Tirer les ficelles de l'architecture TCP/IP avec Marionnet

Franck Butelle
Jean-Vincent Loddo
IUT de Villetaneuse - Laboratoire d'Informatique de Paris Nord (LIPN, UMR 7030)
Université Paris 13, Avenue J.B. Clément, F-93430 Villetaneuse

Résumé

Marionnet est un logiciel permettant de définir, configurer, exécuter et contrôler un réseau virtuel constitué d'ordinateurs utilisant le système d'exploitation GNU/Linux, de concentrateurs, de commutateurs, et de routeurs. Avec Marionnet, il est possible d'expérimenter la mise en œuvre complète d'un réseau local : le projet, le câblage, le lancement, la configuration, l'administration, l'étude de protocoles et le test de services ou d'applications. Il permet de pratiquer, analyser et contrôler les différentes couches réseaux qui constituent l'architecture DoD (TCP/IP) : depuis le niveau physique jusqu'au niveau application. Adopté dans une structure universitaire, Marionnet permet de réduire l'utilisation des vraies salles de « TP réseau », constituées d'équipements souvent onéreux et difficiles à maintenir en parfait état de marche. Il permet aussi aux étudiants de travailler à distance ou dans les salles informatiques ordinaires en accès libre.

Mots clefs

Virtualisation, simulation, pédagogie, didacticiel, réseau, protocole, TCP/IP, OSI, VLAN, 802.1Q, STP, IP, ARP, TCP, UDP, SNAT, DNAT

1 Introduction

Marionnet est un outil développé à l'université Paris 13 dans le cadre de projets « e-learning » avec la contribution de l'IUT de Villetaneuse et du LIPN[1]. Il est employé avec succès depuis 2007 pour enseigner les réseaux et l'administration système à l'université Paris 13 ainsi que dans d'autres établissements universitaires en France (dont Paris 11 Orsay, Paris 12 Créteil et L'IUT de Béziers) et à l'étranger (dont Pise en Italie, Wuhan en Chine et Rio de Janeiro au Brésil). Sous licence GNU GPL, il est disponible en téléchargement à partir du site du projet http://www.marionnet.org[2] hébergé par l'Université Paris 13. L'interface graphique GTK (internationalisée) rend Marionnet facile à prendre en main, aussi bien pour les étudiants débutants que pour les administrateurs réseaux souhaitant tester des architectures complexes. Outre la vocation pédagogique originelle de Marionnet, cet outil peut également devenir une plate-forme de tests pour techniciens qui voudraient préparer un réseau, pratiquement grandeur nature, constitué de machines Linux et des divers équipements classiques d'interconnexion de réseaux. La topologie du réseau virtuel peut être très complexe. Marionnet est par ailleurs un logiciel libre basé sur des logiciels libres. Les principaux outils utilisés/appelés par Marionnet sont les suivants :

- UML (User Mode Linux [1]) qui permet d'exécuter un noyau linux en tant que simple utilisateur. Ce noyau doit être compilé spécialement à cette fin. L'administrateur (root) de cette machine virtuelle n'a rien à voir avec le compte administrateur de la machine hôte. De même, si un noyau UML « plante », l'hôte n'est pas affecté. Cela représente un avantage évident dans la gestion d'un parc de machines pour étudiants.
- VDE2 (Virtual Distributed Ethernet) du projet Virtualsquare [2]. VDE fourmille de petits outils de simulation d'équipement Ethernet depuis les câbles eux-mêmes jusqu'aux hubs et switchs ainsi qu'une pseudo internet-box (voir le composant gateway dans Marionnet).
- DOT, du projet Graphviz [3], pour la génération de l'image (dessin) du réseau virtuel dans son état courant.

Dans cet article, nous nous proposons de faire un tour d'horizon des capacités de l'outil en suivant le découpage en cinq couches : les quatre couches de la RFC 1122 (*Requirements for Internet Hosts -- Communication Layers*) [4] plus la couche physique. Nous remonterons les couches de l'architecture TCP/IP à partir des problèmes physiques de câblage (~ OSI 1), en passant par la configuration des commutateurs (VLAN, STP, niveau *Ethernet*, ~ OSI 2) et par celle des routeurs (IOS, RIP, OSPF, BGP, ISIS, niveau *Internet*, ~ OSI 3). Nous verrons enfin comment pratiquer le niveau *Transport* (OSI 4) avec de simples programmes client-serveur (en UDP ou TCP), et comment étudier le niveau *Application* (OSI 5 à 7) en analysant le trafic réseaux d'un protocole spécifique (FTP, DNS, HTTP, NFS...).

