Virtualisation de réseau avec Network Emulator et application à l'évaluation d'un réseau recouvrant

Vincent Autefage, Damien Magoni

Virtualisation de réseau avec *N*etwork *Emu*lator et application à l'évaluation d'un réseau recouvrant

Vincent Autefage
Univ. Bordeaux, LaBRI
autefage@labri.fr

Damien Magoni
Univ. Bordeaux, LaBRI
magoni@labri.fr

***Résumé*—L'expérimentation est généralement la dernière phase dans l'élaboration d'une application réseau avant sa diffusion. Malheureusement, il est assez difficile, voir impossible, de posséder une infrastructure physique contrôlée et suffisante, en terme de taille et de performance, afin de tester et valider l'application. Ainsi, la virtualisation est une technique à portée de main pour jouer le rôle de plate-forme de test. Nous proposons un outil appelé *NEmu* ayant pour but de générer des réseaux virtuels dynamiques à la demande afin de tester et de valider des prototypes d'applications réseaux avec un contrôle complet de la topologie et des propriétés des liens. *NEmu* permet la création et la gestion de réseaux virtuels avec des ressources matérielles limitées et sans aucun droit d'accès particulier. Nous proposons également une illustration de l'utilisation de *NEmu* afin de tester l'efficacité d'une application de distribution de fichiers reposant sur un chaînage de connexions TCP. Nous mesurons ainsi l'impact d'un tel chaînage sur les débits et les délais en fonction du nombre de nœuds dans la chaîne, de la taille des paquets et de la bande passante globale des liens.**

***Index Terms*—Réseau virtuel, QEMU, plate-forme d'émulation, chaînage de connexions TCP.**

I. Introduction

L'expérimentation est un point angulaire dans le cycle de développement d'une application, permettant de tester, évaluer et approuver son utilisation ainsi que son degré de maturité.

L'expérimentation sur des algorithmes ou des protocoles peut être faite par *simulation*. Cette technique est réalisée grâce à des simulateurs comme *ns* [1] ou encore *OMNeT++* [2], et permet d'apprécier le bon fonctionnement, l'efficacité ainsi que le passage à l'échelle de modèles d'algorithmes ou de protocoles. L'expérimentation des implémentations diffère dans la mesure où elle opère sur des applications natives et s'intéresse d'avantage à la vitesse d'exécution, à l'utilisati

Il est difficile d'effectuer cette tâche quand l'application repose sur une infrastructure impliquant plusieurs dizaines de machines. En effet, utiliser directement Internet comme plate-forme de test est assez peu praticable dans la mesure ou très peu de paramètres peuvent être contrôlés. Acquérir sa propre infrastructure nécessite d'importants moyens, tant sur le plan financier que spatial. De plus, la topologie d'une telle plate-forme est difficilement dynamique. La virtualisation permet à la fois de réduire les coûts, mais également de simplifier les manipulations. Par ailleurs, il est prouvé que la virtualisation permet de réduire les dépenses énergétiques [3].

Notre solution permet donc de dépasser les contraintes matérielles et ainsi de fournir une infrastructure suffisante permettant de créer des réseaux virtuels. Un réseau virtuel utilise des machines virtuelles, au lieu d'utiliser directement des machines physiques, et les inter-connecte par des liens virtuels dans le but de générer une topologie arbitraire. Ces entités virtuelles peuvent être hébergées sur un ou plusieurs hôtes physiques afin d'outrepasser toute limite matérielle due à la taille de la topologie.

Nous proposons un outil permettant de générer des réseaux virtuels afin de tester et de valider des prototypes d'applications reposant sur des *overlays* (i.e., réseaux dits recouvrants ou superposés) avec un contrôle accru sur les propriétés de la topologie, des nœuds et des inter-connexions. Le but de notre travail est donc de permettre la création d'un réseau virtuel de taille raisonnable en minimisant l'équipement matériel nécessaire à sa réalisation. *NEmu* peut ainsi instancier des *overlays* virtuels en utilisant des émulateurs tiers comme QEMU [4]. Le nom *NEmu*, pour *N*etwork *Emu*lator, fait à la fois référence à sa capacité à générer des réseaux émulés (et non simulés) mais aussi à l'émulateur QEMU qui est utilisé au sein de *NEmu*.

Notre contribution s'articule autour des points suivants :

- Une description détaillée de notre outil *NEmu* permettant de créer et de gérer des flottes distribuées de nœuds et de liens virtuels dans le but d'émuler une topologie réseau arbitraire (Section II).
- Un cas d'utilisation de *NEmu* illustrant l'évaluation des performances d'une application de distribution de fichiers utilisant un *overlay* et reposant sur un arbre de connexions TCP. Nous fournissons ainsi des résultats d'impact sur les débits et les délais en fonction du nombre de nœuds de l'arbre (Section III).
- Un état de l'art sur les travaux similaires dans la virtualisation de réseaux, ainsi qu'une comparaison détaillée des fonctionnalités proposées par *NEmu* et de celles des autres travaux (Section IV).

II. Description de *NEmu*

A. Description générale

NEmu est un programme de 5000 lignes, écrit en python, permettant de créer un réseau distribué de machines virtuelles. Il est basé sur un concept appelé *Network Virtualization Environment* (NVE) [5]. La principale caractéristique d'un NVE est de pouvoir abriter plusieurs *réseaux virtuels* (VN) qui sont indépendants les uns des autres. Par défaut, un VN n'est pas

conscient de l'existence d'un autre VN en cas d'hébergement partagé sur une même infrastructure physique. Un VN est un ensemble de *nœuds virtuels* inter-connectés par des *liens virtuels* formant une topologie réseau émulée. *NEmu* permet ainsi de construire plusieurs VN en assurant une séparation stricte de chaque réseau afin de garantir l'intégrité de chacune des topologies.

