

NEmu : un outil de virtualisation de réseaux à la demande pour l'enseignement

Vincent Autefage

Univ. Bordeaux, LaBRI

Damien Magoni

Univ. Bordeaux, LaBRI

Résumé

NEmu (Network Emulator for mobile universes) est un environnement permettant d'instancier des réseaux virtuels statiques, dynamiques ou mobiles avec un contrôle accru sur les propriétés de la topologie ainsi que sur la configuration des nœuds et des inter-connexions. Cet outil trouve son utilité dans les enseignements mais peut aussi être utilisé comme plate-forme de tests ou bien encore comme outil d'aide à la recherche. NEmu permet de créer des réseaux virtuels à la volée, distribués sur plusieurs hôtes et sans droits particuliers sur l'infrastructure physique sous-jacente. Notre outil repose sur plusieurs caractéristiques fondamentales héritées d'un concept appelé NVE (Network Virtualization Environment) lui conférant une flexibilité importante dans le but de répondre aux besoins les plus spécifiques en matière de création de réseaux virtuels.

Mots clefs

Réseau virtuel, virtualisation, QEMU

1 Introduction

Prônée depuis de nombreuses décennies par divers philosophes et pédagogues [1], la pratique dans l'apprentissage d'une discipline est nécessaire pour en acquérir la maîtrise. De récentes études [2] pointent cette vérité en précisant qu'un apprentissage actif (i.e. pratique) augmenterait les résultats scolaires des élèves ou étudiants à hauteur de 84%. Ceci est d'autant plus vrai pour l'informatique [3] qui nécessite une pratique accrue afin d'accéder à la maîtrise de ses différentes branches et sous-domaines.

Il est donc indispensable de fournir aux enseignants et aux étudiants les moyens structurels nécessaires leur permettant facilement de pratiquer.

Certains enseignements requièrent l'utilisation d'infrastructures particulières en termes de taille, d'hétérogénéité, etc. Par exemple, l'administration des systèmes et des réseaux requiert, pour chaque étudiant, un certain nombre de machines, un accès à l'ensemble des droits d'administration, etc. La programmation distribuée peut, quant à elle, nécessiter une certaine dynamique, ou encore une mobilité particulière du réseau. Ces types de contraintes peuvent s'avérer complexes à mettre en œuvre et extrêmement coûteuses.

En effet, utiliser directement Internet comme plate-forme de test est assez peu praticable dans la mesure où très peu de paramètres peuvent être contrôlés. Acquérir sa propre infrastructure nécessite d'importants moyens, tant sur le plan financier que sur l'espace nécessaire à son stockage. De plus, la dynamique 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 [4]. C'est pour ces raisons que nous proposons un outil nommé *NEmu* [5] (*Network Emulator for mobile universes*), permettant d'instancier des réseaux virtuels statiques, dynamiques ou mobiles avec un contrôle accru sur les propriétés de la topologie ainsi que sur la configuration des nœuds et des inter-connexions.

Un réseau virtuel utilise des machines virtuelles, au lieu d'utiliser directement des machines physiques, et les interconnecte 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.

Notre contribution s'articule autour de plusieurs points. Dans un premier temps nous présentons en section 2 une description du concept théorique sur lequel repose notre outil. Par la suite, en section 3, nous proposons une description détaillée de notre outil *NEmu* permettant de créer et de gérer des réseaux statiques, dynamiques ou mobiles dans le but d'émuler une topologie réseau arbitraire et évolutive. Enfin, nous fournissons en section 4 des exemples concrets d'utilisation de *NEmu* illustrant la mise en place de séances de travaux pratiques.

2 Concept fondamental de NEmu

NEmu est une implémentation d'un concept appelé *NVE* [6] (*Network Virtualization Environment*). 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* garantit 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 réels plus ou moins anciens.
- La *programmabilité* fournit des services réseaux auxiliaires facilitant l'utilisation du VN (comme DHCP ou DNS par exemple).