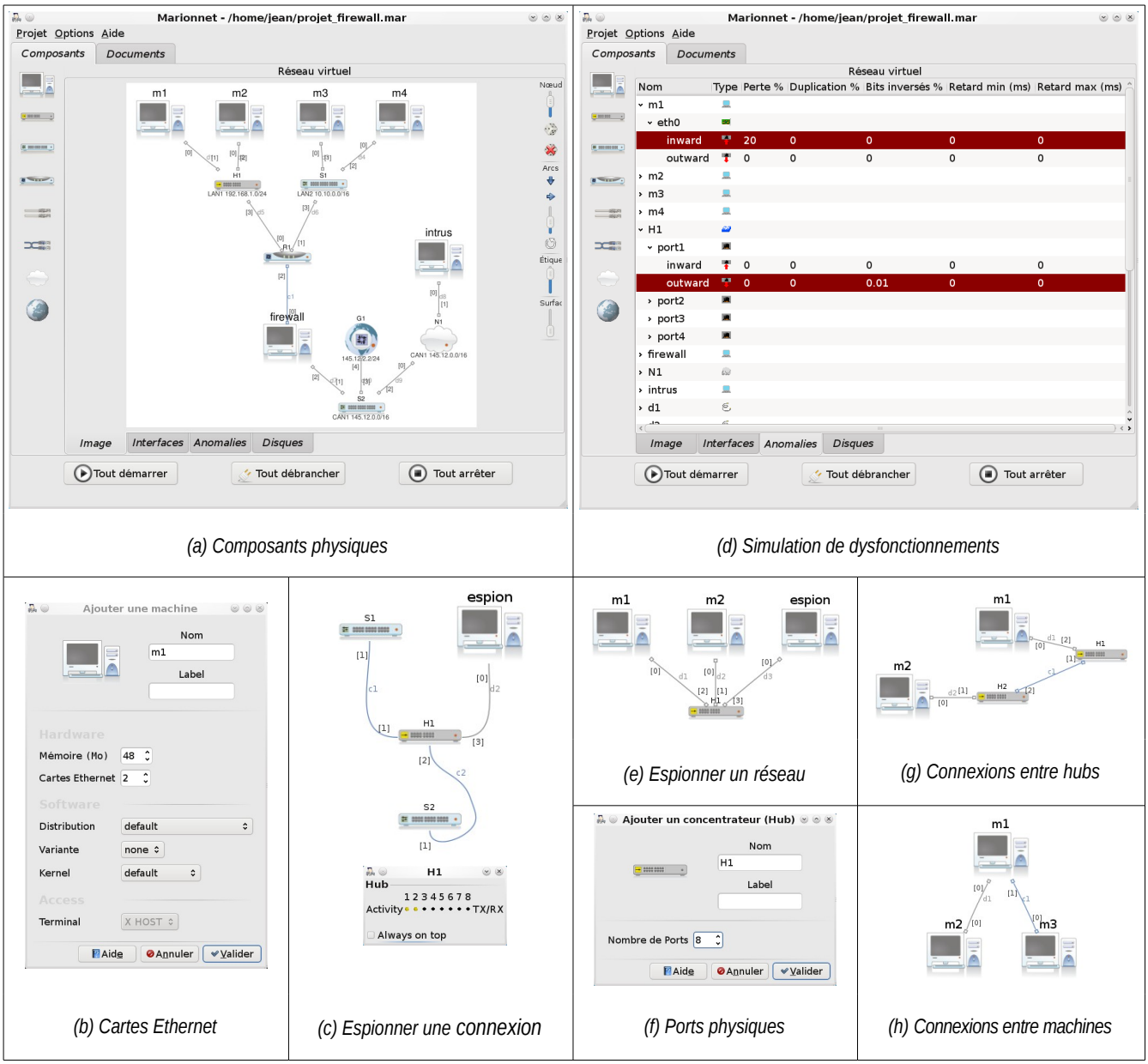
---

1 Laboratoire d'Informatique de Paris Nord

2 Des vidéos démonstratives sont disponibles à partir de la même adresse (onglet *Resources*).

1.1 Couche physique (OSI 1)

Travailler au niveau de la couche physique signifie essentiellement définir l'ensemble des appareils utiles, les ports physiques nécessaires pour chaque appareil, et l'ensemble des connexions entre ces ports. Des dysfonctionnements associés aux ports d'entrée-sortie (voir Figure 1.d) peuvent être éventuellement simulés, si l'utilisateur le souhaite.



(a) Composants physiques

(d) Simulation de dysfonctionnements

(b) Cartes Ethernet

(c) Espionner une connexion

(e) Espionner un réseau

(g) Connexions entre hubs

(f) Ports physiques

(h) Connexions entre machines

Figure 1 – *Couche Physique*

1.2 Composants physiques

Les composants physiques simulés par Marionnet sont disponibles sur la *palette des composants* située à gauche sur la fenêtre principale de l'application (voir Figure 1.a). On y trouve de haut en bas : la *machine*, le *concentrateur* (*hub*), le *commutateur* (*switch*), le *routeur*, le *câble droit*, le *câble croisé*, le *nuage*, la *passerelle internet* (*gateway* ou *bridge*). Notons que l'utilisateur ne dessine pas le réseau directement. En ajoutant les composants dans son projet, c'est-à-dire dans sa salle de TP virtuelle, il définit l'ensemble des *appareils* ou *nœuds* (tout ce qui n'est pas de type câble) et des *connexions* ou *arêtes* (câbles) qui constituent son réseau. Le *graphe* du réseau, qui est dessiné dans la partie centrale de l'interface, est alors mis à jour automatiquement en tenant compte des ajouts et retraits de composants et de l'état courant de chaque appareil (*éteint*, *en marche* ou *en pause*) et de chaque câble (*branché*, *débranché*). Par exemple, chaque machine est représentée par un *écran noir*, lorsque elle est éteinte et par un *écran azur* lorsqu'elle est en marche. En

particulier, l'état « en pause » des nœuds et l'état « débranché » des câbles, sont des états *logique*s, purement virtuels, qui servent à simuler des pannes ou interruptions de service occasionnelles et temporaires. Pour une description plus détaillée des différents composants de Marionnet et de leur comportement, voir [5] et [6]. Ajouté à partir de la version publique 0.90.x (février 2011), le composant *passerelle internet* (« world gateway »), n'est pas traité dans les publications citées. Il s'agit d'une sorte de « internet box », comme certains FAI[3] en proposent, équipée d'un commutateur et d'un service DHCP intégrés, qui agit comme une passerelle IP (routeur) entre le réseau virtuel et le réel. Ce composant permet, entre autres, de télécharger des fichiers, mettre à jour les machines virtuelles ou installer des logiciels par Internet.

Ports et interfaces réseaux. Dans la configuration physique d'un réseau, il faut prévoir, pour chaque nœud, un nombre de ports suffisant pour que le nœud en question remplisse son rôle dans le réseau. Le *nombre* de ports, qu'on appelle « interfaces réseaux » quand cela concerne les machines, peut donc être choisi *arbitrairement* par l'utilisateur. Toutefois, pour de simples raisons de performances, dans la version actuelle du logiciel, le nombre maximal d'interfaces réseaux est limité à 5 et celui des autres équipements à 16. En effet, le système GNU/Linux « hôte », hébergeant le réseau virtuel, pourrait se trouver encombré par trop de processus à gérer, 3 processus au moins étant nécessaires à la simulation de chaque port et des éventuels dysfonctionnements.

