Virtualisation distribuée de réseaux dynamiques et mobiles avec NEmu

Vincent Autefage, Damien Magoni

Virtualisation distribuée de réseaux dynamiques et mobiles avec *NEmu*

Vincent Autefage* et Damien Magoni*

*Univ. Bordeaux, LaBRI
UMR 5800, F-33400, Talence, France
autefage,magoni@labri.fr
http://nemu.valab.net

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 suffisamment puissante, contrôlée et dynamique de réseau fixe ou mobile afin de tester et valider l'application. La virtualisation est par conséquent une technique fiable et économe pour jouer le rôle de plate-forme de test. Nous proposons un outil appelé *NEmu* ayant pour but de générer des réseaux virtuels statiques, dynamiques ou mobiles à la demande afin de tester et de valider des prototypes d'applications réseaux avec un contrôle complet de la topologie, de la mobilité des nœuds ainsi que 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 arbre 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 l'arbre, de la taille des paquets et de la bande passante globale des liens.

1 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* (Henderson et al., 2008) ou encore *OMNeT++* (Varga et al., 2001), 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 sur des implémentations diffère dans la mesure où elle opère sur des applications natives et s'intéresse davantage au temps d'exécution, à l'utilisation du processeur et de la mémoire, au trafic réseau généré ainsi qu'aux différentes réactions du système d'exploitation.

Il est difficile d'effectuer cette tâche quand l'application repose sur une infrastructure impliquant plusieurs dizaines de machines. De plus, la mobilité et le dynamisme de la topologie requis par certaines expérimentations, augmente la complexité de mise en œuvre de cette tâche.

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 dynamicité ou encore la mobilité de la topologie d'une telle plate-forme est fortement restreinte et complexe à réaliser. 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 (Yamini et Selvi, 2010).

Notre solution permet donc de dépasser les contraintes matérielles et ainsi de fournir une infrastructure suffisante permettant de créer des réseaux virtuels dynamiques et mobiles. 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. La topologie peut évoluer au cours du temps que ce soit manuellement, i.e. par l'action directe d'un utilisateur, ou bien automatiquement en suivant un scénario de mobilité préalablement défini.

Nous proposons un outil permettant de générer des réseaux virtuels afin de tester et de valider des prototypes d'applications pour réseaux statiques, dynamiques ou mobiles avec un contrôle accru sur les propriétés de la topologie, sur la mobilité ainsi que sur la configuration 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 (Bellard, 2005). Le nom *NEmu*, pour ***N****etwork* ***E****mulator for* ***m****obile* ***u****niverses*, 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 de plusieurs points. Dans un premier temps nous présentons en Section 2 une description détaillée de notre outil *NEmu* permettant de créer et de gérer des réseaux statiques, dynamiques ou mobiles de nœuds et de liens virtuels dans le but d'émuler une topologie réseau arbitraire et évolutive. Dans le Section 3, nous décrivons notre outil *nemo* implémentant la mobilité au sein de *NEmu* et permettant de faire évoluer un réseau virtuel en temps réel suivant un modèle de mobilité pré-établi. Nous proposons en Section 4 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. Enfin, nous établissons en Section 5 un état de l'art sur les travaux similaires dans la virtualisation de réseaux, ainsi qu'une comparaison détaillée des fonctionnalités et usages principaux proposés par *NEmu* avec ceux des autres travaux existants.

2 L'émulateur réseau

2.1 Description

NEmu est un programme d'environ 6000 lignes, écrit en python, permettant de créer un réseau distribué de machines virtuelles. Il est basé sur un concept appelé *Network Virtualization Environment* (NVE) (Chowdhury et Boutaba, 2009). 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 et hétérogène constituée de nœuds et de liens virtuels 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, et également entre les VNs et l'infrastructure elle même.
- 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, i.e. *virtual IaaS* (Mell et Grance, 2009), 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.

2.2 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*.

2.2.1 Les nœuds virtuels

Un *nœud virtuel* dans *NEmu* (appelé `VNode`) 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* (KVM) 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). 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 (e.g. DHCP, DNS, NFS, HTTP, SSH, NTP , etc.), des protocoles de routage dynamique (e.g. RIP, OSPF), un pare-feu Netfilter (NetFilter, 2000) ainsi qu'une couche QoS avec *Traffic Control* (Hubert et al., 2003). 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 *TinyCore* qui est une distribution Linux extrêmement légère et hautement configurable (Shingledecker, 2008). Un `VRouter` requiert environ $\sim$100 Mo en mémoire vive en activant la totalité de ses services. Chaque service peut être activé ou désactivé que le `VRouter` soit actif ou inactif.

2.2.2 Les liens virtuels

Un *lien virtuel* dans *NEmu* (appelé `VLink`) 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 `VLine` 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 ou UDP. Nous avons conçu un programme nommé *vswitch* (Magoni, 2012b), composé de 2500 lignes de C++, pouvant à la fois émuler un `VLine`, 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, ce qui nous 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 (TcpDump, 1987).