De plus, *NEmu* inclut quatre 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, tels que les universités ou les laboratoires, ne fournissent pas de droits supérieurs aux utilisateurs sur leurs infrastructures afin de prévenir tout risque de sécurité ou d'intégrité sur l'ensemble du domaine. C'est pourquoi *NEmu* fonctionne en mode utilisateur standard.
- La *dynamisme* 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.
- La *mobilité* permet à une topologie d'évoluer au cours du temps de manière prédéfinie. En d'autres termes, il est possible de définir un scénario de connectivité évolutif autonome.
- 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 peut également former un unique AS composé de plusieurs VN.

L'ensemble de ces propriétés constitue un nouveau concept que nous appelons *NVE augmenté*. *NEmu* peut être considéré comme une *Infrastructure As A Service* [7] virtuelle (i.e. *virtual IaaS*), 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.

3 Description de NEmu

NEmu est un programme d'environ 8000 lignes, écrit en python, permettant de créer un réseau distribué de machines virtuelles. Il est basé sur plusieurs briques virtuelles distinctes. En effet, *NEmu* utilise des *nœuds virtuels* inter-connectés par des *liens virtuels* dans le but de créer une topologie arbitraire. 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*. Un ou plusieurs utilisateurs distincts peuvent administrer un même réseau virtuel découpé en sessions distinctes. Le *mobilisateur réseau* permet de faire évoluer la connectivité de la topologie au cours du temps en simulant des mouvements physiques appliqués aux nœuds virtuels dans une zone délimitée.

3.1 Éléments virtuels

3.1.1 Les nœuds virtuels

Un *nœud virtuel* dans *NEmu* (appelé *VNode*) est une machine virtuelle émulée par *QEMU* [8] 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 ou bien sur un hôte distant. 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.

Un *VHost* nécessite une image disque créée au préalable et fournie par l'utilisateur. 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* [9] ainsi qu'une couche QoS avec *Traffic Control* [10]. 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* [11] qui est une distribution Linux extrêmement légère et hautement configurable.

3.1.2 Les liens virtuels standards

Un *lien virtuel* dans *NEmu* (appelé *VLink*) est une inter-connexion réseau émulée entre deux *entités virtuelles* ou bien entre une *entité virtuelle* et le *monde réel*. 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 réalisée par un programme tiers.

Un certain nombre de *liens virtuels* existent au sein de *NEmu* :

- Un *VLine* est un *lien virtuel point-à-point* inter-connectant deux entités virtuelles.
- Un *VHub/VDEHub* est un *lien virtuel multi-points* émulant un concentrateur Ethernet (*hub*) et pouvant inter-connecter un ensemble d'entités virtuelles.
- Un *VSwitch/VDESwitch* est un *lien virtuel multi-points* émulant un commutateur Ethernet (*switch*) et pouvant inter-connecter un ensemble d'entités virtuelles.
- Un *VSlirp* est un *lien virtuel point-à-point* permettant à une machine virtuelle d'accéder au monde réel. Ce système utilise une technique proche du NAT [12] afin de permettre à une machine virtuelle de transmettre (et de recevoir) des trames Ethernet allant vers (ou provenant) des cartes réseaux physiques de l'hôte qui l'héberge.
- Un *VTap* est un *lien virtuel point-à-point* reposant sur la technologie *TUN/TAP* [13] et pouvant inter-connecter une entité virtuelle au monde réel au travers du noyau du système d'exploitation de l'hôte qui l'héberge.
- Un *VRemote* est un *lien virtuel point-à-point* permettant de connecter deux éléments virtuels qui vivent sur deux hôtes physiques différents.