Ainsi, *NEmu* intègre l'ensemble des propriétés fondamentales qui définissent un NVE :

- La *flexibilité* et l'*hétérogénéité* permettent à l'utilisateur de construire une topologie arbitraire constituée de nœuds et de liens virtuels qui sont eux mêmes paramétrables.
- L'*extensibilité* (ou *passage à l'échelle*) permet aux différents nœuds d'une même topologie d'être hébergés sur plusieurs hôtes physiques afin d'outrepasser les limitations d'une seule machine physique.
- L'*isolation* garantie une stricte séparation entre chaque VN qui vit au sein de la même infrastructure physique.
- La *stabilité* assure que des erreurs dans un VN ne peuvent affecter un autre VN.
- La *maintenabilité* permet à un VN d'être complètement indépendant de l'infrastructure physique qui l'héberge. Ainsi, ce même VN pourra être re-déployé au sein d'une autre infrastructure.
- Le *support hérité* permet au NVE d'émuler des composants anciens.
- La *programmabilité* fournit des services réseaux auxiliaires facilitant l'utilisation du VN (comme DHCP ou DNS par exemple).

De plus, *NEmu* inclut trois propriétés supplémentaires :

- L'*accessibilité* permet à *NEmu* d'être exécuté sans aucun droit d'accès particulier sur l'infrastructure physique. En effet, la majeure partie des instances publiques, telles que les universités et les laboratoires, ne fournissent pas de droits supérieurs aux utilisateurs sur leur infrastructure afin de prévenir tout risque de sécurité ou d'intégrité sur l'ensemble du domaine. C'est pourquoi *NEmu* fonctionne en espace utilisateur.
- La *dynamicité* de la topologie permet aux nœuds de joindre ou de quitter un VN à tout moment sans perturber celui-ci. Un lien virtuel peut également être instancié ou supprimé à tout moment et à tout endroit dans un VN.
- L'*aspect communautaire* permet à plusieurs utilisateurs d'inter-connecter différents VN dans le but de créer un réseau plus large. Dans ce cas, chaque VN du réseau communautaire peut être considéré comme un système autonome (AS). Un ensemble d'utilisateurs pourra également former un unique AS composé de plusieurs VN.

Par conséquent, *NEmu* peut être considéré comme une *Infrastructure As A Service* virtuelle (*virtual IaaS*) [6], c'est à dire une infrastructure où l'utilisateur est libre de fixer l'ensemble des différents paramètres topologiques, tant au niveau des nœuds que des liens virtuels.

B. Éléments réseaux

NEmu est un générateur de réseaux virtuels distribués qui permet à un ou plusieurs utilisateurs de créer une topologie arbitraire et dynamique. À cet effet, *NEmu* est basé sur plusieurs briques distinctes. En effet, *NEmu* utilise des *nœuds virtuels* inter-connectés par des *liens virtuels* dans le but de créer une topologie réseau virtuelle. Cette topologie peut être hébergée sur une unique machine physique ou bien sur plusieurs hôtes afin de permettre d'étendre la taille du réseau virtuel. La partie du réseau reposant sur un même hôte représente une *session* qui est configurée au travers du *manager NEmu*.

1) Les nœuds virtuels : Un *nœud virtuel* dans *NEmu* est une machine virtuelle émulée qui requiert une unité de stockage principale hébergeant son système d'exploitation. Cette unité est généralement représentée par une image disque présente sur l'hôte physique. Deux types distincts de *nœuds virtuels* existent actuellement au sein de *NEmu* :

- Un `VHost` est une *machine virtuelle utilisateur* entièrement configurable (composants internes, périphériques, système d'exploitation, etc.).
- Un `VRouter` est un *router virtuel* directement configuré par *NEmu* proposant différents services *clés en main* dans le but de simplifier la gestion du réseau émulé.

Chaque nœud virtuel utilise une ou plusieurs *unités de stockage* qui peuvent être représentées par un périphérique physique (disque dur, disque externe, etc.) ou bien par un système virtuel tel que :

- Un *fichier à trou* (appelé aussi *sparse file* ou *CoW*) qui est le clone d'une image disque et qui ne stocke que les différences avec celle-ci.
- Un système de fichier *squash* qui est une image disque en lecture seule.
- Une émulation du système de fichier *FAT16* sur un répertoire de l'hôte physique.
- Une interface au travers de *virtio* [7] sur un répertoire de l'hôte physique qui permet de rendre celui-ci accessible à une machine virtuelle.
- Une image disque accessible à une ou plusieurs machines virtuelles au travers du réseau IP en utilisant le protocole *Network Block Device* (NBD) [8]. Cela permet notamment de différencier l'hôte de l'image disque, de celui de la machine virtuelle.

Un `VHost` nécessite une image disque créée au préalable et fournie par l'utilisateur.

Contrairement à un `VHost`, un `VRouter` est directement configuré par *NEmu*. Son unité de stockage est générée par *NEmu* et contient un ensemble de services facilitant la gestion du réseau virtuel. Ainsi les `VRouter` proposent différents serveurs : DHCP, DNS, NFS, HTTP, SSH, NTP ; des protocoles de routage dynamique comme RIP et OSPF ; un pare-feu Netfilter ainsi qu'une couche QoS avec *Traffic Control* [9]. Il est possible d'inclure de nouveaux services en utilisant un mécanisme de *plugins* extrêmement simple fourni par *NEmu*. Un `VRouter` repose sur une version modifiée de *Microcore* qui est une distribution Linux extrêmement légère et hautement configurable [10]. Un `VRouter` requiert environ $\sim$100 Mo

en mémoire vive en activant la totalité des services. Chaque service peut être activé ou désactivé que le `VRouter` soit actif ou inactif.