La Figure 1 représente 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*.

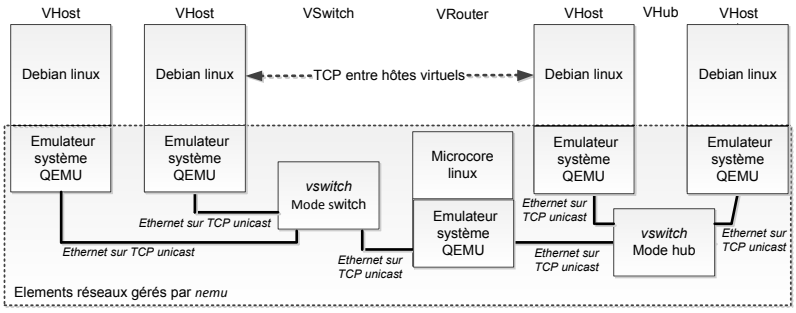

FIG. 1 – *Éléments réseaux de NEmu en action.*

2.2.3 Ressources matérielles

Le Tableau 1 expose l'ensemble des éléments réseaux définis par *NEmu* ainsi que leur besoin en mémoire vive. *Dynamips* (Fillot, 2007) 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*.

2.3 Gestion du réseau virtuel

Une *session* au sein de *NEmu* représente la configuration d'une topologie réseau résidant sur un même hôte physique et appartenant à un unique utilisateur. Un réseau virtuel distribué sur *n* machines physiques se composera donc au minimum de *n sessions*. De la même manière,

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

TAB. 1 – *Ressource mémoire nécessaire pour les éléments réseaux.*

un réseau virtuel fournit par *n* utilisateurs distincts se composera au minimum de *n sessions*. Une *session* est représentée par un système de fichiers auto-généré qui peut être sauvegardé et ré-utilisé sur la même ou sur une infrastructure différente.

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 scripts 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* (Graphviz, 1988).

3 Le mobilisateur réseau

3.1 Présentation

NEmu est un émulateur de réseaux virtuels pouvant être mobiles. En d'autres termes, il est possible de créer un réseau virtuel dont la topologie est évolutive dans le temps de manière autonome. *nemo* (Magoni, 2012a) est un programme léger (944 Ko en RAM) implémenté en C++, d'environ 3000 lignes de codes, permettant de générer des scénarios de connectivité pour les réseaux mobiles. Un scénario de connectivité est une suite temporelle d'événements de connexion et de déconnexion entre des noeuds mobiles représentant des liens sans fil. *nemo* est capable d'envoyer des ordres à *NEmu* en temps réel ce qui permet d'émuler les changements de connectivité entre noeuds mobiles en créant, détruisant ou changeant les caractéristiques des liens aux moments voulus. *nemo* se compose de deux parties : une partie basée sur un ordonnanceur en temps simulé et une autre basée sur un ordonnanceur en temps réel.

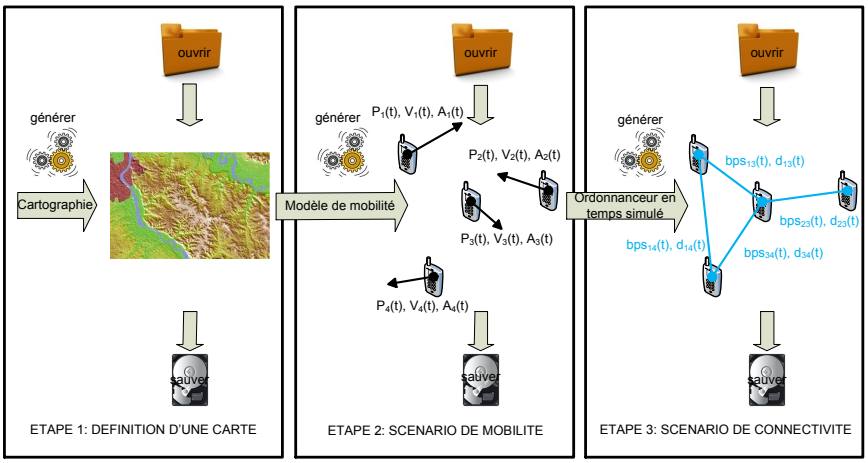

FIG. 2 – *Génération de scénarios de connectivité avec nemo.*

3.2 Ordonnanceur en temps simulé

L'ordonnanceur en temps simulé est le cœur de la partie simulation de *nemo*. Il permet de générer des scénarios de connectivité pour l'ordonnanceur en temps réel. Trois étapes sont nécessaires pour générer un scénario de connectivité :

- générer une carte ;
- générer un scénario de mobilité sur cette carte ;
- générer un scénario de connectivité à partir du scénario de mobilité.

