الجمـــــهوريـــــة الجـــزائريـــــة الديـــمقراطيـــــــــــــة الشـــعبيـــــــــــة
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعـــليـــم الـعـالـــي و الـبـحـث الـعـلـــمي
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
جـامـعـــة عـمّـــار ثـلـيـــجـــي بالأغـــــواط
UNIVERSITE AMAR TELIDJI LAGHOUAT
كـــلـيـــــة الـعـــلـــــوم
FACULTE DES SCIENCES

DEPARTEMENT D'INFORMATIQUE

Mémoire de MASTER

Domaine : *Mathématiques et Informatique*
Filière : *Informatique*
Option : *Réseaux, Systèmes et Applications Réparties*

Par:
Chehrouri abdelkader
Salmi mohamed

THEME

La virtualisation avec Netkit

Soutenu publiquement le 12-06-2017 devant le jury composé de:

Mlle ABDELHAFIDI Zohra	MC-B	Président
Mr BENDOUMA Taher	MC-B	Examinateur
Mlle BELABBACI Amel	MA-A	Encadreur

Année Universitaire 2016/2017

Remerciement

Notre remerciement s’adresse en premier lieu à Allah le tout puissant pour la volonté,
la santé et la patience qu'il nous a donné durant toutes ces longues années.

Nous tenons également à remercier Mlle Belabbaci Amel notre encadreur
qui a apporté une aide précieuse.

Nos remerciements vont aussi à tous nos enseignants de département
d’Informatique « UNIVERSITE AMAR TELIDJI LAGHOUAT »
qui ont contribué à notre formation durant nos années des études.

À tous les membres du jury qui nous font l’honneur de juger notre travail…

Enfin, nous tenons à exprimer notre très reconnaissance à tous nos amis et nos collègues pour le soutien moral et matériel...

Dédicace

Je dédie mon travail au

le symbole de la tendresse, qui s'est sacrifiée pour mon bonheur et ma réussite, à ma mère ...

A mon père, l’école de mon enfance, qui a été mon ombre durant

toutes les années des études, et qui a veillé tout au long de ma vie

à m’encourager, à me donner l'aide et à me protéger.

Mes frères et sœurs

«Pour leur soutien et leur aide permanente».

Pour toute ma famille.

Mes professeurs

« Pour leur encouragement et leurs précieux conseils, ceux qu’ils n'ont épargné aucun effort pour

m’accompagner durant tout mon étude »

A tous mes amis.

Salmi Med

Dédicace

Je dédie ce modeste travail

A ceux qui m'ont donné la vie

A ceux qui sont la source de mon inspiration et mon

courage

A ma très chère mère

A mon cher père

A mes frères et mon sœur

A mes chers cousins et cousines

A toute ma famille

A tous mes amis

Chehrouri Aek

Résumé

Un émulateur est un environnement capable de reproduire de près les fonctionnalités d'un système réel. Les émulateurs sont très utiles pour effectuer des expériences susceptibles de compromettre le fonctionnement du système cible ou tout simplement lorsque le système réel lui-même n'est pas disponible. Cela est vrai en particulier pour les réseaux informatiques, où les configurations de périphériques réseau doivent souvent être testées avant d'être déployées.

Dans ce travail, nous présentons la virtualisation avec Netkit, un émulateur léger basé sur User-Mode Linux. En plus d'être un instrument efficace pour soutenir l'enseignement des réseaux informatiques, Netkit s'est également avéré utile pour tester la configuration des réseaux à grande échelle du monde réel. Netkit s'installe et s'exécute complètement dans l'espace utilisateur et fournit aux utilisateurs un environnement familier composé d'une boîte Linux basée sur Debian et d'un logiciel de routage et d'outils de réseaux bien connus.

Mots clefs : Virtualisation, Emulation, Netkit, Uml.

