 Mbps pour un flux unique, près de 45 Mbps par flux lorsque deux flux traversent Tamanoir. Il est à noter que la mise en œuvre actuelle de Tamanoir ne supporte pas l'architecture SMP pour des services appliqués à des flux UDP. On retiendra donc de ces mesures que nous atteignons rapidement le débit maximum proposé par un réseau FastEthernet avec un service léger et un faible nombre de flux sur TCP.

Taille paquets	4K	8K	32K
1 flux	44	54	57
2 flux	61	81	82

TAB. 1 – Débits obtenus (Mbits) sur réseau FastEthernet avec un service lourd.

Les débits présentés Table 1 ont également été obtenus avec le réseau FastEthernet de la plateforme 3. Nous appliquons un service lourd à un ou deux flux de données transportés sur

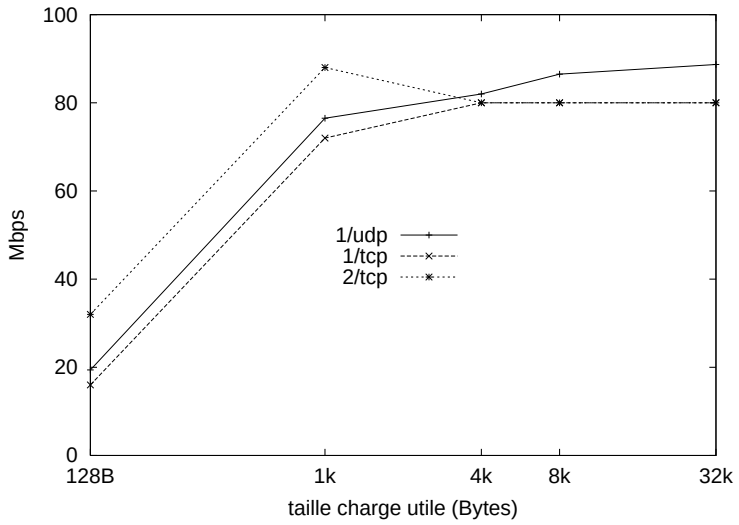

FIG. 18 – Débits obtenus sur réseau FastEthernet avec un service léger.

TCP. On constate que ce service est plus efficace sur des paquets dont la charge utile est d'au moins 8kB. Pour deux flux, nous tirons clairement bénéfice de l'architecture SMP (biprocesseurs) du nœud Tamanoir. Cela laisse présager de l'intéret de distribuer la charge de calcul pour des réseaux plus rapide. Pour deux flux nous atteignons déjà quasiment la bande passante maximum offerte par la technologie FastEthernet.

5.3.2 Mesures de débits sur réseau GigaEthernet

Les mesures de débits présentées dans les figures 19, 20 et 21 sont obtenues sur des machines similaires à celles utilisées précédemment mais cette fois-ci interconnectées par un réseau local GigaEthernet (plateforme 2).

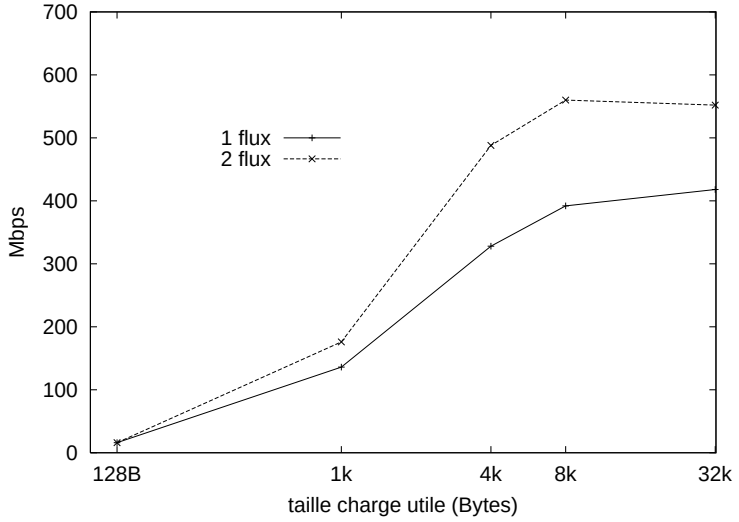

FIG. 19 – Débits obtenus sur réseau Giga Ethernet en TCP avec un service léger.

La figure 19 présente les débits obtenus sur TCP avec un service léger pour un ou deux flux.

Les performances en bande passante benéficient de l'utilisation de paquets supérieurs à 8kB. Nous constate que nous ne saturons pas le lien avec deux flux. Les traitements à effectuer étant très légers, les performances ne bénéficient pas fortement du support multi-processeur avec TCP (gain de 100 Mbps).

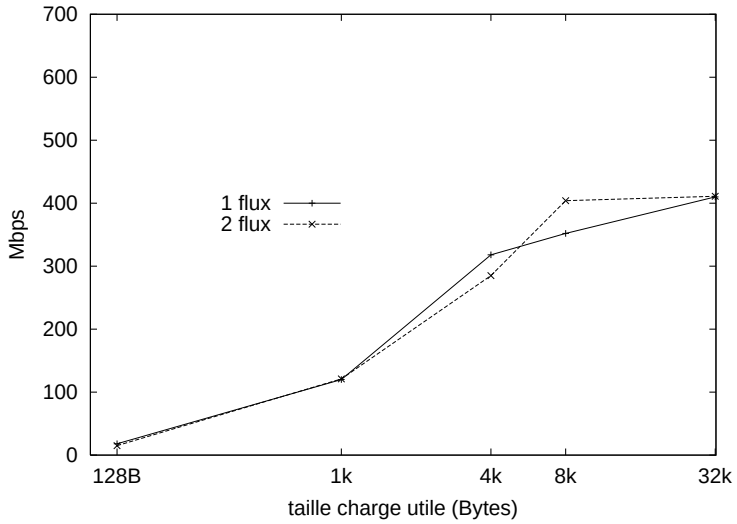

FIG. 20 – Débits obtenus sur réseau Giga Ethernet en UDP avec un service léger.