A chaque étape, les résultats de l'étape peuvent être sauvegardés puis chargés ultérieurement afin d'éviter les re-calculs. L'ordonnanceur en temps simulé déroule le scénario de mobilité et à chaque intervalle de temps (fixé par l'utilisateur), il calcule les distances et les éventuelles connexions sans fil possibles entre toutes les paires de noeuds mobiles. Son fonctionnement est illustré à la Figure 2. Pour le moment *nemo* génère des cartes rectangulaires et des scénarios de mobilité aléatoires. À l'avenir, *nemo* devra être capable de charger des cartes 3D contenant des informations d'atténuation et des scénarios de mobilité provenant de générateurs existants réalistes tels que le *Gauss-Markov Mobility Model* (Liang et Haas, 1999) ou encore le *Reference Point Group Mobility Model* (Hong et al., 1999).

L'ordonnanceur en temps simulé souffre de quelques limitations :

- Il peut nécessiter des calculs intensifs.
- Il y a un compromis à faire entre la précision temporelle (intervalle de temps entre chaque évaluation de la connectivité) la durée totale des calculs et le nombre d'évènements détectés.

3.3 Ordonnanceur en temps réel

L'ordonnanceur en temps réel est le cœur de la partie émulation de *nemo*. Il exécute des évènements de connectivité à leurs intervalles de temps exacts fixés par rapport au début du scénario. La précision temporelle utilisée dans l'ordonnanceur temps réel est égale à celle fixée lors du traitement du scénario de mobilité par l'ordonnanceur en temps simulé. Son fonctionnement est illustré à la Figure 3. Celle-ci montre que *NEmu* joue le rôle de contrôleur central vis à vis des autres processus. Dans un réseau virtuel mobile, les *vswitch* font office de cartes réseaux sans fil pour les nœuds virtuels, c'est donc au sein des *vswitch* que les liens réels (i.e. tunnels TCP ou UDP) sont instanciés. *NEmu* récupère les événements de connectivité générés par *nemo* et les retransmet aux différents *vswitch* correspondants aux cartes réseaux des différents nœuds virtuels afin de faire évoluer la topologie du réseau. L'ordonnanceur temps réel peut être mis en pause puis remis en marche à tout moment par l'utilisateur.

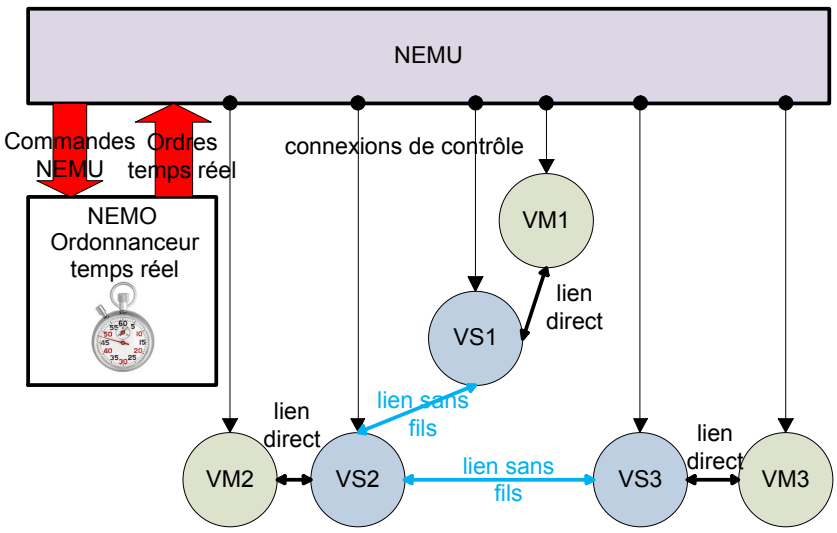

FIG. 3 – *Emulation de réseaux mobiles virtuels avec nemo.*

L'ordonnanceur temps réel souffre de quelques limitations :

- Contrairement à *NEmu*, *nemo* est centralisé.
- Il est préférable d'exécuter toutes les sessions sur un unique hôte physique pour ne pas impacter les performances de l'émulation.
- Les *sockets* utilisées pour inter-connecter les *vswitch* introduisent des délais et réduisent la précision temporelle significative. Cette dernière sera au mieux de l'ordre de la milliseconde.

4 Application

Afin d'illustrer l'utilisation de *NEmu* au sein des étapes de test et d'évaluation d'une application réseau, 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.

4.1 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 (Kwon et al., 2004) 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 (Urvoy-Keller et Biersack, 2002) 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 4.

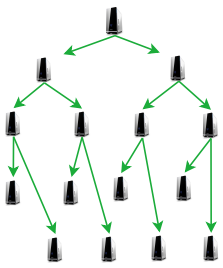

FIG. 4 – *Overlay applicatif.*

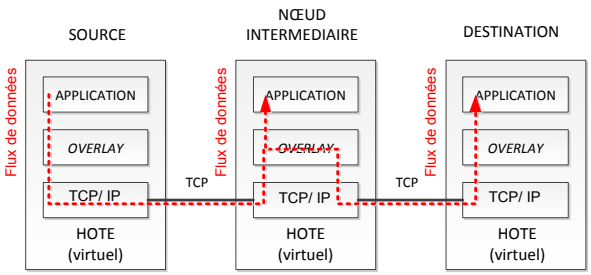

FIG. 5 – *Mécanisme de transmission de l'overlay.*

4.2 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 5.