Les *liens virtuels* transportent des trames Ethernet d'une interface réseau virtuelle à une ou plusieurs autres. Nous avons conçu un programme nommé *vnd* [14], composé de 5000 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 *vnd* de régler à chaud la bande passante, le délai, la gigue, le taux d'erreurs ainsi que la taille de file sur n'importe quelle interface et de manière strictement indépendante, ce qui nous permet d'obtenir un réglage plus fin de la topologie réseau. Il est également possible d'utiliser des VLAN de niveau 1 (port). L'ensemble de ces réglages est

applicable aux flux ascendants et descendant de manière indépendante. Dans ce cas, le trafic Ethernet est transporté entre les interfaces réseaux par des tunnels mis en place sur des connexions TCP ou UDP.

Nous apportons également une intégration de la suite logicielle VDE [15] qui utilise des mécanismes de mémoire partagée plutôt que des sockets réseau pour effectuer les inter-connexions entre les différentes entités virtuelles.

Le support des interfaces systèmes TUN/TAP augmente également les possibilités d'interactions entre le réseau virtuel et le monde réel.

Cette variété de types de connexions (TCP, UDP, VDE, TAP) permet d'éventuels raccordements entre un réseau virtuel généré par NEmu et des composants externes tel que des machines virtuelles tierces ou encore des hôtes physiques particuliers.

La Figure 1 représente un exemple de topologie réseau générée par NEmu. Sur le côté gauche, deux VHost sont inter-connectés à un VRouter au travers d'un VSwitch en utilisant des tunnels TCP. Sur le côté 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 vnd.

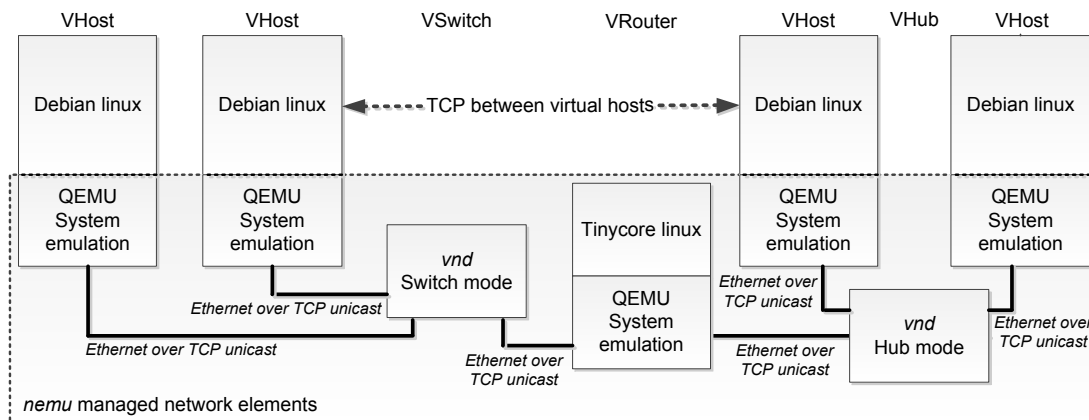


Figure 1 - Éléments réseaux de NEmu en action.

3.1.3 Les liens virtuels mobiles

Nous avons également conçu un certain nombre de liens virtuels dédiés à la mobilité appelés VAirLink. L'émulateur QEMU ne disposant pas d'interface réseau sans fil, nous avons implémenté des équipements sans fil afin d'intégrer la notion de lien aérien :

- Un VAirWic est un équipement virtuel fournissant à une entité virtuelle une *interface de communications sans fil*.
- Un VAirAp est un équipement virtuel assimilé à un *commutateur Ethernet* mais disposant d'une *interface de communications sans fil* supplémentaire.
- Un VAirRemote est équipement virtuel permettant à des VAirLink reposant sur des hôtes physiques différents de pouvoir communiquer.

Les communications sans fil sont exclusivement transportées par des tunnels UDP. Les éléments de type VAirLink sont implémentés par des vnd, il est donc possible de fixer la bande passante, le délai, la gigue, la taille de file et le taux d'erreurs de n'importe quel tunnel.

3.2 Gestion du réseau virtuel

3.2.1 Session

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, un réseau virtuel fourni 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 différente infrastructure.

3.2.2 Manager

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 différentes :

- 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.