La figure 20 montre les débits obtenus sur UDP avec un service léger pour un ou deux flux. Les résultats obtenus sont identiques à ceux atteints avec un flux TCP et sont similaires quels que soient le nombre de flux (un seul thread gère les flux UDP dans le noeud Tamanoir).

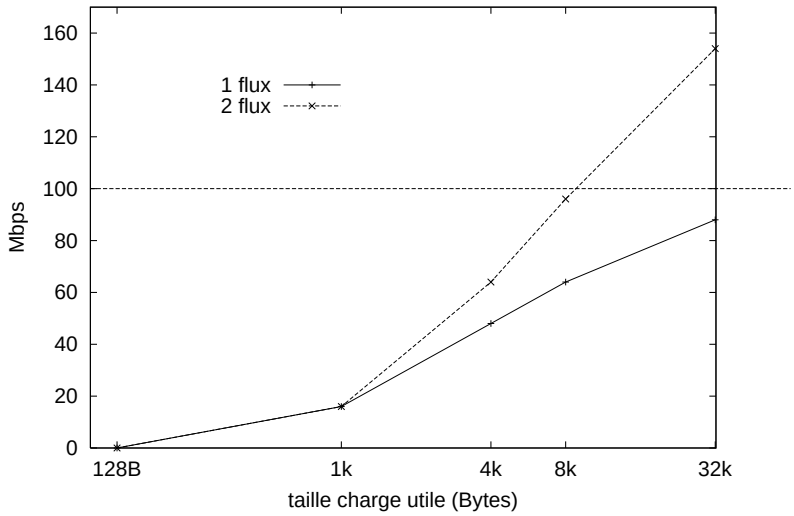

FIG. 21 – Débits obtenus sur réseau Giga Ethernet en TCP avec un service lourd.

Les mesures suivantes dont les résultats sont présentés sur la figure 21 ont été obtenus avec un service lourd appliqué à un ou deux flux sur TCP. Ici les paquets de très grande taille sont bien adaptés. A partir de 8kB pour deux flux on double presque le débit agrégé. On tire donc clairement avantage dans cette expérience de l'architecture biprocesseurs (SMP) du nœud Tama-

noir. On constate que pour un flux nous atteignons près de 90 Mbps qui est la limite d'un réseau FastEthernet, or sur la figure 1 le débit maximum d'un unique flux n'était que de 55 Mbps bien que le service appliqué soit identique et consomme la même quantité de ressources. Cette différence permet donc d'illustrer le temps supplémentaire nécessaire pour accéder à une interface réseau et l'impact de la qualité du pilote réseau sur les performances.

5.3.3 Mesures de débits sur réseau Myrinet

Nous avons effectué une série de mesures préliminaires des débits obtenus sur un réseau Myrinet très haut débit faible latence avec un nœud actif Tamanoir simple qui applique un service léger à un flux unique sur TCP ou UDP. Pour cela nous utilisons la plateforme 3. Nous faisons varier la taille de la charge utile des paquets ANEP entre 128 octets et 32 kB (en abscisse, échelle logarithmique). Le service appliqué à chaque flux est un service léger. Les résultats de ces mesures sont présentés figure 22. On constate que le débit obtenu avec TCP est légèrement supérieur au débit obtenu avec UDP. Ceci est dû à l'implémentation de Tamanoir qui, lors du transport d'un flux TCP, ne nécessite pas le passage à un démultiplexeur (les paquets étant directement envoyés au thread du service actif). Ces courbes peuvent être comparées à celles obtenues sur un réseau GigaEthernet, illustrées figures 19 et 20 afin de mettre en évidence l'impact de la technologie réseau employé. Myrinet atteint 600Mbps alors que la technologie GigaEthernet fournit seulement autour de 500Mbps pour des paquets entre 8kB et 32kB.

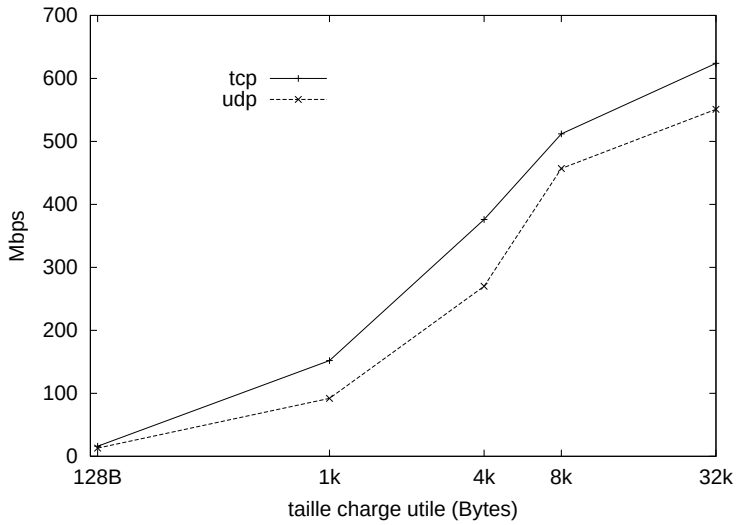

FIG. 22 – Débit obtenu sur réseau Myrinet pour un flux traité par un service léger sur UDP ou TCP.