4.3 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.

4.4 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 (McGregor et al., 2004; Borgnat et al., 2009; Dewaele et al., 2010), 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.

Débits Les résultats des débits 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 débit de réception du client placé en fin de chaîne en Kbits/sec.

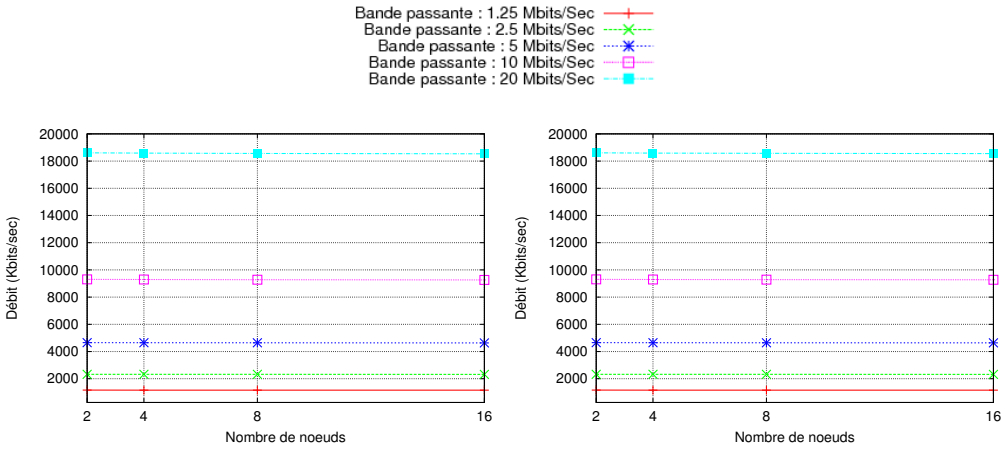

FIG. 6 – *Débit de réception du client pour des paquets de 500 octets.*

FIG. 7 – *Débit de réception du client pour des paquets de 1024 octets.*

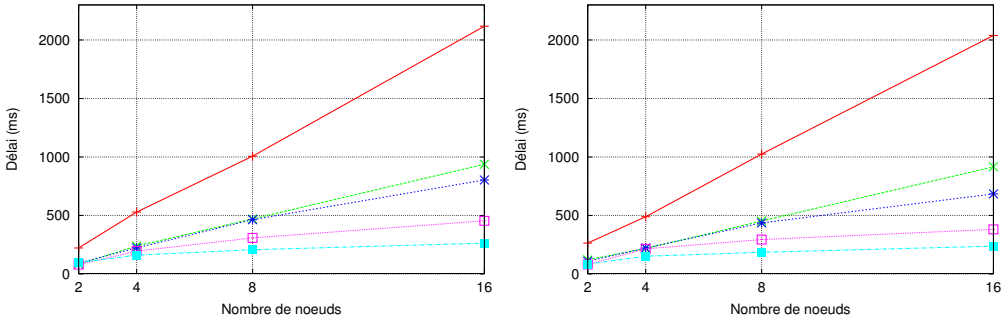

FIG. 8 – *Délai de réception du client pour des paquets de 500 octets.*

FIG. 9 – *Délai 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 :

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

Délais Les résultats pour les délais sont présentés dans les Figures 8 et 9. 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.

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.

4.5 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 4.4). On mesure ici les débits et les délais de réception des clients placés sur les feuilles de l'arbre.

Débits Les résultats des débits 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 débit de réception des clients placés sur les feuilles de l'arbre en Kbits/sec.

Nous constatons premièrement que le débit n'est pas dégradé et 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 :

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

Délais Les résultats pour les délais sont présentés dans les Figures 12 et 13. 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.

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 12 et 13. 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.