2) Les liens virtuels : Un *lien virtuel* est une inter-connexion réseau émulée entre deux *nœuds virtuels*. Cette inter-connexion peut être faite directement au sein de l'émulateur du nœud (le lien sera, dans ces conditions, attaché au nœud) ou bien réalisé par un programme tiers. Trois types de *liens virtuels* existent au sein de *NEmu* :

- Un `VLink` est un *lien virtuel point-à-point* qui inter-connecte deux entités virtuelles.
- Un `VHub` est un *lien virtuel multi-points* émulant un concentrateur Ethernet (*hub*) et pouvant inter-connecter un ensemble d'entités virtuelles.
- Un `VSwitch` est un *lien virtuel multi-points* émulant un commutateur Ethernet (*switch*) et pouvant inter-connecter un ensemble d'entités virtuelles.

Les *liens virtuels* transportent des trames Ethernet d'une interface réseau virtuelle à une ou plusieurs autres. Le trafic Ethernet est transporté entre les interfaces réseaux par des tunnels mis en place par des connexions TCP. Chacune de ces connexions TCP est appelée `VLink`. *NEmu* peut instancier lui même un émulateur vide (i.e., sans système d'exploitation, unité de stockage, etc.) pour jouer le rôle d'un `VHub`. Une telle instance de QEMU coûte 7.2 Mo de mémoire vive. Nous avons également conçu un programme nommé *vswitch*, composé de 2500 lignes de C++, pouvant à la fois émuler un `VLink`, un `VHub` ou bien un `VSwitch`. L'avantage de ce programme réside dans ses paramètres réseaux hautement configurables. Ainsi, il est possible au sein d'un *vswitch* de régler à chaud la bande passante, le délai ainsi que le taux d'erreurs sur n'importe quelle interface et de manière strictement indépendante. Cela permet d'obtenir un réglage plus fin de la topologie réseau. Enfin, un `Slirp` est un lien spécial qui permet à une machine virtuelle d'accéder à Internet. Ce système utilise une technique proche du NAT afin de permettre à une machine virtuelle d'utiliser les cartes réseaux physiques de l'hôte qui l'héberge. Le trafic d'une interface peut être sauvegardé dans un fichier *pcap* en vue d'être analysé par la suite [11].

La Figure 1 montre un exemple de topologie réseau générée par *NEmu*. Sur le coté gauche, deux `VHost` sont inter-connectés à un `VRouter` au travers d'un `VSwitch` en utilisant des tunnels TCP. Sur le coté droit, deux autres `VHost` sont inter-connectés au même `VRouter` au travers d'un `VHub` en utilisant également des tunnels TCP. Ici, l'ensemble des liens virtuels sont gérés par des *vswitch*.

3) Ressources matérielles : La Table I expose l'ensemble des éléments réseaux définis par *NEmu* ainsi que leur besoin en mémoire vive. *Dynamips* [12] est un autre émulateur permettant de manipuler des équipements virtuels CISCO (router, switch, etc.). Comme nous pouvons le constater, les différents éléments de *NEmu* requièrent beaucoup moins de puissance que *Dynamips*.

La Table II expose les débits des différents *liens virtuels* utilisables au sein de *NEmu*. Nous pouvons constater que notre programme *vswitch*, qui reste celui possédant le plus de possibilités, reste très efficace.

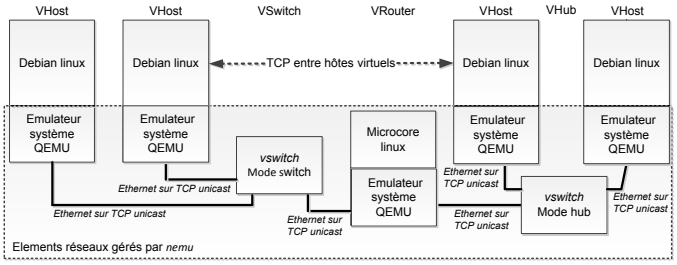

FIGURE 1. Éléments réseaux de *NEmu* en action

TABLE I
RESSOURCE MÉMOIRE NÉCESSAIRE POUR LES ÉLÉMENTS RÉSEAUX

Élément réseau	Système d'émulation	Mémoire nécessaire
`VLink`	Dynamips	9.4 Mo
	vswitch	250 Ko
`VHub`	QEMU hub	7.2 Mo
	vswitch	250 Ko
`VSwitch`	Dynamips	9.4 Mo
	vswitch	250 Ko
`VRouter`	Microcore	100 Mo
	Vyatta	512 Mo
	Cisco IOS	256 Mo
`VHost`	Debian	256 Mo
	Windows XP	512 Mo
	Windows 7	2 Go

TABLE II
DÉBIT DES DIFFÉRENTS LIENS VIRTUELS

Élément réseau	Système d'émulation	Débit effectif
`VLink`	Lien direct	34.0 Mo/s
	vswitch	30.0 Mo/s
`VHub`	*vswitch*	30.0 Mo/s
	QEMU hub	20.0 Mo/s
`VSwitch`	*vswitch*	19.5 Mo/s

C. Gestion du réseau virtuel

Comme expliqué précédemment, une *session* au sein de *NEmu* représente la configuration d'une topologie réseau résidant sur un même hôte physique. Un réseau virtuel distribué sur *n* machines physiques se composera donc au minimum de *n sessions NEmu*. Une *session* est représentée par un système de fichiers auto-généré qui peut être sauvegardé et ré-utilisé sur la même ou sur une différente infrastructure.

Le *manager* est le moyen qui permet à un utilisateur de manipuler une *session*. Les *sessions* sont indépendantes même si elles font partie d'une unique topologie. Cela revient à dire qu'une erreur dans une *session* n'aura pas d'impact sur les autres. Le *manager* peut s'utiliser de trois façons :

- Comme un module python classique à importer dans un code source.

– Comme un interpréteur de commandes dynamique.
– Comme un lanceur de script python.

Le *manager* permet également de manipuler plusieurs *sessions* au travers de la même interface en utilisant des tunnels SSH pour communiquer avec les *sessions* distantes.

Une topologie peut être visualisée à tout moment au travers de la suite logiciel *Graphviz* [13].