3.2.3 Mobilisateur réseau

Le *mobilisateur réseau* permet de faire évoluer la connectivité des équipements sans fil d'un réseau virtuel au cours du temps. Cette évolution repose sur un *scénario de connectivité*, c'est à dire une suite temporelle d'événements de connexion et de déconnexion entre des nœuds mobiles représentant les liens sans fil. *nemo* [16] est un programme léger de 3000 lignes écrites en C++ permettant à la fois de générer un scénario de connectivité, mais également d'envoyer des ordres à *NEmu* en temps réel afin d'appliquer les changements de connectivité entre nœuds mobiles en créant, détruisant ou changeant les caractéristiques des liens aux moments voulus.

Le scénario de connectivité peut être directement écrit par l'utilisateur, sous forme d'un fichier texte, ou bien généré automatiquement par *nemo* selon un *scénario de mobilité* prédéfini. Le scénario de mobilité peut être lui même défini explicitement par l'utilisateur sous la forme d'un fichier texte contenant la position, la vitesse et l'accélération de chaque nœud à intervalle de temps régulier. Il peut être également auto-généré grâce à un certain nombre de paramètres comme la taille de la zone de simulation, la vitesse ainsi que l'accélération maximum des éléments dans la zone, la durée et la précision temporelle de la simulation. Pour le moment les zones sont des plans rectangulaires vides, à terme il sera possible de définir des zones en trois dimensions avec éventuellement des obstacles.

L'*ordonnanceur temps réel* intégré à *nemo* transmet les événements de connectivité à *NEmu* à leur date exacte fixée par rapport au début du scénario. Son fonctionnement est illustré à la Figure 2.

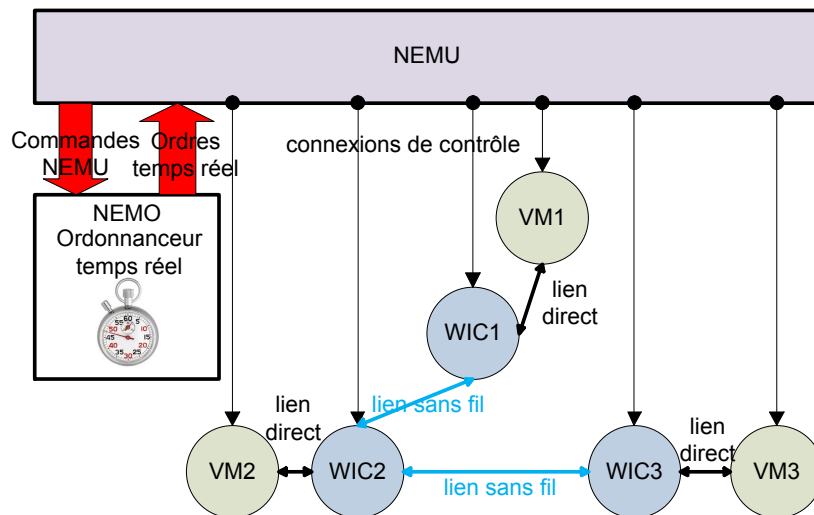


Figure 2 - Interactions entre NEmu et nemo.

3.2.4 Visualisation

Une topologie peut être visualisée à tout moment au travers de la suite logicielle *Graphviz* [17].

4 Exemples d'utilisation

Afin d'illustrer l'utilisation de *NEmu*, nous proposons ici trois scénarios détaillés pouvant être utilisés lors de séances de travaux pratiques. La première porte sur l'administration réseau (réseau filaire simple); la seconde sur la sécurité réseau (réseau hybride distribué); la dernière sur la programmation distribuée (réseau mobile).