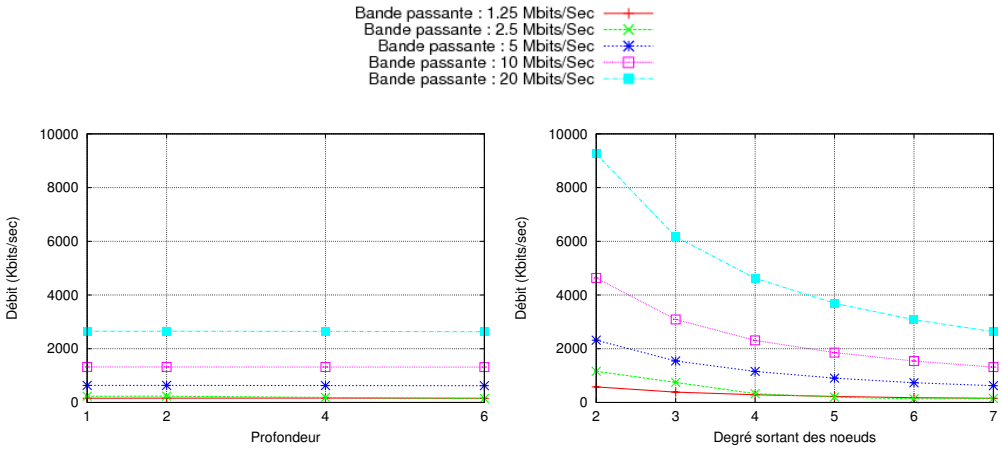

FIG. 10 – *Débit de réception* vs *profondeur de l'arbre pour une arité de 7.*

FIG. 11 – *Débit de réception* vs *arité de l'arbre pour une profondeur de 6.*

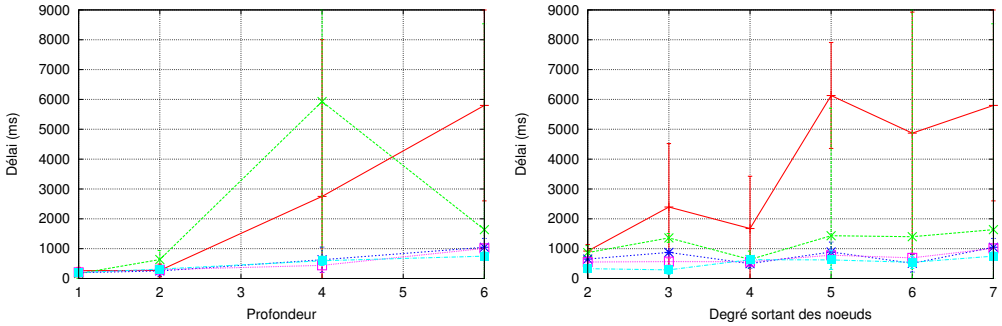

FIG. 12 – *Délai de réception* vs *profondeur de l'arbre pour une arité de 7.*

FIG. 13 – *Délai de réception* vs *arité de l'arbre pour une profondeur de 6.*

5 État de l'art

5.1 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 (Bellard, 2005). QEMU ne requiert aucun droit particulier sur l'hôte physique qui l'héberge et permet d'émuler un grand nombre de périphériques et composants actuels et plus anciens (Ribiere, 2010).

D'autres systèmes tels que *VMware* (EMC, 2004) ou *Xen* (Barham et al., 2003) possèdent de meilleures performances au niveau des entrées/sorties mais ne supportent que des architectures x86/x86_64 (Domingues et al., 2009; Che et al., 2010), ce qui compromet le *support*

hérité propre à un NVE défini en Section 2. 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* (Oracle, 2007) et *Hyper-V* (Microsoft, 2008) ne supportent également que des architectures x86/x86_64 et sont propriétaires.

UML (Dike, 2000) ne peut émuler qu'un système d'exploitation de type Linux. De plus, le projet n'est plus supporté depuis des années.

OpenVZ (Parallels, 2006) est désigné comme le successeur d'UML mais possède les mêmes limitations quant au système d'exploitation.

LXC (Lezcano, 2008) est un système du noyau Linux permettant d'encapsuler un groupe de processus ainsi qu'une partie d'un système de fichier dans un conteneur virtuel. Cette solution n'est donc pas à proprement parler de la virtualisation en ne permet aucune configuration matérielle.

QEMU nous permet donc une configuration matérielle et logicielle accrue répondant parfaitement aux critères d'un NVE.

5.2 Émulation de liens

5.2.1 Switchs virtuels

NEmu utilise un programme nommé *vswitch* pour émuler les liens, ainsi que leurs caractéristiques, entre machines virtuelles. Il existe néanmoins d'autres outils permettant d'interconnecter des machines virtuelles.

VDE (Davoli, 2005) est un switch virtuel effectuant les inter-connexions au travers des mécanismes de mémoire partagée du noyau Linux. Un tel système ne peut être distribué. De plus, VDE n'inclut pas de mécanisme pour modifier les propriétés des liens comme par exemple la bande passante.

Open vSwitch (Pfaff et al., 2009) est un projet libre de switch virtuel permettant un réglage fin des propriétés des inter-connections. Néanmoins, il utilise les interfaces réseaux virtuels du noyau Linux qui ne peuvent être crées et configurés que par un administrateur. L'accessibilité serait donc restreinte avec cet outil.

Vnet (Sundararaj et al., 2004) 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, il n'inclut pas de gestion des propriétés des liens.

5.2.2 Gestionnaire de propriétés des liens

Nous avons implémenté au sein de notre programme *vswitch* la gestion des propriétés des liens comme la bande passante, le délai ou encore le taux d'erreur par bit.