III. Application

Afin d'illustrer l'utilisation de *NEmu* au sein des étapes de test et d'évaluation d'une application reposant sur un *overlay*, nous considérons le cas d'une application de distribution de fichiers. Une telle application a souvent besoin de mettre en place un *overlay* recouvrant la totalité des nœuds qui composent ce dit réseau. On parle d'*overlay* lorsque l'ensemble des nœuds du réseau maintiennent des connexions entre eux. Dans notre étude, nous considérons que cet *overlay* forme un arbre et que le fichier est distribué grâce à des mécanismes de routage et de diffusion. Contrairement aux applications de type *pair-à-pair*, où la topologie possède une forte volatilité due à des connexions/déconnexions intempestives, les nœuds qui composent l'*overlay* doivent maintenir les connexions qui composent le réseau virtuel.

A. Application de distribution de fichiers

Le but de notre application est d'envoyer un fichier de taille importante au travers d'un flux continu sur un *overlay* en forme d'arbre, et ainsi de délivrer le contenu de ce fichier à un ensemble de clients placés sur les feuilles de l'arbre. Les paquets sont dupliqués par les nœuds afin de réaliser du *multicast* au niveau applicatif. Les inter-connexions entre les nœuds composant l'*overlay* sont des liens TCP. L'utilisation de TCP est nécessaire afin d'assurer une réception correcte de l'ensemble du flux, contrairement aux applications *multicast* standards qui se placent au dessus d'UDP.

Des travaux existants traitent de cette problématique de distribution *multicast* de fichiers de taille importante. Par exemple, ROMA [14] est basé sur la gestion de tampons, au niveau applicatif, dans les nœuds intermédiaires dans le but de contrôler la congestion. En cas de congestion d'un tampon, celui-ci rejette les nouveaux paquets entrants. Cette solution augmente, dans des proportions importantes, le délai global de réception dans la mesure où le taux de retransmissions explose en cas de saturation des tampons. MCC [15] est une autre technique traitant de la même problématique. Cette solution introduit un mécanisme de contrôle de congestion similaire à celui inclus dans TCP. Cette technique est donc plus adaptée à des réseaux où les inter-connexions reposent sur des protocoles peu sûrs comme UDP.

Notre étude se porte sur l'efficacité de l'utilisation du mécanisme de *back pressure* (ou fenêtrage) interne à TCP dans la distribution de fichiers de taille importante.

Une illustration de notre application et de son *overlay* est fournie par la Figure 2.

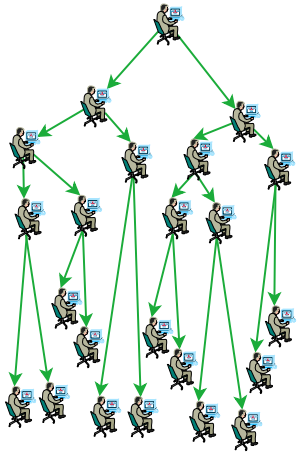

Figure 2. *Overlay* de distribution de fichier

B. Chaînage de connexions TCP

Le but de notre étude est d'obtenir des résultats préliminaires sur l'efficacité de notre *overlay* basé sur un chaînage de connexions TCP. L'idée est de maximiser les débits et de minimiser les délais de réception dans une structure arborescente. Par conséquent, nous avons conduit deux scénarios distincts :

– Nous avons en premier lieu considéré une chaîne simple et directe de connexions TCP en faisant varier le nombre de nœuds composant la chaîne. Le but étant de vérifier si ce nombre de nœuds intermédiaires a un impact sur la réception du client en terme de débit et de délai. Nous avons pour cela émulé cette chaîne dans *NEmu* et mesuré le débit ainsi que le délai résultants. Dans cet étude, la profondeur de l'arbre varie tandis que l'arité des nœuds est fixée à 1 et à 0 pour le nœud final qui représente le client.
– Dans un second temps, nous avons reconduit l'expérience en émulant l'arbre complet dans le but d'inclure la largeur de l'arbre comme paramètre supplémentaire. L'arité des nœuds est donc variable mais homogène.

Un simple programme maintenant l'*overlay* est placé sur chaque nœud et transmet les paquets qu'il reçoit à l'ensemble des ses fils comme illustré dans la Figure 3.

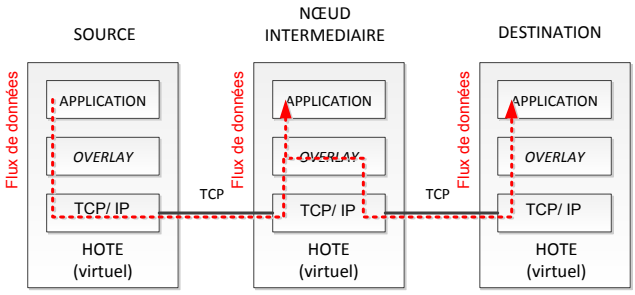

Figure 3. Mécanisme de transmission de l'*overlay*

C. Paramètres de l'expérimentation

Nous avons configuré chaque nœud virtuel comme suit :

- Processeur Intel Core 2 Duo,
- 256 Mo de mémoire vive,
- 1 ou 2 cartes réseaux Realtek rtl8139,
- Système d'exploitation Debian Squeeze 32 bits.

Le réseau virtuel est hébergé sur un serveur possédant 48 cœurs physiques ainsi que 64 Go de mémoire vive. La simulation a été réalisée plusieurs fois avec différentes bandes passantes entre les nœuds : respectivement 1.25, 2.5, 5, 10 et 20 Mbits/s afin de vérifier l'impact de la bande passante initiale entre les nœuds sur les délais et les débits généraux. Chaque connexion entre les nœuds est effectuée par un *vswitch* de manière à régler la bande passante disponible. Le flux à diffuser est un fichier de 1 Go.