Reconfiguration dynamique. Dans l'idéal pédagogique du logiciel, qui est de simuler le plus fidèlement possible la réalité d'un réseau physique, les composants peuvent être ajoutés ou enlevés à chaud, c'est-à-dire pendant qu'une partie ou la totalité d'un réseau est en exécution. Le réseau peut donc être *reconfiguré dynamiquement*. La présence des boutons « Tout démarrer » et « Tout arrêter », visibles en bas de la fenêtre principale de l'application (Figure 1.a), ne doit pas faire croire que le logiciel oblige à une utilisation strictement *séquentielle*, du genre construction-démarrage-arrêt, du réseau dans sa globalité. En revanche, et comme dans la réalité, les composants sont *indépendants* les uns des autres : ils peuvent être ajoutés, se trouver dans n'importe quel état (éteint, en marche, en pause), changer d'état et être enlevés *individuellement*. Ce comportement dynamique peut être vérifié par une simple expérience. Il suffit de faire tourner un « ping en boucle » (un ECHO REQUEST par seconde) sur une machine en faisant en sorte que les premières requêtes échouent (à cause, par exemple, d'un mauvais câblage, ou d'un partenaire non existant ou éteint). On observera le « ping » se débloquer soudainement, lorsque la connexion sera réparée ; enfin, on observera le lien « se couper » suite à une nouvelle utilisation incorrecte du matériel physique.

Comportement à l'exécution. À l'exécution les nœuds se comportent de façon différente selon leur type. Les machines sont accessibles par des terminaux GNU/Linux sur lesquels on se connecte, comme sur tout système Unix, avec un « login » et un mot de passe (par défaut le compte « root » a comme mot de passe « root » ; certes, ce n'est pas conseillé, mais ce n'est qu'une machine virtuelle après tout !). Les concentrateurs, commutateurs, routeurs et passerelles (de type « gateway ») sont représentés à l'exécution par de petites fenêtres de diodes (voir Figure 1.c et Figure 2.b) qui clignotent, comme dans la réalité, au passage des trames. L'utilisateur peut demander à avoir accès à la configuration de chaque commutateur défini. Dans ce cas, une fenêtre graphique supplémentaire hébergeant le terminal d'accès au commutateur sera lancée. Pour les routeurs, l'utilisateur peut aussi demander un terminal d'accès. Il s'agit cependant d'un terminal Unix, les routeurs étant simulés (ou plus exactement « émulés »[4], de l'anglais technique « emulated ») par des systèmes GNU/Linux équipés du logiciel *quagga* (suite libre du projet *zebra*). Les nuages n'affichent pas de fenêtres à l'exécution, leur but étant de représenter un lien physique dont on ignore les caractéristiques, mais qui ne comporte que des équipements d'interconnexion de niveau 2.

Concentrateurs. Les *concentrateurs* (ou *Hubs*) sont des nœuds dits « de niveau 1 », au sens où leur activité est cloisonnée au traitement des signaux électriques (bits), sans que l'information véhiculée par ces signaux soit interprétée d'une quelconque manière. On les situe ainsi à plein titre dans la couche physique, l'interprétation des séquences de bits étant propre aux couches réseaux supérieures. Ils répliquent tout simplement les séquences de bits écoutées depuis un port sur tous les autres ports et sont, pour cette raison, appelés aussi *répéteurs multiports*. Ce comportement peut être vérifié, par exemple, avec l'exercice de Figure 1.e où la machine « espion » peut écouter une communication, qui ne la concerne pas, entre m1 et m2. En raison de leurs performances faibles par rapport aux commutateurs, les concentrateurs sont de moins en moins utilisés dans les réseaux réels. Ils sont cependant utiles pour observer du trafic réseau. La Figure 1.c montre une technique, utilisable aussi bien dans un réseau virtuel que dans la réalité, pour observer le trafic circulant sur un câble. Le câble reliant les deux commutateurs S1 et S2 est remplacé par deux câbles (c1 et c2) en interposant un concentrateur (H1). La fenêtre de diodes de H1 permet de s'intéresser au passage des trames alors que l'espion permet de s'intéresser à leur contenu.

[3] Fournisseur d'Accès Internet

[4] Le terme « simulation » est considéré réducteur, les noyaux invités étant identiques au noyau qui les héberge.

1.3 Connexions physiques

Polarité des ports et câblage. Marionnet laisse l'utilisateur libre de choisir n'importe quel type de câble, *droit* ou *croisé*, pour connecter n'importe quel type de paire de ports, *MDI*[5] ou *MDI-X*. C'est un piège, dans une démarche qui est pédagogique. Pour mémoire, MDI est un port type interface réseau de PC : l'émission se fait sur la paire 1-2 et la réception sur la paire 3-6. MDI-X est de type port de hub : émission sur la paire 3-6 et réception sur la paire 1-2. On relie une interface MDI à une MDI-X par un câble droit et deux MDI ou deux MDI-X par un câble croisé. Les appareils de niveau 1 et 2 (concentrateurs et commutateurs) ont des ports MDI-X, ceux de niveau 3 ou supérieur (machines et routeurs) ont des ports (interfaces) MDI, les autres nœuds (nuage et passerelles), pour lequel la question et l'intérêt pédagogique sont moins évidents, ont des ports *Auto MDI-X*, qui négocient automatiquement le croisement et fonctionnent ainsi avec n'importe quel câblage. Comme dans une situation réelle, l'utilisateur peut donc réaliser, éventuellement, un « mauvais câblage ». Dans ce cas, il sera obligé de le corriger par la suite, en détectant les liens « coupés », c'est-à-dire présents mais se comportant comme s'ils étaient absents. Par exemple, il est possible de connecter deux machines par un câble droit, solution incorrecte, ou par un câble croisé, solution correcte. La Figure 1.h illustre les deux connexions possibles, le *mauvais* câble (droit, dessiné en gris) entre m1 et m2, le *bon* câble (croisé, dessiné en bleu) entre m1 et m3 : seul le ping entre m1 et m3 fonctionnera. La Figure 1.g montre une connexion correcte entre concentrateurs réalisée avec un câble croisé. Des maquettes de « baies de brassage » peuvent ainsi être construites au besoin.