5.3.4 Mesure de débits sur Tamanoir cluster

Dans cette section en plus d'évaluer Tamanoir sur un réseau local très haute performance, nous évaluons le bénéfice d'utiliser une architecture distribuée dans un nœud actif Tamanoir. L'expérience consiste à charger le plus possible le nœud Tamanoir constitué de plusieurs back-ends (BE) et d'un frontal chargé de distribuer les flux.

Nous ne présentons ici que les résultats obtenus avec l'approche DR (*Direct Routing*). Bien que l'approche TUN (*Tunneling*) proposée par LVS offre sensiblement les même performances celle-ci est plus difficile à mettre en place. Enfin les mesures réalisées avec l'approche NAT ont été décevantes en reportant le goulot d'étranglement sur la machine frontale.

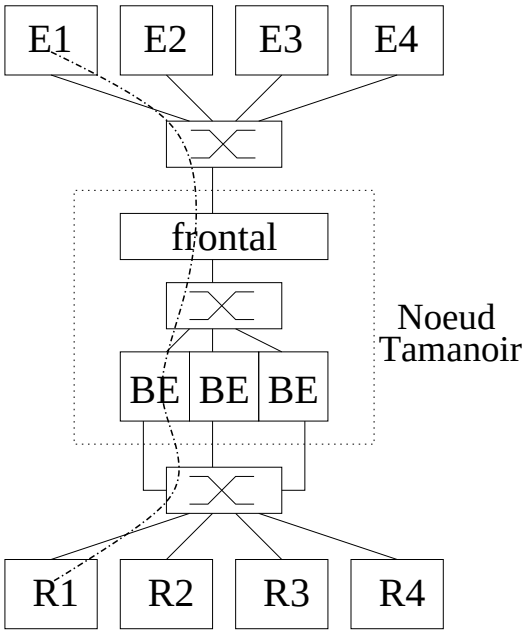

FIG. 23 – Vue logique de la plateforme d'expérimentation Tamanoir cluster sur Myrinet.

La figure 23 présente une vue logique de la plateforme d'expérimentation locale. Elle met en évidence le parcours des paquets ANEP émis par un émetteur E vers un récepteur R. On constate que les paquets traversent trois commutateurs. C'est le frontal qui est en charge de sélectionner un BE.

Les figures 24 et 25 présentent les résultats d'un service léger appliqué par un nœud Tamanoir cluster à trois Back Ends (BE) sur un à douze flux TCP simultanés. Les meilleurs résultats sont obtenus pour 3 flux avec des paquets de 8 à 32 kB. Sur la figure 24, on constate que pour des paquets de 32kB le débit pour quatre flux devient inférieur à celui obtenu avec trois flux. Nous expliquons cela par le fait qu'à un tel débit il se produit des phénomènes difficilement maîtrisable notamment à travers une VM. Le noeud Tamanoir doit gérer plus de conflits pour quatre flux et les paquets de 32kB placent l'environnement dans une situation défavorable.

Dans la figure 25, pour des paquets de 8 à 32kB plus le nombre de flux est important plus le débit agrégé se réduit. Contrairement à ce que l'on pourrait supposer cette expérience ne met pas en évidence la limite du nœud Tamanoir mais celle des clients émetteurs. Bien que les émetteurs soient des machines performantes, bi-processeurs, le nombre optimal de flux qu'elles devraient être à même d'émettre est de deux, soit un flux par processeur. Nous avons généralement employé trois machines pour émettre et trois machines pour recevoir. Donc le nombre de flux optimal est de six, soit deux flux par émetteur. Mais les émetteurs sont équipés d'une carte d'interface réseau unique. C'est donc dans celle-ci que des contentions apparaissent et ainsi dégradent les performances globales. Il est donc nécessaire d'avoir autant de postes émetteurs que de flux. Le nombre de postes récepteur est moins critique, car la réception de données est beaucoup moins coûteuse en cycle CPU. Il est donc possible de lancer plusieurs clients récepteurs sur une même machine en lui associant plusieurs adresses IP virtuelles.

La figure 26 présente les résultats d'un service lourd appliqué par un nœud Tamanoir à 3 Back Ends (BE) sur un à douze flux TCP simultanés. Les débits agrégés bénéficient de la taille des paquets. Pour 6 ou 12 flux, les débits affichés par les courbes sont sensiblement identiques. Cela montre que pour 6 flux nous avons atteint la capacité de traitement maximum pour cette configuration de nœud actif (à 3 BE) et ce service. (débit maxi de 270Mbps). On tire partie de l'architecture bi-processeur car de 3 à 6 flux nous doublons littéralement de débit.

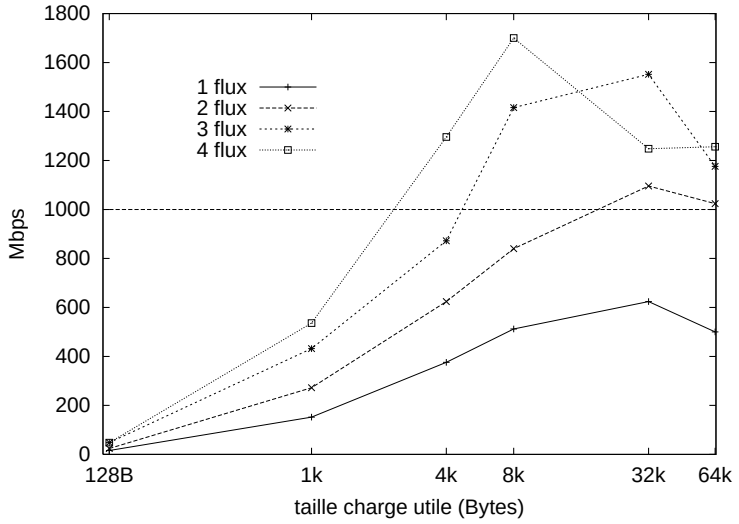

FIG. 24 – Débits sur réseau Myrinet avec un Tamanoir cluster à 3 BE appliquant un service léger.