D. Résultats pour la chaîne

Pour cette première expérience, nous avons réalisé les simulations avec des chaînes de 2, 4, 8 et 16 nœuds. En tenant compte de la littérature [16]–[18], nous savons que la taille moyenne des paquets avoisine 500 octets ; néanmoins, les récentes augmentations des MTU ont amené les paquets à doubler de taille. C'est pourquoi nous avons réitéré l'ensemble des simulations pour des paquets de 1024 octets de données.

a) Débits: Les résultats des débits sont présentés dans les Figures 4 et 5. L'axe des abscisses représente le nombre de nœuds dans la chaîne, l'axe des ordonnées représente le débit de réception du client placé en fin de chaîne en Kbits/sec.

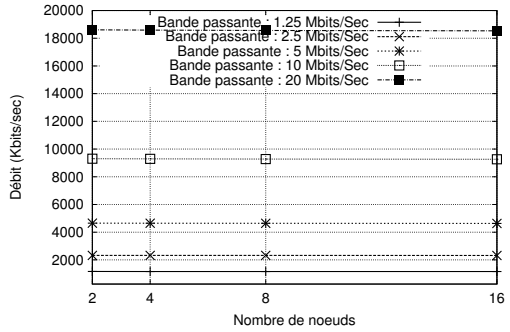

FIGURE 4. Débit de réception du client pour des paquets de 500 octets

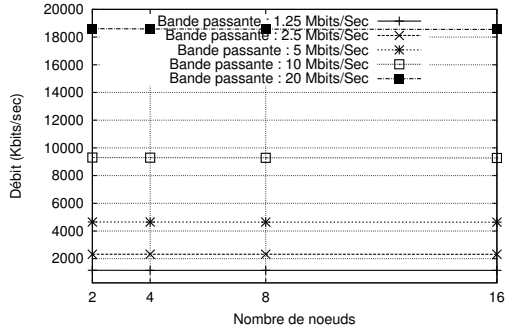

FIGURE 5. Débit de réception du client pour des paquets de 1024 octets

Nous pouvons observer que les débits sont stables et non dégradés par l'augmentation du nombre de nœuds intermédiaires. De plus, nous pouvons affirmer que la taille des paquets n'a aucune influence sur les débits. La faible perte comparée à la bande passante initialement fixée est due à l'approximation de réglage au sein du *vswitch*. Ces résultats assurent que le débit d'un flux n'est pas dégradé même si la chaîne est composée d'un nombre significatif de nœuds intermédiaires. Par conséquent, nous pouvons dire que dans un chaînage de connexions TCP semblable au nôtre :

$$debit_{client} \simeq bande\ passante$$

b) Délais: Les résultats pour les délais sont présentés dans les Figures 6 et 7. L'axe des abscisses représente le nombre de nœuds dans la chaîne, l'axe des ordonnées représente le délai général de réception du client placé en fin de chaîne en ms.

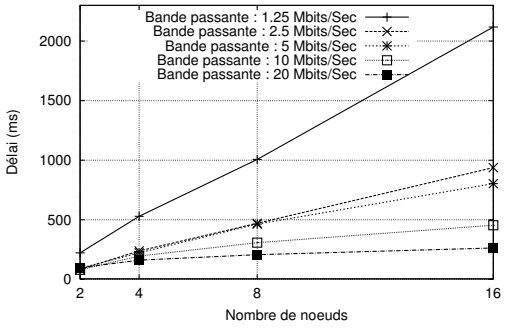

FIGURE 6. Délai de réception du client pour des paquets de 500 octets

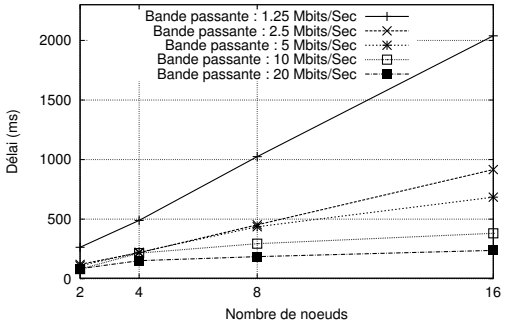

FIGURE 7. Délai de réception du client pour des paquets de 1024 octets

Nous remarquons que le délai augmente linéairement avec la taille de la chaîne. Le phénomène est d'autant plus marqué que la bande passante initiale est faible. La raison réside dans le fait que les faibles bandes passantes génèrent plus de congestion en raison de la faible vitesse de déchargement des tampons. Encore une fois, la taille des paquets a un impact extrêmement négligeable.

Avec l'aide de *NEmu*, nous avons donc observé qu'un chaînage de connexions TCP maintient le débit mais génère des délais peu ou prou importants. Pour une distribution de fichiers, les délais observés sont acceptables. Cependant, le

chaînage de connexions TCP n'est pas adapté aux applications temps réel d'autant plus que la bande passante est faible.

E. Résultats pour l'arbre

Pour la réalisation de l'arbre, nous avons réitéré la simulation avec une profondeur d'arbre variable (1, 2, 4 et 6) ainsi qu'une arité des nœuds variable (2 à 7) mais homogène. Nous avons uniquement réalisé ces simulations avec des paquets de 1024 octets au vue des résultats de la précédente simulation (paragraphe III-D). On mesure ici les débits et les délais de réception des clients placés sur les feuilles de l'arbre.

c) Débits: Les résultats des débits sont présentés dans les Figures 8 et 9. L'axe des abscisses représente respectivement la profondeur et l'arité des nœuds de l'arbre, l'axe des ordonnées représente le débit de réception des clients placés sur les feuilles de l'arbre en Kbits/sec.