4.1 Scénario filaire simple

Ce scénario s'inscrit dans une séance d'initiation à l'administration réseau et plus particulièrement l'adressage et le routage de sous réseaux filaires. Chaque étudiant dispose ici d'un réseau virtuel isolé qu'il peut exploiter à sa guise. Nous avons auparavant créer une image disque *Debian* [18] pour les machines virtuelles. La Figure 3 illustre la topologie accompagnée de son script *NEmu*. Dans ce scénario, le réseau virtuel est composé de 3 postes sous *Debian* inter-connectés par un switch géant ainsi un slirp (réseau réel) faisant ainsi office de passerelle vers le monde extérieur. Le but pour l'étudiant est ici de configurer le réseau pour que l'ensemble des nœuds puissent communiquer entre eux ainsi que vers l'extérieur.

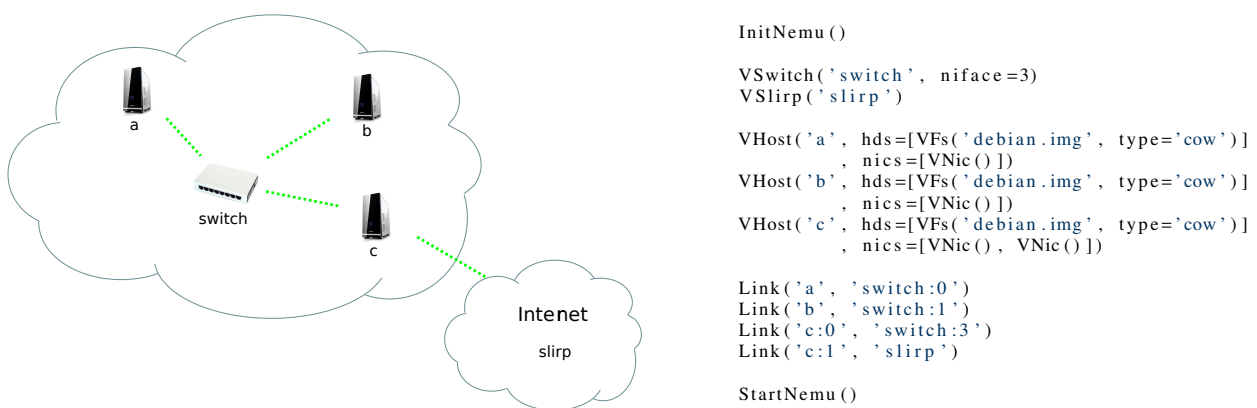


Figure 3 - Illustration et script *NEmu* du scénario filaire simple.

4.2 Scénario hybride distribué

Ce scénario est la base d'une séance de sécurité réseau. Cette topologie nécessite qu'un réseau virtuel soit composé par deux étudiants depuis deux machines physiques différentes. Le réseau virtuel est donc constitué de deux sessions distinctes. L'un des étudiants possède une infrastructure qu'il doit protéger des attaques extérieurs, l'autre est un attaquant ayant pour but de prendre le contrôle d'un équipement mobile particulier qui se trouve dans le réseau du défenseur. Une illustration du réseau virtuel global est représentée en Figure 4.

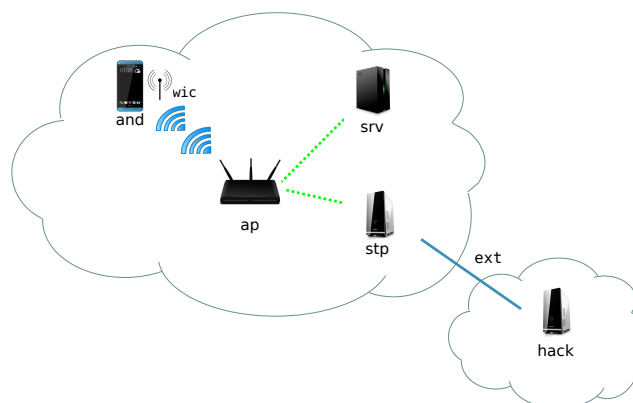


Figure 4 - Illustration du scénario hybride distribué.