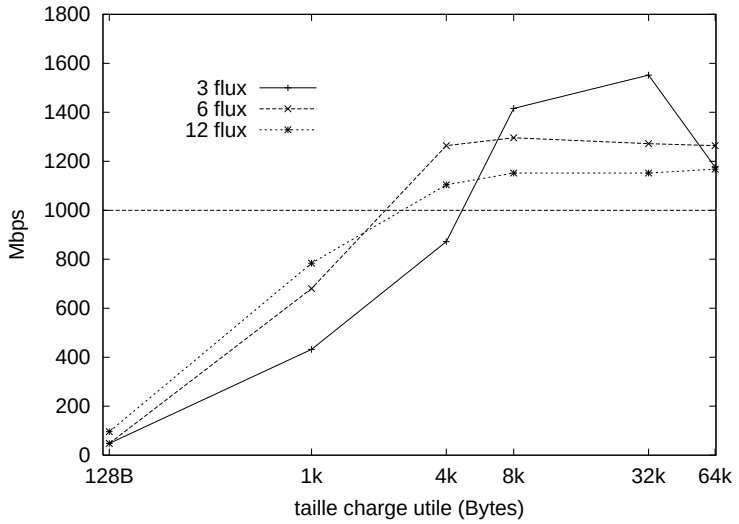

FIG. 25 – Débits sur réseau Myrinet avec un Tamanoir cluster à 3 BE appliquant un service léger.

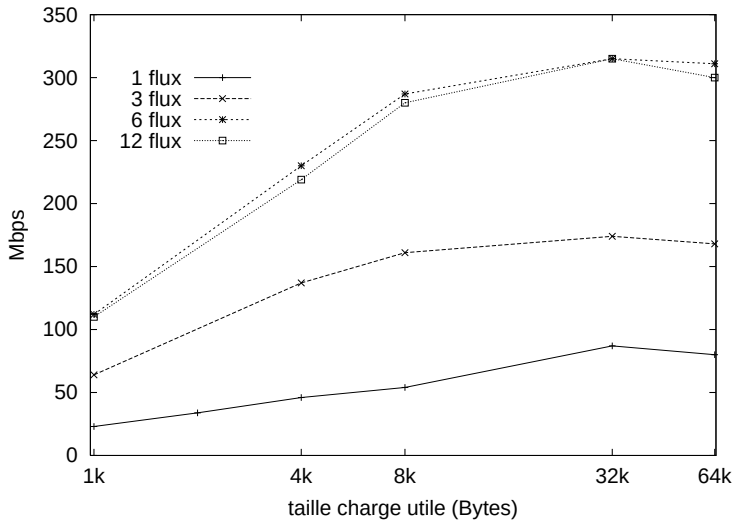

FIG. 26 – Débits sur réseau Myrinet avec un Tamanoir/3BE appliquant un service lourd.

5.3.5 Mesures de débits sur réseau Gigabit longue distance (VTHD)

La figure 27 illustre les trois sites mis à contribution pour effectuer les mesures qui suivent. Ces sites sont :

- SUN labs Europe localisé à Grenoble et relié à VTHD par un routeur gigabit CISCO 2912G. Ce site met à notre disposition neuf racks *cobalt* (SUN LX50) interconnectés par un réseau FastEthernet sur un commutateur CISCO avec un lien GigaEthernet,
- INRIA de Roqucencourt qui met à notre disposition trente PC interconnectés par un réseau FastEthernet sur un commutateur avec un lien GigaEthernet,
- ENS Lyon reliée également directement à VTHD par un routeur gigabit CISCO 6500. A ce routeur sont connectés en GigaEthernet sept racks Compaq Proliant (plateforme 2) ainsi qu'un cluster de seize racks SUN LX50 interconnecté en FastEthernet par l'intermédiaire d'un commutateur Netgear.

La figure 28 présente les résultats des premières mesures sur le réseau longue distance haut débit VTHD. Nous fixons le nombre de flux à 12 et faisons varier le nombre de Back Ends (BE). Le service appliqué est un service léger. Afin de supporter au mieux les petits paquets, il semble préférable d'employer le plus grand nombre de BE. Par contre pour des paquets de 8kB et plus, il semble inutile de mettre à disposition un nœud Tamanoir avec plus de deux BE puisque nous obtenous déjà environ 900Mbps. Enfin pour les très gros paquets (64kB) le Tamanoir à six BE supporte mieux la charge. Un relevé de la consommation CPU des BE lors de l'expérience à 12 flux distribués sur 6 BE, pour des paquets de 8kB auxquels on applique un service léger montre que les BE sont chargés à seulement 25%. Lors de la même mesure avec seulement 2 BE, ces derniers sont chargés à plus de 90%, cela signifie qu'ils ont atteint leur limite et qu'il n'ont plus de ressources suffisantes pour traiter des flux supplémentaires avec le même service.

La figure 29 permet de comparer les débits obtenus par un service léger et un service lourd. Les machines clientes réparties autour de VTHD génèrent six flux qui sont traités par un Tamanoir cluster à 6 BE. Le débit agrégé du service lourd est d'un peu moins de 200Mbps alors que le service léger oscille autour de 500Mbps. La courbe supplémentaire, *1 BE service léger* illustre l'intérêt d'employer un Tamanoir cluster autour d'une épine dorsale hautes performances.