Dysfonctionnements des liaisons. Comme illustré dans la Figure 1.d, des dysfonctionnements des ports ou interfaces peuvent être simulés en entrée (« inward ») ou en sortie (« outward »). Les symptômes que le logiciel peut simuler sont : pertes de trames (en pourcentage), duplication de trames (en pourcentage), bits inversés (« flip ») n'importe où dans une trame (en pourcentage), retard de transmission minimum et maximum (loi gaussienne entre ces extrêmes).

2 Couche liaison (~ OSI 2)

À ce niveau, trois sujets fondamentaux peuvent être abordés avec Marionnet : étudier la structure des trames Ethernet 802.3, la mise en œuvre de *VLAN* de type 1 (basées sur une partition des ports physiques) et l'étude, et en partie la configuration, du *Spanning Tree Protocol* (STP) entre commutateurs reliés en boucle. Des exercices complémentaires très formateurs comme la répartition de charge (par exemple par l'*Ethernet Bonding*) et les pare-feux de niveau 2 (wire firewall) sont possibles en exploitant les fonctionnalités réseaux avancées du noyau Linux.

2.1 Étudier la structure des trames Ethernet 802.3

L'analyse de trames ARP est un exercice qui paraît bien adapté à l'étude de la norme 802.3. D'une part, ARP est un protocole à la frontière entre les couches Internet (Internet layer) et Liaison (Link layer). D'autre part, le nombre d'encapsulations est minimal (les trames véhiculent juste la requête en diffusion ou la réponse ARP) et il est possible de provoquer facilement un dialogue incluant ce type de trames en utilisant la commande `ping` sous GNU/Linux. On demande aux étudiants d'utiliser la commande `ifconfig` pour configurer les interfaces des machines et la commande `arp` pour vider la table (cache) ARP de la machine source si nécessaire. En même temps, pour capturer les trames, les étudiants utilisent un « sniffeur » en modalité graphique comme `wireshark`, disponible sur la plupart des machines virtuelles dotées du support graphique. D'autres machines virtuelles (par exemple celles de la série « pinocchio »[6]), privées de ce support (pour des raisons d'économie d'espace disque), proposent des sniffeurs en modalité texte comme `tcpdump`. Par expérience, ces derniers se révèlent intéressants pour avoir l'aperçu rapide d'une série de trames générées par un protocole de niveau supérieur (boucle ECHO ICMP, UDP, TCP, DHCP, DNS...). Ils sont cependant moins efficaces que leurs homologues graphiques, d'un point de vue pédagogique, lorsqu'on souhaite étudier la structure et l'information véhiculées par les différents protocoles encapsulés dans une même trame.

2.2 VLAN de type 1

Pour expliquer le concept et motiver l'introduction des VLAN, les étudiants peuvent commencer par réaliser plusieurs réseaux, par exemple un LAN1 en 192.168.1.0/24 et un LAN2 en 10.0.0.0/8, avec le même matériel de liaison (un ou plusieurs commutateurs et/ou concentrateurs, comme illustré dans les figures 2.d et 2.e). L'utilisateur constate d'abord que cela est possible, mais que les réseaux ainsi réalisés ne sont pas complètement étanches. En particulier, les diffusions ARP, étudiées auparavant, seront visibles sur les deux LAN. Plus grave, les diffusions DHCP, facilement provoquées par la commande `dhclient` sous GNU/Linux, seront écoutées par toutes les machines (dont d'éventuels serveurs DHCP a priori non concernés).

---

5 Medium Dependent Interface

6 Série de machines virtuelles (installations GNU/Linux) minimalistes conçues dans le cadre du projet et livrées avec le logiciel