D'autres outils permettent d'effectuer cette tâche comme par exemple *NetEm* (Hemminger et al., 2005). Ce système repose sur un module de QoS du noyau Linux appelé *Traffic Control* (Hubert et al., 2003). Il nécessite des droits d'administrateur sur l'infrastructure. Un tel système est donc peu flexible.

Trickle (Eriksen, 2005) est un outil permettant de gérer la bande passante d'un processus. Notre switch virtuel pouvant gérer plusieurs connexions avec des propriétés différentes, il est beaucoup plus complet que *Trickle*.

5.3 Environnement réseau virtuel

Dynagen (Anuzelli, 2006) est à Dynamips (Fillot, 2007), 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 (GNS3, 2007) 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 (Kneale et al., 2004) 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 (Vahdat et al., 2002) 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 (Hashimoto et Bender, 2010) 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 (Bavier et al., 2006) est un réseau virtuel distribué qui repose sur la plate-forme *PlanetLab* (Bavier et al., 2004) 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 (Jiang et Xu, 2003) est semblable à *VINI* mais propose des services réseaux comme *NEmu*. En revanche, ce système possède les mêmes limitations que *VINI*.

NetKit (Pizzonia et Rimondini, 2008) repose également sur UML et utilise des VDE. Un tel système ne peut pas être distribué.

Marionnet (Lodo et Saiu, 2007) est un environnement virtuel dédié à l'enseignement. Il fournit plusieurs services réseaux mais n'est pas distribué et repose sur UML.

Virconel (Benchaib et Hecker, 2011) 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.

VNUML (DIT, 2003) est un système statique abandonné reposant sur UML et nécessitant des droits administrateurs pour certaines actions.

VNX (DIT, 2008) est le successeur de VNUML. Il permet la gestion de machines virtuelles autres que UML. En revanche et comme son prédécesseur, il nécessite des droits administrateurs et n'inclut aucune gestion des propriétés des liens.

Cloonix (Perrier, 2007) est un outil permettant de gérer des réseaux virtuels dynamiques. En revanche, il ne gère pas les propriétés des liens.

CORE (Ahrenholz et al., 2008) est un outil graphique émulant des réseaux mobiles. Cet outil est basé sur un framework appelé IMUNES (Puljiz et Mikuc, 2003) qui est en fait une sur-couche au système d'exploitation FreeBSD (FreeBSD, 1993). Les nœuds et liens virtuels sont donc directement gérés dans le noyau du système d'exploitation ce qui rend cette solution extrêmement limité en terme de flexibilités.

MobiEmu est un outil permettant de simuler des réseaux mobiles. La gestion de la mobilité y est déléguée à ns3 et la partie virtualisation est effectuée par des conteneurs LXC. Cet outil n'est donc pas conforme aux pré-requis d'un NVE.

Les plate-formes de recherches telles que *PlanetLab* (Bavier et al., 2004), *GENI* (Van Vorst et al., 2011) ou encore *FEDERICA* (GARR, 2008) 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 5.3 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, sa mobilité 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.

	Test	Évaluation	Enseignement	Dynamique	Distribué	Mobile	Communautaire	Services réseaux	Droits standards
Dynagen	●	○	●	○	●	○	○	●	●
GNS3	●	○	●	○	●	○	○	●	●
Velnet	○	○	●	○	○	○	○	●	●
ModelNet	●	●	○	○	●	○	○	○	●
Vagrant	●	○	●	○	○	○	○	●	●
VINI	●	●	○	●	●	○	○	○	○
Violin	●	●	○	●	●	○	○	●	●
NetKit	●	○	●	●	○	○	○	●	●
Marionnet	●	○	●	●	○	○	●	●	●
Virconel	●	●	●	○	●	○	○	●	○
VNUML	●	○	●	○	○	○	○	○	○
VNX	●	●	●	○	●	○	○	○	○
Cloonix	●	●	●	●	○	○	○	○	●
CORE	●	●	○	●	○	●	○	●	○
MobiEmu	●	●	●	●	○	●	○	○	○
NEmu	●	●	●	●	●	●	●	●	●

● *Oui* ○ *Non*

TAB. 2 – *Comparaison des environnements réseaux virtuels.*

6 Conclusion

NEmu et ses utilitaires compilés *vswitch* et *nemo*, sont des outils pour la création et la gestion de réseaux virtuels dynamiques et hétérogènes pouvant être fixes ou mobiles. Ces outils fournissent un bon compromis entre facilité d'utilisation, coût faible et réalisme.

Ces réseaux virtuels peuvent être distribués sur plusieurs hôtes physiques et administrés par plusieurs utilisateurs sans droit particulier sur l'infrastructure réelle. Ils peuvent également évoluer en temps réel grâce à *nemo* suivant un scénario de connectivité préalablement calculé à partir d'un scénario de mobilité défini par un générateur externe.