L'objectif ici n'est pas de fournir un service actif performant, optimal dans un contexte de traitement à la volée par exemple, mais de fournir un EE optimal, suffisamment générique pour des services actifs à développer. Si l'on considère la latence introduite par un service léger comme extrêmement faible, les résultats produits ci-dessus mettent en évidence les performances brutes

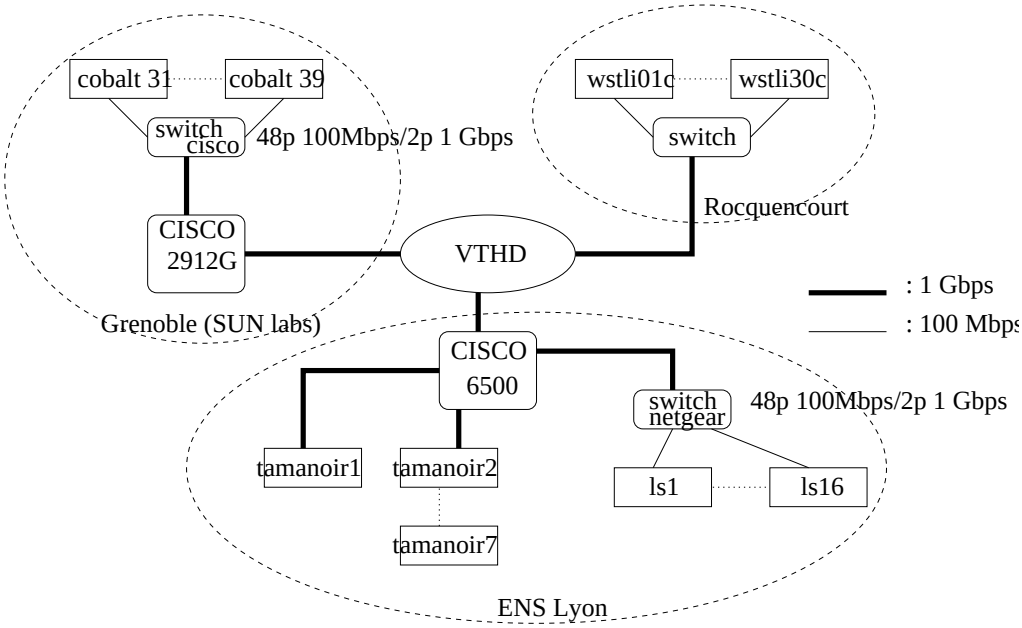

FIG. 27 – Topologie physique de l'environnement d'expérimentations sur VTHD.

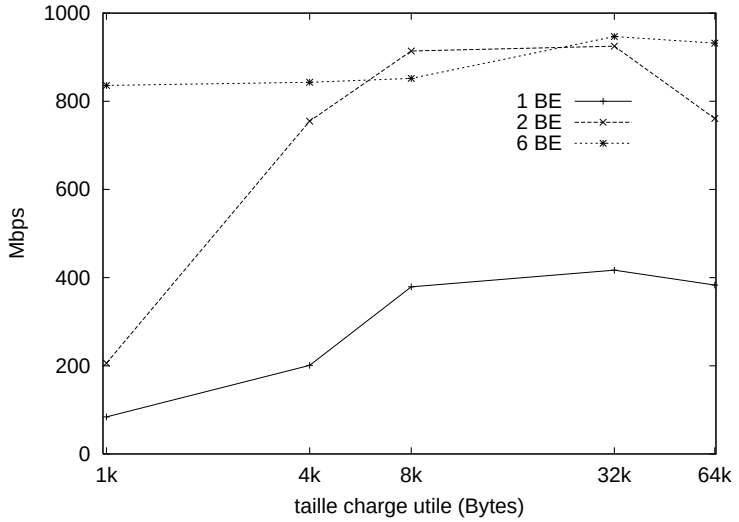

FIG. 28 – Mesures sur VTHD : 12 flux traités par un Tamanoir simple (1BE), 2 ou 6 BE.

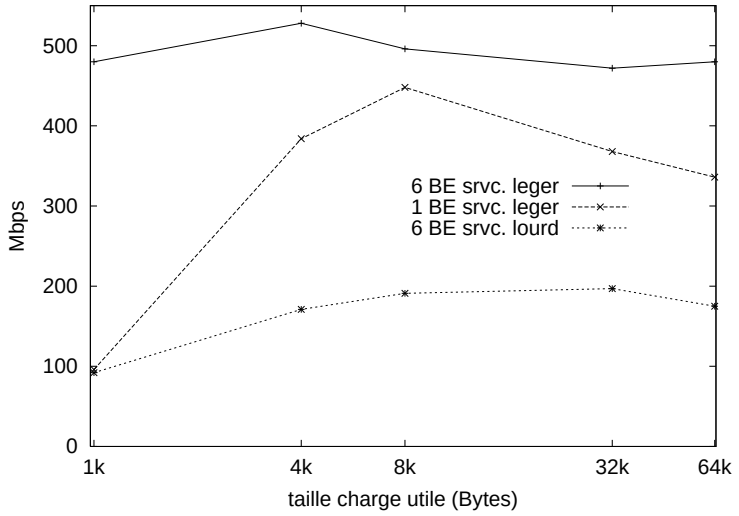

FIG. 29 – Mesures sur VTHD d'un service lourd contre un service léger exécuté sur 6 flux par un Tamanoir Cluster à 6 BE ou un Tamanoir simple.