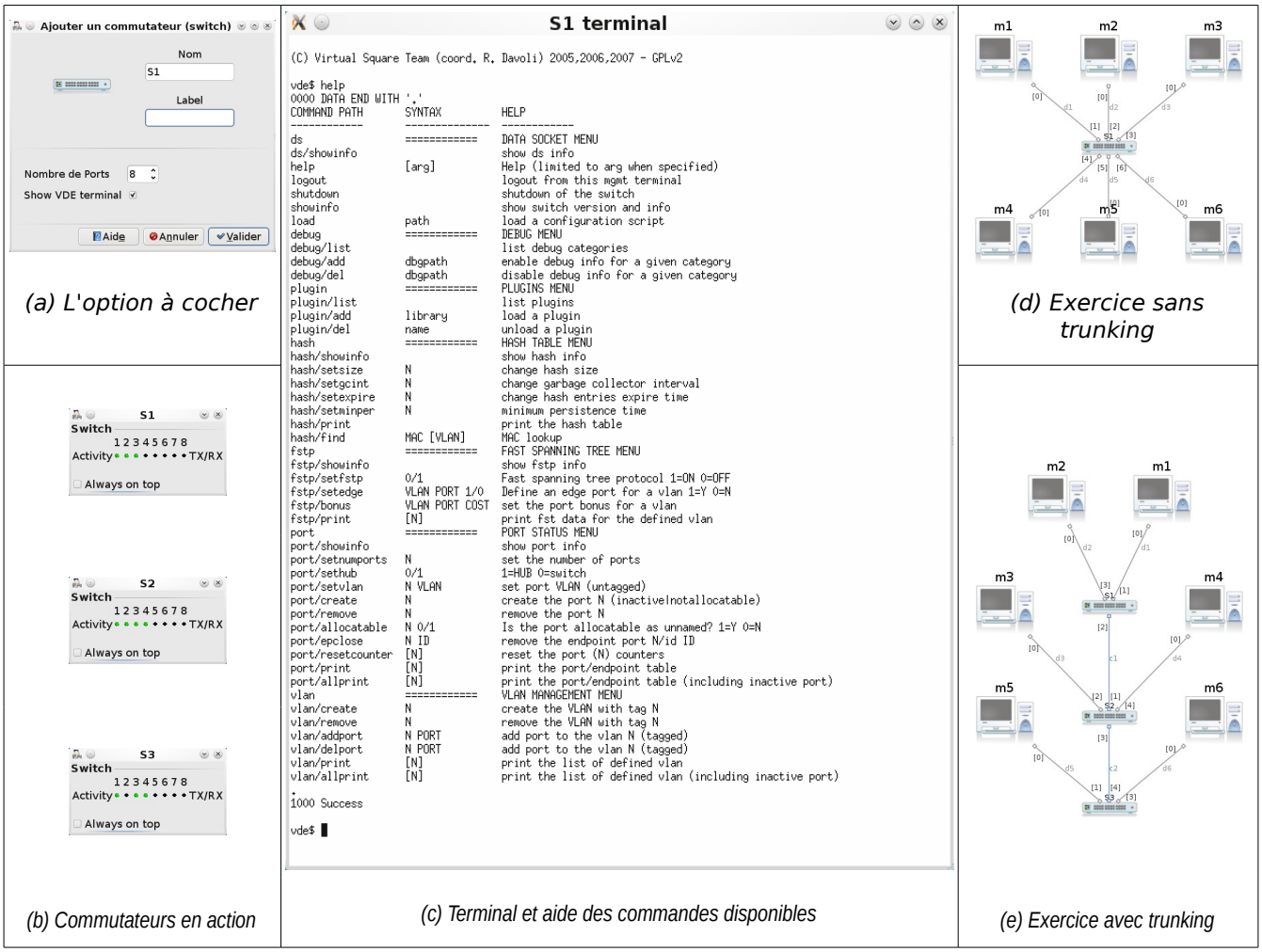


Figure 2 – *Couche Liaison*

C'est le moment d'apprendre que, pour obtenir une certaine étanchéité, divers commutateurs du marché peuvent être configurés pour découper un réseau physique en plusieurs réseaux logiques (VLAN), et cela en fonction des ports utilisés (type 1), des adresses MAC utilisées (type 2), ou des adresses IP utilisées (type 3). Et que le composant commutateur de Marionnet permet de simuler exclusivement les VLANs de type 1. Plus précisément, Marionnet ne réalise pas directement cette simulation mais pilote (lance et contrôle comme processus externe) un utilitaire qui implémente seulement, à l'heure actuelle, ce type de VLAN (`vde_switch` du projet V*irtualsquare*). Pour activer ces fonctionnalités, l'utilisateur doit au préalable demander un terminal de configuration au moment de l'ajout d'un nouveau commutateur ou au moment d'une modification (menu *Propriétés*) d'un commutateur existant (modification qui est permise seulement si l'appareil est éteint). À cet effet, il doit cocher la case " Show VDE terminal ", comme illustré dans la Figure 2.a, pour voir apparaître le terminal (Figure 2.c) en plus de la fenêtre de diodes habituelle (Figure 2.b). Ce terminal permet la configuration des ports du commutateur.

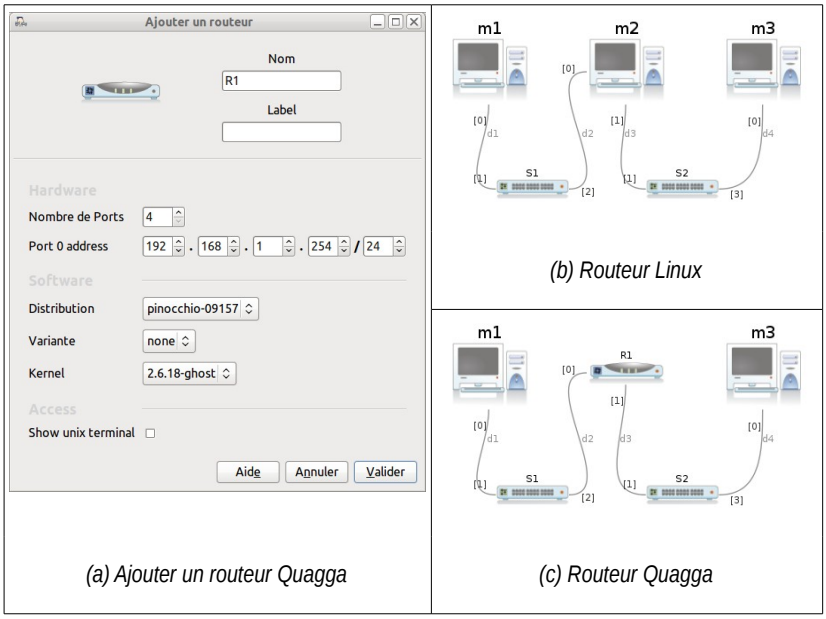
Deux exercices, d'une difficulté croissante, peuvent être réalisés à ce stade. Le premier correspond au cas simple où l'ensemble des ports physiques d'un *unique* commutateur est partitionné (Figure 2.d). Nul besoin du *trunking* et du protocole IEEE 802.1Q associé. Les commandes utiles seront, d'une part, `vlan/create`, permettant de créer un VLAN en lui affectant une étiquette numérique (le tag), et, d'autre part, `port/setvlan`, permettant d'associer un port au VLAN dénoté par son étiquette. Le second exercice porte sur une situation plus courante : *plusieurs* équipements d'interconnexion (commutateurs et/ou concentrateurs) sont reliés entre eux par des câbles croisés (en cas de ports n'ayant pas la fonctionnalité Auto MDI-X, ce qui est le cas des commutateurs simulés dans Marionnet). Dans ce cas, les ports d'interconnexion entre commutateurs constituant les liens « trunk » devront être « taggés » (dans la terminologie de `vde_switch`) en utilisant la commande `vlan/addport`. Il s'agit de comprendre que le but est ici essentiellement économique : partager une seule connexion (une seule paire de ports physiques et un seul câble) pour l'ensemble des VLANs définis. Mais, le prix d'une telle simplicité et économie au niveau physique est cependant payé avec un niveau d'encapsulation supplémentaire entre la couche Ethernet 802.3 et la couche supérieure. L'utilisateur pourra le vérifier par la technique du répéteur interposé (Figure 1.c) :

l'analyse de trames montre que les trames IEEE 802.3 circulant sur les câbles croisés transportent maintenant des trames du protocole IEEE 802.1Q (annoncées par le type 0x8100) qui, à leur tour, transportent IP, ARP ou autre.