Le défenseur dispose d'un réseau virtuel constitué d'une passerelle (*stp*), d'un serveur DHCP/DNS (*srv*) qui est ici auto-géré par *NEmu*, d'un point d'accès sans fil (*ap*) ainsi que d'un terminal mobile (*and*) sous *Android* [19]. Ce dernier est relié au point d'accès par un équipement de type *VAirWic* (*wic*) permettant de connecter le nœud mobile au point d'accès. L'attaquant quant à lui dispose d'une unique machine (*hack*) utilisant la distribution *Kali Linux* [20] et reliée à la machine *stp* du défenseur. Les scripts *NEmu* permettant de générer cette topologie sont fournis en Figure 5.

```
# Session victime [10.0.0.1]
InitNemu ()

VAirAp('ap', niface=2)
VAirWic('wic')
VLine('ext')

VHost('stp', hds=[VFs('debian.img', type='cow')]
      , nics=[VNic(), VNic()])
VHost('and', hds=[VFs('android.img', type='cow')]
      , nics=[VNic()], m=512)

VRouter('srv', nics=[VNic()]
      , services=[Service('dnsmasq')
      , Service('ifup', '1:192.168.0.254')
      , Service('dnsmasq', domain='local'
      , net='192.168.0.0/24'
      , start='192.168.0.10'
      , end='192.168.0.20')
      , Service("sshd")])

Link('stp:0', 'ap:0')
Link('stp:1', 'ext:0')
Link('srv', 'ap:1')

Link('and', 'wic')
Join('wic', 'ap')

SetIface('ext:1', addr='10.0.0.2', port=8080)

StartNemu ()

# Session pirate [10.0.0.2]
InitNemu ()

VRemote('remote')

VHost('hack', hds=[VFs('kali.img', type='cow')]
      , nics=[VNic(hw='a0:d0:2b:62:ab:37')])

SetIface('remote', addr='10.0.0.1', port=8080)

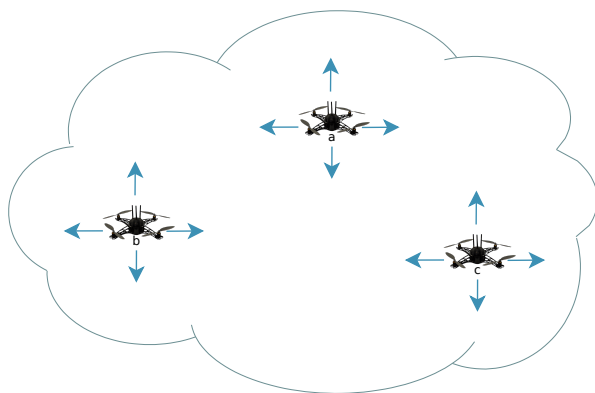
Link('hack', 'remote')

StartNemu ()
```

Figure 5 - Scripts *NEmu* du scénario hybride distribué.

4.3 Scénario mobile

Ce scénario s'inscrit dans l'étude d'un algorithme distribué dédié aux réseaux de drones. Une illustration de la topologie virtuelle utilisée accompagnée de son script *NEmu* est fournie en Figure 6. Ici, les inter-connexions entre les 3 drones sont directement générées par le mobilisateur de *NEmu* pour une durée de 20 secondes comportant 100 événements de mobilité (changement de vitesse, d'accélération ou de direction pour un nœud du réseau).



```
InitNemu ()

VAirWic('awic')
VAirWic('bwic')
VAirWic('cwic')

VHost('a', hds=[VFs('drone.img', type='cow')]
      , nics=[VNic()])
VHost('b', hds=[VFs('drone.img', type='cow')]
      , nics=[VNic()])
VHost('c', hds=[VFs('drone.img', type='cow')]
      , nics=[VNic()])

Link('a', 'awic')
Link('b', 'bwic')
Link('c', 'cwic')

MobNemu('mob', nodes=['awic', 'bwic', 'cwic'])
GenMobNemu('mob', width=1000, height=1000
      , time=20, events=100)

StartNemu ()
StartMobNemu('mob')
```

Figure 6 - Illustration et script *NEmu* du scénario mobile.