de l'EE à récupérer un paquet, l'analyser, le transmettre au service adéquat, puis le retransmettre sur le réseau. Le tableau 2 est un récapitulatif des débits maximum atteints avec un service léger

	FastEthernet		GigaEthernet		Myrinet	
	Mbps	pps	Mbps	pps	Mbps	pps
1 flux	80	8759	418	16600	624	18554
x flux	96	9765	560	21484	792	30273
x flux/LVS	-	-	-	-	1700	95703

TAB. 2 – Débits maximum atteints avec un service léger sur TCP.

sur TCP et sur trois technologies réseaux différentes (FastEthernet, GigaEthernet et Myrinet). Les résultats en Mbps ont été obtenus avec des paquets de 8 ou 32kB. Les résultats en paquets par seconde (pps) ont été obtenus avec des paquets de plus petite taille (1kB). Finalement, les expériences menées dans ce chapitre ont montré que nous sommes aujourd'hui capable de supporter les débits des réseaux locaux (LAN) classiques à 100 Mbps. Nous supportons également à l'aide d'une solution issue du calcul parallèle (grappe de PC) les réseaux au Gigabit. Enfin nous avons également montré qu'il est facile de déployer des nœuds Tamanoir sur un réseau longue distance et de mener des expériences autour d'une épine dorsale Gigabit maîtrisée (VTHD).

6 Conclusions et perspectives

Dans cet article, nous nous proposions de résoudre deux problématiques : l'une sur un aspect architectural et l'autre sur un aspect mise en œuvre. Nous avons également proposé une validation expérimentale et fonctionnelle de notre architecture logicielle active.

L'architecture de nœud actif logiciel est basée sur un modèle en couches. Chaque couche correspond à un espace d'exécution des services. La distinction d'une couche par rapport à une autre se fait en fonction des caractéristiques et fonctionnalités qu'elle propose. La couche de l'espace utilisateur autorise un accès aisé à toutes les ressources du systèmes (mémoires et CPU générique). La couche de l'espace noyau et à l'extrême celle d'une carte d'interface réseau programmable sont plus proches du lien. Enfin, la couche des ressources distribuées (grappe) permet de profiter de

l'accélération conférée par le traitement en parallèle des paquets. Remarquons que les services actifs qui font du cache dans le réseau pourront également profiter de la grande quantité de mémoire de masse disponible.

Nous avons ensuite proposé une classification des services en fonctions des ressources qu'ils sont susceptibles de consommer. Nous associons une pondération à chaque type de service (hyper léger, léger, moyen et lourd). Un service est considéré comme lourd s'il est très consommateur en cycle processeur ou mémoire. Un service est léger s'il n'a besoin de garder que quelques états sur le flux et s'il applique un traitement peu consommateur en cycle processeur. Grâce à cette classification nous positionnons l'exécution des services dans les couches appropriées (déploiement vertical). Plus le service est léger (ex : plan contrôle) plus on privilégiera une couche basse, près du lien.

De plus, nous proposons deux modes de communication. Un mode proxy et un mode bout en bout. Le premier repose sur des équipements en attente sur un numéro de port donné. Il permet de lier deux clients par l'intermédiaire de connections indépendantes entre chaque équipement actif. On peut lui reprocher de briser le modèle de bout en bout mais il a pour avantage de faciliter le déploiement de nœuds actifs, et en adressant explicitement leur paquet à un équipement actif, les applications s'assure que le flux passe par un équipement actif.

Le second mode de communication repose sur des équipements autonomes non déclarés auprès des applications. Ce mode propose d'attraper les paquets à la volée quand il passe dans un équipement actif. Cela implique de repérer les paquets d'un signe distinctif pour qu'ils soient capturés et traités.

La mise en œuvre de notre architecture nous a conduit à l'écriture du premier Environnement d'Exécution (EE) logiciel actif haute performance, appelé *Tamanoir*. Il constitue une mise en œuvre pragmatique de notre architecture de nœuds actifs logiciels haute performance.

Pour l'espace utilisateur nous avons écrit un EE complet et une librairie de services en Java. Nous avons utilisé Java pour répondre au problème de l'hétérogénéité importante dans ce type d'application distribué. Un service peut donc être chargé dynamiquement et exécuter sur n'importe quel équipement quel que soit sont architecture matérielle et logicielle pourvu qu'une machine virtuelle Java soit disponible. Bien que l'usage du langage Java puisse faciliter le déploiement il constitue une contrainte supplémentaire pour une mise en œuvre de notre architecture dans un contexte haute performance, car Java propose une abstraction importante du système sous-jacent.

Notre EE Tamanoir est un des rares nœuds actifs disponibles qui supporte le protocole de transport TCP. Cette caractéristique est maintenant nécessaire pour certaines applications qui exige de communiquer leurs flux sur des liaisons fiables. On tire avantage de l'aspect connecté de TCP en associant une instance de service par flux. Cela permet de maintenir des variables d'état sur chaque flux.

On profite du mécanisme de chargement dynamique de modules et des caractéristiques offertes par *Netfilter* disponible dans le système d'exploitation Linux pour l'écriture de services légers exécutés dans l'espace noyau. Ces services sont écrits en C ce qui limite la portabilité au profit des performances. Les services noyau ont l'avantage d'être exécuté au plus près du lien et d'accéder directement à la zone mémoire où les paquets arrivent, sans passer par une interface et sans changement de contextes. Tous les projets existant concentrent l'exécution de leurs services dans une couche unique de l'équipement : espace utilisateur ou espace noyau. Par exemple, le projet PAN[LJK99b] propose les deux mises en œuvres, mais sous la forme de deux prototypes indépendants. Notre architecture propose de déployer un même service sur plusieurs couches de l'équipement actif afin de placer les différentes fonctionnalités du service dans les couches les mieux appropriées. On utilise la portion de service exécuté dans l'espace utilisateur pour assister le déploiement de la portion de service exécuté dans l'espace noyau. Un mécanisme de communication inter couche est également prévu afin que les deux portions d'un service communique. La portion d'un service exécutée dans une couche haute peut émettre des consignes vers ses fonctions simples et rapides exécutées dans l'espace noyau.