2.3 Spanning Tree Protocol (STP)

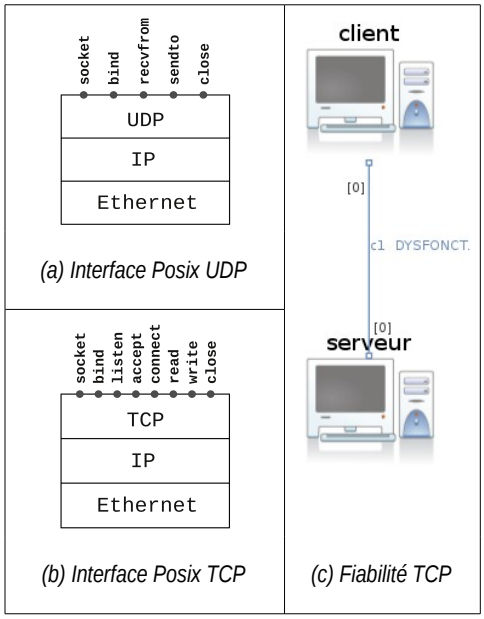
Dans une baie de brassage, plusieurs commutateurs peuvent être reliés de façon à créer éventuellement des boucles de liens par souci de redondance ou par erreur. Le protocole STP (ou FSTP), utilisé par les commutateurs, intervient alors pour « couper » un ou plusieurs liens de façon à éviter que les trames circulent en boucle d'un commutateur à l'autre. Le graphe des connexions devient ainsi un arbre. Puisque les commutateurs utilisés dans Marionnet (les `vde_switch`) implémentent ce protocole, il est possible d'observer l'échange de trames liées à ce protocole à des fins pédagogiques, par exemple en utilisant, une fois de plus, la technique du répéteur interposé (Figure 1.c). À travers le terminal, il est par ailleurs possible d'interroger le commutateur sur sa connaissance actuelle des routes (`fstp/showinfo`) et paramétrer le protocole en présence de VLANs (`fstp/setedge` et `fstp/setbonus`).

3 Couche Internet (~ OSI 3)



(a) Ajouter un routeur Quagga — (b) Routeur Linux — (c) Routeur Quagga

Figure 3 – *Couche Internet (IP)*



(a) Interface Posix UDP — (b) Interface Posix TCP — (c) Fiabilité TCP

Figure 4 – *Couche Transport*

3.1 Routeur Linux

La Figure 3.b illustre un plan réseau très simple pour débuter le travail sur cette couche. En effet, pour découvrir le routage internet, l'utilisateur construit un réseau composé de trois machines, dont une (m2), avec ses deux cartes réseaux, fera office de « routeur du pauvre ». Une fois la configuration des adresses réalisée, il va de soi qu'il faut considérer les tables de routage. Mais avant cela, on peut se préoccuper du fonctionnement du protocole ARP, partie intégrante de IP, nécessaire à la résolution adresse IPv4 → adresse MAC. Le cache ARP des machines Linux est manipulable par la commande `arp`, qui permet d'ajouter des entrées statiques (permanentes) ou encore de supprimer des entrées. Un outil tel que `arp-sk`, par exemple, peut être utilisé pour introduire la notion de ARP-spoofing lorsque l'on s'inquiète de la sécurité.

Les tables de routage sont riches d'exercices (et d'erreurs). Une fois les routes par défaut positionnées sur les machines d'extrémité, les étudiants découvrent que cela ne marche toujours pas car le routage (*IP forwarding*) n'est pas activé par défaut sur la machine faisant office de routeur. Ensuite on peut jouer sur les erreurs volontaires de configuration d'une chaîne de routeurs capables de se « renvoyer la balle » de façon quasi-infinie. Évidemment, grâce au champ TTL, le bouclage n'est pas infini...

L'expérimentation peut à loisir continuer sur la notion de CIDR en agrégeant des routes dans un réseau complexe. On peut aussi jouer sur le MTU (*Maximum Transmit Unit*) qui est la taille maximale des paquets IP que supporte la couche inférieure (1500 octets par défaut pour *Ethernet*). La commande `ifconfig` permet de réduire cette valeur et d'observer une multiplication des fragments IP. On peut alors aussi jouer avec la commande `tracepath` pour découvrir le PMTU (Path Maximum Transmit Unit).

3.2 Routeur Quagga

Marionnet rend disponible un composant « boîtier routeur » spécialisé dans le routage, aussi bien *statique* que *dynamique*. Lorsqu'un tel routeur est introduit dans le réseau, l'utilisateur choisit une adresse IP pré-déterminée qui permettra l'accès en *telnet* à l'équipement (voir Figure 3.a, champ « Port 0 address »). Pour réaliser ce composant, Marionnet utilise le logiciel Quagga [7] embarqué dans une machine virtuelle. L'icône représentant ce composant est un boîtier similaire à ceux des hubs et switchs (voir Figure 3.c, où le routeur est R1). L'utilisateur se connecte par le port 0 avec la commande `telnet` sous GNU/Linux (depuis la machine m1) et peut ensuite configurer le routeur avec des commandes d'inspiration IOS CISCO (`enable`, `configure terminal`, `interface`, `ip address`...). Pour des exercices complexes et spécifiques au routage avec ce type de composant, voir [8].