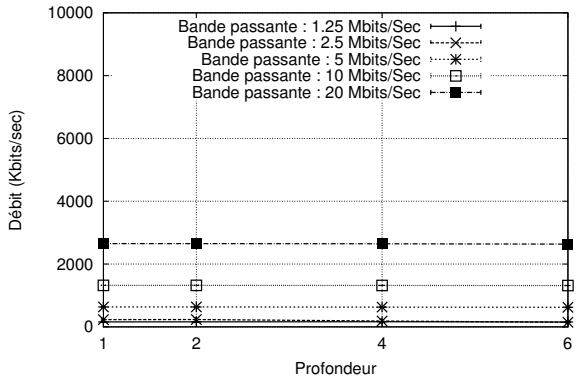

FIGURE 8. Débit de réception *vs* profondeur de l'arbre pour une arité de 7

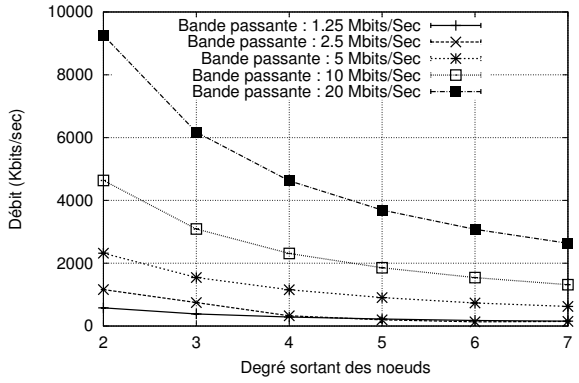

FIGURE 9. Débit de réception *vs* arité de l'arbre pour une profondeur de 6

Nous constatons premièrement que le débit n'est pas dégradé est reste stable en augmentant la profondeur de l'arbre même si l'arité des nœuds est importante. On constate également que le débit décroît logarithmiquement quand l'arité augmente. On peut donc noter que :

$$debit_{client} \simeq \frac{bande\ passante}{arite\ de\ l'arbre}$$

d) Délais: Les résultats pour les délais sont présentés dans les Figures 10 et 11. L'axe des abscisses représente respectivement la profondeur et l'arité des nœuds de l'arbre, l'axe des ordonnées représente le délai de réception des clients placés sur les feuilles de l'arbre en ms.

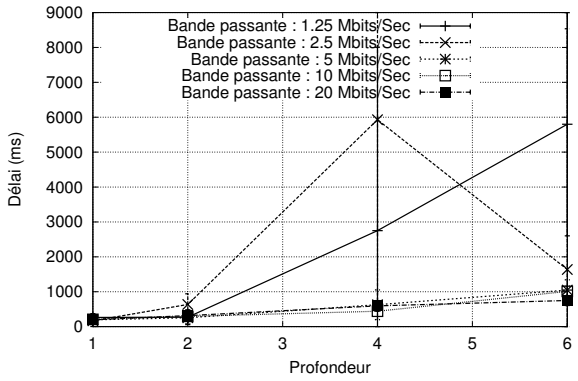

FIGURE 10. Délai de réception *vs* profondeur de l'arbre pour une arité de 7

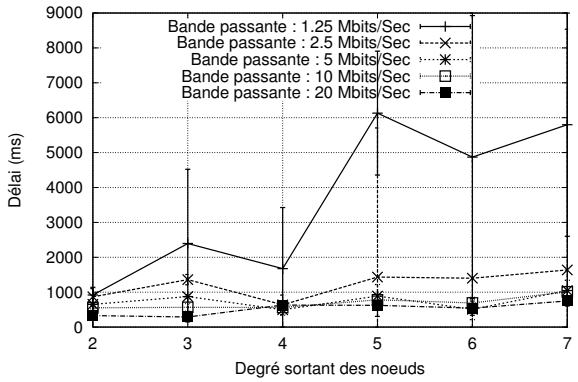

FIGURE 11. Délai de réception *vs* arité de l'arbre pour une profondeur de 6

Nous constatons que les délais sont plus instables, surtout pour une faible bande passante initiale. En effet, plusieurs reconductions de l'expérience fournissent des résultats différents. Le phénomène est fortement visible grâce aux écarts-types présents sur les Figures 10 et 11. Ces latences aléatoires peuvent refléter des artefacts du système d'exploitation des machines virtuelles (i.e., appels systèmes, changements de contexte, etc.) et nécessiteraient des recherches plus approfondies.

Avec l'aide de *NEmu*, nous avons observé qu'un arbre de diffusion reposant sur un *overlay* maintient un débit stable mais ne permet pas d'obtenir de prédictions sur les délais.

IV. ÉTAT DE L'ART

A. Émulation de nœuds

Actuellement, *NEmu* utilise des instances de QEMU pour ses nœuds virtuels. En dépit du nombre de solutions de virtualisations existantes, nous avons choisi QEMU qui est un puissant émulateur générique [4]. QEMU ne requiert aucun droit particulier sur l'hôte physique qui l'héberge et permet d'émuler d'anciens périphériques [19]. D'autres systèmes tels

que *VMware* [20] ou *Xen* [21] possèdent de meilleures performances au niveau des entrées/sorties mais ne supportent que des architectures x86/x86_64 [22] [23], ce qui compromet le *support hérité* propre à un NVE défini en Section II. De plus, *Xen* est proche du noyau, ce qui l'oblige a demander des droits supérieurs sur le système physique pour fonctionner. Les systèmes tels que *VirtualBox* [24] et *VirtualPC* [25] ne supportent également que des architectures x86/x86_64. *UML* [26] ne peut émuler qu'un système d'exploitation de type Linux. De plus, le projet n'est plus supporté depuis des années. *OpenVZ* [27] est désigné comme le successeur d'UML mais possède les mêmes limitations quant au système d'exploitation.

QEMU nous permet donc une configuration matérielle et logicielle accrue répondant parfaitement aux critères d'un NVE (paragraphe II-A).

B. Environnement réseau virtuel

Dynagen [28] est à Dynamips [12], ce que *NEmu* est à QEMU. C'est à dire un système de gestion des machines virtuelles. En revanche, *NEmu* propose des fonctionnalités bien plus avancées que *Dynagen* comme le dynamisme de la topologie, l'ajout de services réseaux ou encore l'aspect communautaire.