En profitant de notre expérience du monde du calcul distribué, nous avons proposé un mo-

dèle qui exploite des solutions issues des systèmes distribués destinés à paralléliser l'exécution d'applications et fournir un nœud actif extensible en performance. en faite deux types d'exécution distribués. L'une basée sur l'architectures SMP, l'autre basée sur l'usage d'une grappe de PC (Tamanoir cluster). Ces deux types d'exécution distribués peuvent être associés en employant des systèmes SMP dans la grappe de PC. Les services lourds sont exécutés dans l'espace utilisateur d'un Tamanoir simple nœud, ou dans l'espace utilisateur de l'un des nœuds d'un Tamanoir cluster. Les services légers sont exécutés dans l'espace noyau, ou dans l'espace utilisateur

Pour la mise en œuvre de la couche des ressources distribuées nous nous basons sur le projet Linux Virtual Server (LVS). Nous proposons de répliquer un nœud Tamanoir simple sur plusieurs nœuds d'une grappe de PC équipé d'une machine frontal destiné à distribuer les flux sur les nœuds de la grappe.

Nous proposons ensuite deux mécanismes de déploiement de services dynamiques qui libèrent un administrateur réseau des fastidieuses tâches de mise à jour des équipements. Le premier utilise un *service repository*, le second emploi un mécanisme de déploiement de proche en proche. L'usage d'un service repository à le défaut de centraliser les services mais facilite l'injection de services approuvés dans le réseau.

Le premier prototype de l'EE Tamanoir est entièrement basé sur du matériel et des logiciels standards. Il est disponible, *open source*, déposé auprès de l'APP (Agence de Protection des Programmes) et exploitable immédiatement dans des applications concrètes.[3]

Pour la validation expérimentale nous avons mené des campagnes de mesures sur quatre technologies de réseaux différentes. Une technologie de réseau locaux classique (100Mbps, Fast Ethernet), une technologies de réseaux locaux haute performance (1000Mbps, Giga Ethernet), une technologie de type SAN (*System Area Network*) très haute performance, faible latence (Myrinet), et une technologie réseau haut débit (1Gbps) longue distance autour d'une épine dorsale (VTHD).

Grâce au projet Tamanoir nous proposons aujourd'hui un équipement actif apte à supporter un lien Gigabit et donc déployable autour d'une épine dorsale au Gigabits. Bien que nous nous soyons focalisé sur la performance nous ne nous sommes pas limités au support de services légers (uniquement destiné au plan contrôle par exemple). Nous fournissons un environnement apte à supporter des services de haut niveau, capable d'agir sur le plan données des paquets. Les résultats obtenus pour un service lourd (compression à la volée) sont prometteurs et montrent l'intérêt d'agréger encore plus de ressources de calcul dans un équipement actif.

La validation fonctionnelle a été réalisée par l'utilisation de la plate forme Tamanoir dans des projets menés par des membres de notre équipe mais également dans des projets conduits par des équipes extérieures. La mise en œuvre de prototypes de protocoles ou services avec la plate-forme Tamanoir leur ont permis d'établir de rapides validations expérimentales. Le retour d'informations de ces utilisateurs a été très positif, a souvent contribué à accélérer l'amélioration de la suite logicielle Tamanoir sur le plan ergonomique et a confirmé l'intérêt de disposer d'une plate-forme d'expérimentations de réseaux actifs ouverte et facilement déployable.

Nous avons mené quelques tests préliminaires de fonctionnalités sur l'aspect multi-noeuds actifs traversés par des flux actifs, mais les mesures présentées dans ce document se sont principalement focalisées sur l'évaluation d'un nœud actif unique, qu'il soit mono-machine ou équipé d'une grappe. Il est nécessaire de poursuivre une évaluation de l'impact de nœuds Tamanoir déployés dans des réseaux de grande envergure sur le chemin des données. Nous pourrons ainsi évaluer l'impact des connections par morceau (en mode proxy) sur les performances.

Il manque également une évaluation d'un nœud actif Tamanoir soumis à un très grand nombre de flux (plusieurs centaines, voire plusieurs milliers) pour observer son comportement en fonction de la technologie réseau utilisé et du positionnement du service dans les couches. Ce genre d'expériences, si elle n'est pas simulée, demande une quantité de ressources matérielles très importante

[3] L'environnement Tamanoir est disponible à : $http://www.ens-lyon.fr/LIP/RESO/Tamanoir$

et est donc difficile à mettre en œuvre. Ces travaux seront menés à l'aide d'un vaste émulateur réseau comme celui mis en place actuellement en France dans le cadre du projet *Grid5000* (ACI GRID)[gri03].

Références

[apa] Apache server project. http ://httpd.apache.org/.

[BCF+95] Nanette Boden, Danny Cohen, Robert Felderman, Alan Kulawik, Charles Seitz, Jakov Seizovic, and Wen-King Su. Myrinet : a gigabit per second local area network. *IEEE-Micro*, 15(1) :29–36, February 1995.