3.3 Routage dynamique.

Quagga peut émuler un routeur BGP, OSPF, ISIS, RIP et RIPng toujours avec des commandes d'inspiration IOS CISCO. Dynamips-Dynagen [9] pourrait être intégré à terme à Marionnet et nous permettre de faire tourner un vrai IOS CISCO dans une machine virtuelle. L'attribution des adresses IP peut être faite de plusieurs façons. Pour les machines virtuelles, par le biais de `ifconfig` : à chaud et en mémoire vive uniquement, ou bien dans le fichier de configuration dépendant de la distribution Linux concernée. Marionnet propose une troisième solution. Par le biais de l'interface graphique, en utilisant l'onglet « Interface », l'utilisateur peut fixer des adresses IP aux interfaces réseaux et aux ports des routeurs. Cela se traduit au lancement des machine virtuelles, en fournissant une ligne de commande particulière à leur noyau, que des scripts de démarrage (Init System V) spécifiques s'occuperont de lire. L'installation de ces scripts est ce que l'on appelle la « marionnetisation » d'une machine virtuelle.

DHCP. D'une manière plus classique, l'utilisateur peut monter très rapidement un serveur DHCP en utilisant par exemple le démon `dhcpd`. L'enseignant peut fournir dans l'énoncé de son TP les 3-4 lignes d'une configuration basique du serveur. On enrichira éventuellement l'exercice d'un relais DHCP en utilisant `dhcprelay`.

3.4 Préparer le passage à IPv6

Le 3 février 2011, l'IANA a alloué ses derniers blocs d'adresses IPv4 libres. Il va être de plus en plus urgent de passer à IPv6. Marionnet peut aider à monter des TP pour l'apprentissage et l'expérimentation pratiquement « grandeur nature » de IPv6. Nous présentons trois types d'exercices qui nous paraissent formateurs à ce sujet.

Serveur double pile. Une étape initiale peut consister à avoir un client IPv6, un client IPv4 et un serveur S équipé de la double pile IPv6 + IPv4. On donne donc une adresse IPv6 et une adresse IPv4 à la même interface eth0 de S. Les clients peuvent tester par `ping` (et `ping6` pour IPv6) dans un premier temps, puis peuvent se connecter sans problème à un serveur web (par exemple `apache`) activé pour l'occasion sur S (voir la discussion sur le protocole HTTP dans la rubrique Applications). À présent, l'utilisateur peut configurer le serveur pour avoir *deux* « virtualhost » de façon à répondre différemment selon le protocole de communication, IPv4 ou IPv6, choisi par le client.

Routage. Il est facile pour un étudiant ayant déjà réalisé un exercice de routage IPv4, d'explorer le routage statique en IPv6. Le travail est, en effet, similaire à IPv4, avec des petites modifications telle l'activation du routage qui passe par `sysctl -w net.ipv6.conf.all.forwarding=1`.

Auto-configuration. Une autre étape (avant une future étape de « tunneling IPv6 dans IPv4 » que nous n'avons pas encore finalisée et testée) est de jouer avec l'auto-configuration *sans état* (à ne pas confondre avec DHCPv6 qui est *avec état*). Pour ce faire, l'utilisateur peut reprendre le schéma du routage statique (Figure 3.b), sans configurer les adresses IP, et installer le paquetage `radvd` (*Router Advertisement Daemon*) sur le routeur Linux. On profitera à cette occasion du « Gateway » pour accéder à Internet. L'utilisateur pourra alors configurer ce démon et le lancer, puis observer avec `wireshark` l'auto-configuration se faire très simplement et rapidement. L'utilisateur découvrira ainsi l'utilisation par IPv6 de la *multi-diffusion* (*multicast*) en lieu et place de la *diffusion* (*broadcast*).

4 Couche Transport (OSI 4)

Le niveau Transport peut être abordé par une simple analyse de trames. D'une part, cela permet de réviser le format des datagrammes UDP et TCP exposés en cours magistral. D'autre part, l'échange de trames générées par TCP permet de réviser son fonctionnement, notamment la célèbre « poignée de main ». Grâce aux dysfonctionnements simulés, Marionnet permet d'aller plus loin dans cette analyse du comportement : on peut « stresser » TCP pour voir jusqu'où il résistera aux pannes matérielles. Enfin, un troisième type d'exercice qui semble fondamental, à notre avis, pour un traitement exhaustif de la couche Transport, consiste à prendre connaissance des services offerts par UDP et TCP.

Analyse des datagrammes TCP et UDP. Considérons par exemple le réseau de la Figure 1.e. L'analyse de la couche transport par `wireshark` permet d'insister sur la connexion en trois phases de TCP et de sa fermeture de connexion en quatre phases. En liaison avec `iptables`, on peut filtrer la première phase d'ouverture de connexion (seul le drapeau SYN est activé), ce qui permet un filtrage par rapport au sens d'ouverture de la connexion. On peut plus simplement s'inquiéter de n'autoriser que certains ports. Concernant UDP, au delà de l'analyse du format des datagrammes, le service `echo` (fourni par `inetd`/`xinetd`) permet de tester des « tempêtes de diffusions »...

Tester la fiabilité de TCP. Si la plupart des cas d'étude présentés dans cet article semblent impraticables dans la réalité pour des raisons économiques (pas assez de matériel par étudiant) ou de complexité de mise en œuvre (pas assez de temps par séance), voilà un exercice qui serait tout simplement impossible sans la virtualisation, à moins peut-être de « casser » du matériel ! Un réseau comme celui de la Figure 4.c. correspond à la réalisation la plus simple possible d'une liaison logique entre deux machines. Sur le câble croisé modélisant cette liaison, nous allons définir des dysfonctionnements, par exemple des pertes de paquets ou un pourcentage de bits inversés, de plus en plus graves. En lançant une paire de processus client-serveur communiquant sur TCP (deux `netcat`, ou un client web (p.e. `lynx`) et un serveur web (p.e. `apache`)), l'utilisateur constatera la forte résistance de TCP face à ces dysfonctionnements (60-70 % de pertes dans une direction n'empêchera pas le service d'être rendu).

Initiation client-serveur. Prendre connaissance des API système des couches UDP et TCP permet de comprendre clairement comment les protocoles des couches supérieures sont construits. C'est un point crucial pour comprendre la suite, c'est-à-dire la couche Application. Dans un premier temps, il s'agit de programmer des clients et serveurs simples. On se contente de comprendre de quels outils on dispose pour piloter la couche UDP (Figure 4.a) et de quels autres pour piloter TCP (Figure 4.b) : création d'un socket (`socket`, `bind`), envoi de datagramme (`sendto`, `send`), réception (`recvfrom`, `read`), connexion (`listen`, `accept`, `connect`), clôture (`close`). À cette fin, l'enseignant peut fournir du code déjà écrit, totalement ou en partie, et demander aux étudiants de compiler[7] sur les machines virtuelles puis de tester en capturant les trames, ce qui leur permettra de faire le lien entre les événements et les primitives.