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. Ce type d'*overlay* est donc inadapté aux applications temps réel.

NEmu peut donc être utilisé pour tester et évaluer des applications sur des environnements réseaux émulés. Cela permet de s'affranchir des équipements physiques standards tout en exécutant le code réel des applications. Ainsi il est possible d'obtenir un premier niveau de validation et d'évaluation de performances sur le code d'une application réseau avec un réalisme supérieur à celui que peut offrir un simulateur réseau.

Plusieurs étapes sont déjà prévues concernant 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'intégration à un simulateur de mobilité tel que *JBotSim* (Casteigts, 2010) ;
- le chargement dans *nemo* des fichiers de sortie de générateurs externes de cartes et de modèles de mobilité ;
- l'amélioration du réalisme des équipements virtuels (e.g., *switch*, *hub*, etc.) ;
- l'implémentation d'équipements sans fil (e.g., points d'accès, etc.).

Remerciement

Ce travail est co-financé par la *Direction Générale de l'Armement* [1] et la *Région Aquitaine* [2].

1. `http://www.defense.gouv.fr/dga`
2. `http://aquitaine.fr`

Références

Ahrenholz, J., C. Danilov, T. Henderson, et J. Kim (2008). Core : A real-time network emulator. In *Proceedings of IEEE MILCOM*, pp. 1–7.

Anuzelli, G. (2006). *Dynagen*. `http://dynagen.org`.

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

Bavier, A., M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, et M. Wawrzoniak (2004). Operating system support for planetary-scale network services. In *Proceedings of the 1st USENIX NSDI*, pp. 253–266.

Bavier, A., N. Feamster, M. Huang, L. Peterson, et J. Rexford (2006). In vini veritas : realistic and controlled network experimentation. In *Proceedings of ACM SIGCOMM*, pp. 3–14.

Bellard, F. (2005). QEMU, a fast and portable dynamic translator. In *Proceedings of USENIX Annual Technical Conference, FREENIX Track*, pp. 41–46.

Benchaib, Y. et A. Hecker (2011). Virconel : A network virtualizer. In *Proceedings of the 19th IEEE MASCOTS*, pp. 429–432.

Borgnat, P., G. Dewaele, K. Fukuda, P. Abry, et K. Cho (2009). Seven Years and One Day : Sketching the Evolution of Internet Traffic. In *Proceedings of the 28th IEEE INFOCOM*, pp. 711–719.

Casteigts, A. (2010). The jbotsim library. *CoRR abs/1001.1435*.

Che, J., Y. Yu, C. Shi, et W. Lin (2010). A synthetical performance evaluation of openvz, xen and kvm. In *Proceedings of IEEE APSCC*, pp. 587–594.

Chowdhury, N. et R. Boutaba (2009). Network virtualization : state of the art and research challenges. *IEEE Communications Magazine 47*(7), 20–26.

Davoli, R. (2005). Vde : Virtual distributed ethernet. In *Proceedings of IEEE TridentCom*, pp. 213–220.

Dewaele, G., Y. Himura, P. Borgnat, K. Fukuda, P. Abry, O. Michel, J.J., R. Fontugne, K. Cho, et H. Esaki (2010). Unsupervised host behavior classification from connection patterns. *International Journal of Network Management 20*, 317–337.

Dike, J. (2000). A user-mode port of the Linux kernel. In *Proceedings of Linux Showcase and Conference*, Volume 2.

DIT (2003). *VNUML*. `http://dit.upm.es/vnumlwiki`.

DIT (2008). *VNX*. `http://dit.upm.es/vnxwiki`.

Domingues, P., F. Araujo, et L. Silva (2009). Evaluating the performance and intrusiveness of virtual machines for desktop grid computing. In *Proceedings of IEEE IPDPS*, pp. 1–8.

EMC (2004). *VMware*. `http://www.vmware.com`.

Eriksen, M. (2005). Trickle : A userland bandwidth shaper for unix-like systems. In *Proceedings of USENIX Annual Technical Conference, FREENIX Track*, pp. 43.

Fillot, C. (2007). *Dynamips*. `http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator`.

FreeBSD (1993). *FreeBSD*. http://www.freebsd.org.

GARR, C. (2008). *FEDERICA*. http://www.fp7-federica.eu.

GNS3 (2007). *Graphical Network Simulator*. http://www.gns3.net.

Graphviz (1988). *Graph Visualization Software*. http://www.graphviz.org.

Hashimoto, M. et J. Bender (2010). *Vagrant*. http://vagrantup.com.

Hemminger, S. et al. (2005). Network emulation with netem. In *Linux Conf Au*, pp. 18–23.

Henderson, T., M. Lacage, G. Riley, C. Dowell, et J. Kopena (2008). Network simulations with the ns-3 simulator. *ACM SIGCOMM demonstration*.

Hong, X., M. Gerla, G. Pei, et C.-C. Chiang (1999). A group mobility model for ad hoc wireless networks. In *Proceedings of the 2nd ACM MSWiM*, pp. 53–60.