GNS [29] est un système ouvert qui permet de construire des réseaux virtuels à l'aide de Dynamips et de QEMU. Néanmoins, ce système n'est pas communautaire. De plus les services sont bornés à ceux prévus dans les systèmes d'exploitations CISCO. Enfin, il n'est utilisable qu'au travers d'une interface graphique rendant complexe la gestion de réseaux virtuels de taille importante.

Velnet [30] est un environnement virtuel dédié à l'enseignement qui repose sur des machines virtuelles VMware. La topologie ne peut être hébergée que sur un seul hôte physique engendrant d'importantes limitations quant à la taille du réseau.

ModelNet [31] émule un réseau virtuel distribué mais qui reste statique durant l'exécution. Ainsi, la *dynamicité* présente dans *NEmu* est impossible avec *ModelNet*. De plus, la gestion du réseau est centralisée empêchant tout aspect communautaire du réseau émulé.

Vagrant [32] utilise des machines VirtualBox dans le but d'émuler un réseau virtuel. La topologie est hébergée sur une seule machine physique et reste statique à l'exécution. Enfin les inter-connexions entre machines virtuelles sont assurées par l'hôte créant un *flat network*, c'est à dire un réseau qui ne repose pas sur les standards en terme d'adressage et de routage des trames réseaux.

VINI [33] est un réseau virtuel distribué qui repose sur la plate-forme *PlanetLab* [34] qui est un réseau mondial de *clusters* distribués. *VINI* utilise des UML limitant fortement les possibilités de configuration matérielle et logicielle. De plus, les connexions sont réalisées au travers d'interfaces réseaux virtuelles sur l'hôte, ce qui rend impossible l'utilisation de cette solution sans droits supérieurs sur l'infrastructure physique.

Violin [35] est semblable à *VINI* mais propose des services réseaux comme *NEmu*. En revanche, ce système possède les mêmes limitations que *VINI*.

NetKit [36] repose également sur UML. Il utilise des *VDE switch*, qui repose sur le mécanisme de mémoire partagée, afin d'inter-connecter les machines virtuelles. Un tel système ne peut pas être distribué.

Marionnet [37] est un environnement virtuel dédié à l'enseignement. Il fournit plusieurs services réseaux mais n'est pas distribué et repose sur UML.

Virconel [38] utilise des machines virtuelles OpenVZ qui limitent également les possibilités de paramétrage des nœuds. La topologie reste statique à l'exécution et les inter-connexions sont faites dans le noyau de la machine hôte, nécessitant des droits supérieurs.

Vnet [39] est un système distribué qui inter-connecte des machines virtuelles hébergées sur différents hôtes physiques au travers du réseau IP réel. Même si ce système permet de créer un réseau virtuel distribué, seules les inter-connexions sont fournies. Par conséquent, ce système nécessite une configuration accrue et complexe.

Les plate-formes de recherches telles que *PlanetLab* [34], *GENI* [40] ou encore *FEDERICA* [41] fournissent des infrastructures virtuelles composées de *bancs* matériels propres. L'utilisateur doit donc posséder un compte sur ces infrastructures et utiliser des outils, services et API intrinsèques à ces plate-formes.

Le tableau III résume les différentes propriétés des précédentes solutions comparées à celles de *NEmu*. Nous constatons que *NEmu* couvre la totalité des usages (test, évaluation des performances, enseignement). Il peut donc être un outil de recherche puissant de par sa dynamicité et sa distributivité. De plus, il peut être déployé sans nécessiter de droits particuliers sur l'infrastructure physique. Son utilisation est facilitée grâce à ses nombreux services réseaux disponibles et à la possibilité d'en ajouter d'autres. Enfin, l'aspect communautaire de *NEmu* permet à plusieurs utilisateurs d'assembler leurs sous-réseaux afin de construire un réseau virtuel plus large.

TABLE III
COMPARAISON DES ENVIRONNEMENTS RÉSEAUX VIRTUELS

	Test	Évaluation	Enseignement	Dynamique	Distribué	Communautaire	Services réseaux	Droits administrateur
Dynagen	●	○	●	○	●	○	●	○
GNS3	●	○	●	○	●	○	●	○
Velnet	○	○	●	○	○	○	●	○
ModelNet	●	●	○	○	●	○	○	○
Vagrant	●	○	●	○	○	○	●	○
VINI	●	●	○	●	●	○	○	●
Violin	●	●	○	●	●	○	●	○
NetKit	●	○	●	●	○	○	●	○
Marionnet	●	○	●	●	○	○	●	○
Virconel	●	●	●	○	●	○	●	●
NEmu	●	●	●	●	●	●	●	○

● *Oui* ○ *Non*

V. Conclusion

NEmu est un outil pour la création et la gestion de réseaux virtuels dynamiques et hétérogènes. Ces réseaux virtuels peuvent être distribués sur plusieurs hôtes physiques et administrés par plusieurs utilisateurs sans droit particulier sur l'infrastructure réelle. Nous avons montré que *NEmu* peut être utilisé pour émuler, tester et évaluer des *overlays* applicatifs. Nos expériences ont permis de mettre en évidence des résultats sur la distribution de fichiers de taille importante reposant sur un *overlay* ou les nœuds sont inter-connectés par des connexions TCP. Nous en concluons que les *overlays* en forme d'arbre sont efficaces en terme de débits mais beaucoup moins en terme de délais comme cela avait été observé dans des travaux antérieurs réalisés par simulation. Ces résultats montrent que *NEmu* peut être utilisé pour tester et évaluer avec efficacité des applications réseaux, pouvant donc remplacer les plate-formes physiques standards tout en faisant mieux que des simulateurs réseaux en termes de réalisme. Plusieurs étapes sont déjà prévues dans le développement futur de *NEmu* :

- L'intégration de capacités de migration pour les machines virtuelles afin d'intégrer l'équilibrage de charge,
- L'intégration de nouveaux services pour les `VRouter`,
- L'implémentation d'une interface graphique,
- L'implémentation d'équipements sans fil (i.e., cartes réseaux sans fil, stations de base, points d'accès) afin d'émuler des réseaux mobiles.