5 Couche Application (OSI 5 à 7)

D'une manière générale, Marionnet peut servir de plate-forme d'entraînement et de test de services réseaux divers et variés. Les administrateurs débutants peuvent tester divers serveurs avec divers clients avant une éventuelle mise en production réelle. Nous traitons ici les deux cas fondamentaux DNS et HTTP.

DNS. Le protocole DNS est une vraie école de rigueur ! Les utilisateurs peuvent aborder la configuration de *bind 9*. Dans une première étape, il peut être utile de sensibiliser les utilisateurs à la nécessité de DNS par l'usage de la résolution statique `/etc/hosts` et un premier réseau très simple : un client relié à un serveur par un concentrateur (hub). On peut améliorer le schéma en introduisant un serveur DNS cache ou encore un serveur DNS esclave (il est préférable alors de prévoir des délais de refresh assez faibles). Plus riche encore, on peut aussi illustrer la délégation d'autorité d'un serveur DNS maître à des serveurs de sous-domaines, Marionnet supportant la simulation de réseaux complexes.

HTTP. Le serveur Apache est le plus répandu non seulement sous Linux mais aussi sur internet en général (*Web Server Survey* de Netcraft de mars 2011 précise qu'il représente environ 60 % des serveurs web). Toutes les variations sur les configurations de Apache sont possibles. L'analyseur de trames peut permettre de mettre le doigt sur le fonctionnement des cookies et des exercices simples de création de scripts CGI peuvent permettre de les positionner explicitement. De même, l'utilisation de formulaires avec des mots de passe à saisir est instructive. En utilisant l'authentification de type Basic, les mots de passe seront codés simplement en Base64 et pourront être éventés par l'analyseur de trames. Par ailleurs, Apache permet de virtualiser plusieurs serveurs en un par le biais de la directive VirtualHost, qui est très souple et permet aussi de scinder de façon pratique les traces de consultation et d'erreurs.

6 Récapitulatif multi-couches

Un schéma comme celui de Figure 1.a peut être utilisé dans un scénario final ou comme projet permettant de réviser plusieurs sujets à plusieurs niveaux. Deux réseaux locaux privés (LAN1={m1,m2} et LAN2={m3,m4}), équipés d'éventuels services DHCP locaux ou en relais, doivent être protégés par un « firewall » d'éventuelles attaques. Le pare-feu doit agir en tant que routeur en faisant du « masquerading » (SNAT) pour que les machines puissent accéder à Internet (à travers la passerelle G1). Dans l'autre sens, il doit déléguer aux machines (DNAT) certains services offerts à l'extérieur (par ex. HTTP, SSH, DNS,..). La combinaison de SNAT, DNAT et filtrage amène alors l'utilisateur vers des notions de routage et filtrage avancés, basées sur l'état des connexions (NEW, ESTABLISHED, RELATED).

[7] Les machines virtuelles disposent en général, à l'exception de la série « pinocchio », du compilateur gcc de GNU et des librairies nécessaires.

7 Perspectives et conclusions

Nous avons tenté de faire un tour d'horizon des potentialités de Marionnet en suivant le schéma en couches de TCP/IP. Force est de constater que seules nos connaissances et notre imagination nous ont limité. Malgré les défauts que nous chercherons à réduire, Marionnet nous offre déjà une panoplie de solutions pédagogiques et d'expérimentations techniques remarquables en quantité et qualité. Le logiciel tient très bien la charge lorsque les réseaux à simuler sont complexes (nous avons testé des réseaux de l'ordre de la dizaine de machines avec une dizaine d'équipements d'interconnexion). Marionnet étant multi-processus, il s'exécute de façon très efficace sur une machine hôte multi-cœurs.

À l'IUT de Villetaneuse, Marionnet est d'ores et déjà intégré à l'Environnement Numérique de Travail (ENT), ce qui permet aux étudiants de s'entraîner à distance sur des réseaux fournis en tout ou partie par les enseignants.

Dans la longue liste des perspectives de développement, un point qui nous paraît prioritaire est celui d'automatiser la création de machines virtuelles et de noyaux UML « marionnetisés ». UML est un outil puissant et Marionnet en tire parti mais, par exemple, nous ne pouvons pas avoir de machines *Windows* dans le réseau virtuel. À terme, pour étendre le spectre des systèmes d'exploitation invités, nous pensons donc intégrer le pilotage de machines virtuelles lancées par VirtualBox ou QEMU. L'interface graphique, quant à elle, est encore un peu trop statique. Nous espérons pouvoir la rendre plus conviviale en permettant, notamment, de lancer ou d'arrêter des nœuds du réseau par un simple clic sur leur icône.

8 Bibliographie

[1] http://user-mode-linux.sourceforge.net

[2] http://www.virtualsquare.org

[3] http://www.graphviz.org

[4] http://tools.ietf.org/html/rfc1122

[5] Jean-Vincent Loddo. *Marionnet : un logiciel graphique pour l'apprentissage et l'enseignement des réseaux locaux d'ordinateurs*. Premier Workshop pédagogique "Réseaux & Télécoms", Saint-Pierre de la Réunion, 2007.

[6] Jean-Vincent Loddo, Luca Saiu. *Marionnet: a virtual network laboratory and simulation tool*. SimulationWorks, Marseille (France), 2008.

[7] http://www.quagga.net

[8] Rushed Kanawati, Jean-Vincent Loddo. *Le routage IP statique avec Marionnet : retour d'expérience*. Colloque sur l'enseignement des Technologies et des Sciences de l'Information et des Systèmes (CETSIS), Grenoble, 2010.

[9] http://www.dynagen.org