Hubert, B., G. Maxwell, R. Van Mook, M. Van Oosterhout, P. Schroeder, et J. Spaans (2003). Linux advanced routing & traffic control. In *Ottawa Linux Symposium*, pp. 213–222.

Jiang, X. et D. Xu (2003). Violin : Virtual internetworking on overlay infrastructure. In *Proceedings of the 2nd Springer ISPA*, pp. 937–946.

Kneale, B., A. Y. De Horta, et I. Box (2004). Velnet : virtual environment for learning networking. In *Proceedings of the 6th Australasian Conference on Computing Education*, Volume 30, pp. 161–168.

KVM. *Virtio*. http://www.linux-kvm.org/page/Virtio.

Kwon, G., J. Byers, et al. (2004). Roma : Reliable overlay multicast with loosely coupled tcp connections. In *Proceedings of the 23th IEEE INFOCOM*.

Lezcano, D. (2008). *LXC*. http://lxc.sourceforge.net.

Liang, B. et Z. J. Haas (1999). Predictive distance-based mobility management for pcs networks. In *Proceedings of IEEE INFOCOM*, pp. 1377–1384.

Lodo, J.-V. et L. Saiu (2007). *Marionnet*. http://www.marionnet.org.

Magoni, D. (2012a). *Network Mobilizer*. http://www.labri.fr/perso/magoni/nemo.

Magoni, D. (2012b). *Virtual Switch*. http://www.labri.fr/perso/magoni/vswitch.

McGregor, A., M. Hall, P. Lorier, et J. Brunskill (2004). Flow clustering using machine learning techniques. *ACM PAM*, 205–214.

Mell, P. et T. Grance (2009). The nist definition of cloud computing. *National Institute of Standards and Technology 53*(6).

Microsoft (2008). *Hyper-V*. www.microsoft.com/hyper-v-server.

NBD. *Network Block Device*. http://nbd.sourceforge.net.

NetFilter (2000). *NetFilter*. http://www.netfilter.org.

Oracle (2007). *VirtualBox*. https://www.virtualbox.org.

Parallels (2006). *OpenVZ Linux Containers*. http://wiki.openvz.org.

Perrier, V. (2007). *Cloonix*. http://clownix.net.

Pfaff, B., J. Pettit, T. Koponen, K. Amidon, M. Casado, et S. Shenker (2009). Extending networking into the virtualization layer. *Proceedings of ACM HotNets workshop*.

Pizzonia, M. et M. Rimondini (2008). Netkit : easy emulation of complex networks on inexpensive hardware. In *Proceedings of the 4th IEEE TridentCom*, pp. 1–10.

Puljiz, Z. et M. Mikuc (2003). *IMUNES*. `http://imunes.tel.fer.hr`.

Ribiere, A. (2010). Emulation of obsolete hardware in open source virtualization software. In *Proceedings of the 8th IEEE INDIN*, pp. 354–360.

Shingledecker, R. (2008). *TinyCore Linux*. `http://tinycorelinux.net`.

Sundararaj, A. I., A. Gupta, et P. A. Dinda (2004). Dynamic topology adaptation of virtual networks of virtual machines. In *Proceedings of the 7th LCR Workshop*, pp. 1–8.

TcpDump (1987). *LibPcap*. `http://www.tcpdump.org`.

Urvoy-Keller, G. et E. Biersack (2002). A congestion control model for multicast overlay networks and its performance. In *Proceedings of the 4th International Workshop on NGC*.

Vahdat, A., K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, et D. Becker (2002). Scalability and accuracy in a large-scale network emulator. In *The 5th ACM SIGOPS Operating Systems Review*, pp. 271–284.

Van Vorst, N., M. Erazo, et J. Liu (2011). Primogeni : Integrating real-time network simulation and emulation in geni. In *Proceedings of IEEE PADS*, pp. 1–9.

Varga, A. et al. (2001). The omnet++ discrete event simulation system. In *Proceedings of ESM*, Volume 9.

Yamini, B. et D. Selvi (2010). Cloud virtualization : A potential way to reduce global warming. In *Proceedings of IEEE RSTSCC*, pp. 55–57.

Summary

Experimentation is generally the last step before launching a network application in the wild. However, it is often difficult to gather enough hardware resources, control and mobility for experimenting in order to test and validate an application. Virtualization is thus a reliable and cheap technique for creating such an experimentation testbed. We propose a tool called *NEmu* designed to create virtual static, dynamic or mobile networks for testing and evaluating prototypes of network applications with a complete control over the network topology, link properties and mobility behavior. *NEmu* allows users to create customized topologies with limited hardware resources and without any administrative rights. We illustrate the use of *NEmu* in the context of a file distribution application upon a TCP connections tree. We evaluate the impact of such a tree on data rates and delays depending on the number of tree nodes, packets size and general bandwidth of links.