5 Conclusion

NEmu accompagné des outils *vnd* et *nemo*, permet la création et la gestion de réseaux virtuels dynamiques et hétérogènes pouvant être fixes ou mobiles. Il propose un compromis entre facilité d'utilisation, coût réduit et réalisme. Le respect du concept *NVE augmenté* garantit à *NEmu* une grande flexibilité, permettant ainsi de concevoir des topologies diverses, des plus simples aux plus complexes. *NEmu* peut être utilisé dans le cadre des enseignements mais aussi pour des tests ou des évaluations en recherche.

Les étapes en cours d'implémentation et futures du développement de *NEmu* sont les suivantes :

- la réalisation d'une interface graphique ;
- l'intégration de capacités de migration pour les sessions afin d'intégrer l'équilibrage de charge ;
- l'intégration de nouveaux services pour les VRouter ;
- l'ajout de nouvelles options pour *vnd* (*spanning tree*, VLAN de niveaux 2 et 3, etc.) ;
- l'ajout de nouvelles options pour *nemo* (carte 3D, obstacles, modèles de mobilité plus complexes, connectivité plus précise, etc.).

Remerciement

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

Bibliographie

- [1] Gérard Barnier. Philosophie de l'éducation grands courants pédagogiques. IUFM Aix Marseille, 2003.
- [2] Jean-Michel Fourgous. Apprendre autrement à l'ère numérique. *Rapport de la mission parlementaire Fourgous*, Février 2012.
- [3] Orit Hazzan, Tami Lapidot, et Noa Ragonis. *Guide to Teaching Computer Science, An Activity-Based Approach*. Springer, 2011.
- [4] B. Yamini et D.V. Selvi. Cloud virtualization : A potential way to reduce global warming. Dans *IEEE RSTSCC*, pages 55–57, novembre 2010.
- [5] Vincent Autefage. *NEmu*, 2012. <http://nemu.valab.net>.
- [6] N.M.M.K. Chowdhury et R. Boutaba. Network virtualization : state of the art and research challenges. *IEEE Communications Magazine*, 47(7) :20–26, 2009.
- [7] Peter Mell et Timothy Grance. The nist definition of cloud computing. *National Institute of Standards and Technology*, 53(6), 2009.
- [8] Fabrice Bellard. QEMU, a fast and portable dynamic translator. Dans *Proceedings of USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.
- [9] NetFilter. *NetFilter*, 2000. <http://www.netfilter.org>.
- [10] B. Hubert, G. Maxwell, R. Van Mook, M. Van Oosterhout, P.B. Schroeder, et J. Spaans. Linux advanced routing & traffic control. Dans *Ottawa Linux Symposium*, pages 213–222, 2003.
- [11] Robert Shingledecker. *TinyCore Linux*, 2008. <http://tinycorelinux.net>.
- [12] P. Srisuresh et K. Egevang. Traditional ip network address translator (traditional nat). *IETF*, January 2001.
- [13] Maksim Yevmenkin et Florian Thiel. *TUN/TAP*, 2002. <http://www.kernel.org/doc/Documentation/networking/tuntap.txt>.
- [14] Damien Magoni. *Virtual Network Device*, 2012. <http://www.labri.fr/perso/magoni/vnd>.
- [15] Renzo Davoli. Vde : Virtual distributed ethernet. Dans *Proceedings of IEEE TridentCom*, pages 213–220, 2005.
- [16] Damien Magoni. *Network Mobilizer*, 2012. <http://www.labri.fr/perso/magoni/nemo>.
- [17] Graphviz. *Graph Visualization Software*, 1988. <http://www.graphviz.org>.
- [18] SPI. *Debian*, 1993. <http://www.debian.org>.
- [19] Google. *Android*, 2007. <http://www.android.com>.
- [20] Offensive Security. *Kali-linux*, 2013. <http://www.kali.org>.

1. <http://www.defense.gouv.fr/dga>

2. <http://aquitaine.fr>