[BHL+99] J. Biswas, J.F Huard, A. Lazar, K. Lim, S. Mahjoub, L.F Pau, M. Suzuki, S. Torstensson, W. Weiugo, and S. Weinstein. Application programming interfaces for networks. *IEEE P1520 WG Draft White Paper*, jan 1999.

[Cal99] K. Calvert. Architectural framework for active networks. Technical report, gatech, jul 1999. www.cc.gatech.edu/projects/canes/papers/arch-1-0.ps.gz.

[DP98] Dan Decasper and Berhard Plattner. Dan : Distributed code caching for active networks. In *INFOCOMM 98*, apr 1998.

[DPC+99] D. Decasper, G. Parulkar, S. Choi, J. DeHart, T. Wolf, and B. Plattner. A scalable, high performance active network node. In *IEEE Network*, volume 13, January 1999.

[Eto] RNTL e-Toile French Grid Project - http ://www.urec.cnrs.fr/etoile.

[FCF00] Olivier Festor, Isabelle Chrisment, and Eric Fleury. Les réseaux programmables 1.0. Technical Report 3913, INRIA, mar 2000. Rapport de Recherche.

[GCJ] GCJ. The gnu compiler for the java programming language. http ://sourceware.cygnus.com/java/.

[GEHL03] Jean-Patrick Gelas, Saad El Hadri, and Laurent Lefèvre. Towards the design of an high performance active node. *Parallel Processing Letters*, 13(2), jun 2003.

[GL00] Jean-Patrick Gelas and Laurent Lefèvre. Tamanoir : A high performance active network framework. In C. S. Raghavendra S. Hariri, C. A. Lee, editor, *Active Middleware Services, Ninth IEEE International Symposium on High Performance Distributed Computing*, pages 105–114, Pittsburgh, Pennsylvania, USA, August 2000. Kluwer Academic Publishers. ISBN 0-7923-7973-X.

[GL02] Jean-Patrick Gelas and Laurent Lefèvre. Performance et dynamicité dans les réseaux : l'approche tamanoir. In *JDIR 2002*, pages 81–90, Toulouse, FRANCE, March 2002. in french.

[GMC+00] Virginie Galtier, Kevin L. Mills, Yannick Carlinet, Stefan Leigh, and Andrew Rukhin. Expressing meaningful processing requirements among heterogeneous nodes in an active network. In *Workshop on Software and Performance*, pages 20–28, 2000.

[gri03] Globalisation des ressources informatiques et des données, programme national grid'5000, 2003. http ://www.recherche.gouv.fr/appel/2003/grid5000.htm.

[hom] DARPA homepage. Active network. http ://www.darpa.mil/ato/programs/ activenetworks/actnet.htm.

[hot] Hotspot technology. http ://java.sun.com/j2se/1.4/.

[IBM] Java IBM. langage java. http ://www-106.ibm.com/developerworks/java/.

[Kaf] Kaffe. machine virtuelle java sous licence gnu. http ://www.kaffe.org/.

[Kat97] D. Katz. RFC 2113 : IP router alert option, February 1997. Status : PROPOSED STANDARD.

[KRGP02] R. Keller, L. Ruf, A. Guindehi, and B. Plattner. Promethos : A dynamically extensible router architecture supporting explicit routing. In *Fourth annual Internatioanl Working Conference on Active Networking*, Zürich, dec 2002. IWAN'02.

[LG03] Laurent Lefèvre and Jean-Patrick Gelas. Active web : active networking support for web transport. In *The second International Workshop on Active Networks Technologies and Applications*, pages 147–156, Osaka, Japan, May 2003. ANTA 2003.

[LJK99a] Erik L.Nygren, Stephen J.Garland, and M.Frans Kaashoek. PAN : A High-Performance Active Network Node Supporting Multiple Mobile Code Systems. In *IEEE OPENARCH '99*, March 1999.

[LJK99b] Erik L.Nygren, Stephen J.Garland, and M.Frans Kaashoek. Pan : A high-performance active network node supporting multiple mobile code systems. In *IEEE OPENARCH '99*, March 1999. http ://www.pdos.lcs.mit.edu/ nygren/pan/.

[Mic] Java Sun Microsystem. langage java. http ://java.sun.com.

[ND02] Hoa-Binh Nguyen and Andrzej Duda. An active node architecture for proactive services. *IWAN2002*, 2002.

[nor] Openetlab home. http ://openetlab.org/.

[OWM00] Max Ott, Girish Welling, and Saurabh Mathur. Clara : A cluster based active router architecture. In *In Proceedings of the Hot Interconnects VIII*, August 2000. Stanford University, CA.

[PJ99] C. Partridge and A. Jackson. RFC 2711 : Ipv6 router alert option, October 1999. Status : PROPOSED STANDARD.

[Pro] Projet RNRT VTHD++. http ://www.vthd.org.

[Riv02] Gary Rivlin. The madness of king george. Wired, jul 2002. http ://www.wired.com/wired/archive/10.07/gilder_pr.html.

[SKE97] Bhattacharjee S., Calvert K., and Zegura E. An architecture for active networking. In *HPN'97*, White Plains, NY, apr 1997.

[tol] Java technology on linux. Blackdown JVM. http ://www.blackdown.org/.

[Wet99] David Wetherall. Active network vision and reality : lessons from a capsule-based system. *Operating Systems Review*, 34(5) :64–79, December 1999.

[WT96] David Wetherall and David Tennenhouse. The active ip option. Technical report, MIT, sept 1996. ACM SIGOPS European Workshop.

[Zha00] Wensong Zhang. Linux Virtual Server for Scalable Network Services. In *Ottawa Linux Symposium*, 2000. http ://www.linuxvirtualserver.org.