Références

[1] T. Henderson, M. Lacage, G. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, 2008.

[2] A. Varga *et al.*, "The omnet++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM'2001)*, vol. 9, 2001.

[3] B. Yamini and D. Selvi, "Cloud virtualization : A potential way to reduce global warming," in *RSTSCC*, nov. 2010, pp. 55–57.

[4] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. of the USENIX Annual Technical Conference, FREENIX Track*, 2005, pp. 41–46.

[5] N. Chowdhury and R. Boutaba, "Network virtualization : state of the art and research challenges," *Communications Magazine, IEEE*, vol. 47, no. 7, pp. 20–26, 2009.

[6] P. Mell and T. Grance, "The nist definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, 2009.

[7] KVM, *Virtio*, http://www.linux-kvm.org/page/Virtio.

[8] NBD, *Network Block Device*, http://nbd.sourceforge.net.

[9] B. Hubert, G. Maxwell, R. Van Mook, M. Van Oosterhout, P. Schroeder, and J. Spaans, "Linux advanced routing & traffic control," in *Ottawa Linux Symposium*, 2003, pp. 213–222.

[10] R. Shingledecker, *TinyCore Linux*, http://distro.ibiblio.org/tinycorelinux.

[11] Tcpdump, *LibPcap*, http://www.tcpdump.org.

[12] ipflow, *Dynamips*, http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator.

[13] Graphviz, *Graph Visualization Software*, http://www.graphviz.org.

[14] G. Kwon, J. Byers *et al.*, "Roma : Reliable overlay multicast with loosely coupled tcp connections," in *Proc. of the 23th IEEE INFOCOM*, 2004.

[15] G. Urvoy-Keller and E. Biersack, "A congestion control model for multicast overlay networks and its performance," in *Proc. of the 4th International Workshop on NGC*, 2002.

[16] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," *ACM PAM*, pp. 205–214, 2004.

[17] P. Borgnat, G. Dewaele, K. Fukuda, P. Abry, and K. Cho, "Seven Years and One Day : Sketching the Evolution of Internet Traffic," in *Proc. of the 28th IEEE INFOCOM 2009*, April 2009, pp. 711–719.

[18] G. Dewaele, Y. Himura, P. Borgnat, K. Fukuda, P. Abry, O. Michel, J.J., R. Fontugne, K. Cho, and H. Esaki, "Unsupervised host behavior classification from connection patterns," *International Journal of Network Management*, vol. 20, pp. 317–337, 2010.

[19] A. Ribiere, "Emulation of obsolete hardware in open source virtualization software," in *8th IEEE INDIN*, 2010, pp. 354–360.

[20] VMware, *Virtualization Solutions*, http://www.vmware.com.

[21] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM SOPS*, 2003, pp. 164–177.

[22] P. Domingues, F. Araujo, and L. Silva, "Evaluating the performance and intrusiveness of virtual machines for desktop grid computing," in *Parallel Distributed Processing, IEEE IPDPS*, may 2009, pp. 1–8.

[23] J. Che, Y. Yu, C. Shi, and W. Lin, "A synthetical performance evaluation of openvz, xen and kvm," in *IEEE APSCC*, december 2010, pp. 587 – 594.

[24] Oracle, *VirtualBox*, https://www.virtualbox.org.

[25] Microsoft, *VirtualPC*, http://www.microsoft.com/windows/virtual-pc.

[26] J. Dike, "A user-mode port of the Linux kernel," in *Proceedings of the 2000 Linux Showcase and Conference*, vol. 2, no. 4, 2000.

[27] OpenVZ, *Linux Containers*, http://wiki.openvz.org.

[28] G. Anuzelli, *Dynagen*, http://dynagen.org.

[29] GNS3, *Graphical Network Simulator*, http://www.gns3.net.

[30] B. Kneale, A. Y. De Horta, and I. Box, "Velnet : virtual environment for learning networking," in *Proc. of the 6th Australasian Conference on Computing Education*, vol. 30, 2004, pp. 161–168.

[31] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker, "Scalability and accuracy in a large-scale network emulator," in *The 5th OSDI*, 2002, pp. 271–284.

[32] M. Hashimoto and J. Bender, *Vagrant*, http://vagrantup.com.

[33] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In vini veritas : realistic and controlled network experimentation," in *Proceedings of SIGCOMM*, 2006, pp. 3–14.

[34] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating system support for planetary-scale network services," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, 2004, p. 19.

[35] X. Jiang and D. Xu, "Violin : Virtual internetworking on overlay infrastructure," in *In proc. of the 2nd Intl. Symp. on Parallel and Distributed Processing and applications*, 2003, pp. 937–946.

[36] M. Pizzonia and M. Rimondini, "Netkit : easy emulation of complex networks on inexpensive hardware," in *Proceedings of the 4th TridentCom*, 2008, pp. 7 :1–7 :10.

[37] U. P. 13, *Marionnet*, http://www.marionnet.org.

[38] Y. Benchaib and A. Hecker, "Virconel : A network virtualizer," in *19th IEEE MASCOTS*, July 2011, pp. 429–432.

[39] A. I. Sundararaj, A. Gupta, and P. A. Dinda, "Dynamic topology adaptation of virtual networks of virtual machines," in *Proc. of the 7th LCR Workshop*, 2004, pp. 1–8.

[40] N. Van Vorst, M. Erazo, and J. Liu, "Primogeni : Integrating real-time network simulation and emulation in geni," in *PADS, IEEE Workshop on*, june 2011, pp. 1–9.

[41] G. E. project, *FEDERICA*, http://www.fp7-federica.eu.