ملخص

المحاكي هو بيئة تمكن المستخدم من أداء نفس وظائف النظام الحقيقي إذ يعتبر مفيد جدا لأداء التجارب التي قد تاثر سلبا على وظائف النظام الحقيقي أو ببساطة عندما يكون النظام الحقيقي نفسه غير متوفر. وهذا ما ينطبق على شبكات الحاسوب، حيث غالبا ما تحتاج إعدادات أجهزة الشبكة إلى اختبار قبل استخدامها

في هذا العمل، قدمنا مفهوم الافتراضية و قمنا بدراسة اداة المحاكاة نتكيت،هو محاكي لا يستهلك موارد الجهاز بشكل كبير و يعتمد بالأساس على وضع مستخدم لينكس . إلى جانب كونه أداة يدعم تدريس شبكات الكمبيوتر، وقد أثبت أيضا انه مفيد في اختبار تكوين شبكات العالم الحقيقي على نطاق واسع

Sommaire

Liste des figures

Introduction générale

La virtualisation est une technologie qui permet d'exécuter simultanément plusieurs systèmes d'exploitation et plusieurs applications sur un même ordinateur.

Il existe plusieurs techniques de virtualisation, ces techniques visent à reproduire de près les caractéristiques et le comportement des dispositifs réels. Elles sont utilisées par les chercheurs pour mener leurs expérimentations, comme elles sont utilisées pour enseigner, et aider à comprendre le fonctionnement du réseaux informatique. Les environnements de virtualisation sont souvent constitués d'un logiciel ou plate-forme matérielle qui permet d'exécuter des programmes qui pourraient être utilisés sur des appareils réels.

Ce travail présente les concepts fondamentaux des environnements d'émulation et présente Netkit, un logiciel basé sur l'UML (User Mode Linux). Netkit est le résultat du travail conjoint de plusieurs personnes. Le projet est réalisé par le Département d'informatique de l'université de Rome3.

Après avoir décrit l'architecture de Netkit, nous montrons comment il est possible de préparer et d'exécuter facilement des expériences de réseau. Un exemple d'étude de cas est également présenté pour montrer les capacités de Netkit.

Le but de ce mémoire et d'étudier la virtualisation avec Netkit, il est divisé en trois chapitres.

Le premier chapitre est la virtualisation.

Le second chapitre est Netkit et la virtualisation.

Le troisième chapitre est étude d'un réseau d'entreprise avec Netkit.

Chapitre I:

La virtualisation

1. Introduction

La virtualisation occupe une place indispensable dans le domaine de l'informatique car elle permet aux utilisateurs de solliciter leurs ressources matérielles mieux qu'investir de nouveaux matériels.

Dans ce chapitre, nous allons définir la virtualisation, et déterminer le concept d'une machine virtuelle et de l'hyperviseur, puis les différentes techniques de la virtualisation.

2. Principe de la virtualisation

2.1. Historique

La virtualisation remonte aux années 60 dans les travaux de centre de recherche IBM de *Grenoble*. A l'époque, c'est la firme IBM qui a créé le premier système de virtualisation de serveur. Dans ce contexte, l'informatique est peu présente et les rares sociétés qui possèdent des systèmes informatiques sont équipées de gros calculateurs, les Mainframes. Déjà à cette époque, les soucis d'optimisation des ressources matérielles d'une machine se posent. En effet, les supers calculateurs sont parfois sous utilisés. IBM développe alors les produits CP/CMS puis VM/CMS (Virtual Machine / Conversational Monitor System), des systèmes de virtualisation serveurs. Par la suite, les technologies propriétaires pour virtualiser les OS des mainframes sont mise en œuvre (HP, Sun). Au cours des années 80-90, il apparaît l'architecture x86 et les PC se déploient auprès d'un grand nombre d'utilisateurs. Le besoin de virtualiser pour optimiser les machines se fait moins sentir. Mais, dans les années 90-2000, *VMware* réussit à virtualiser un poste x86. Ceci ouvre la porte à plus de possibilité et relance l'envie pour les sociétés informatiques de développer de nouvelles fonctionnalités pour optimiser et pour offrir plus de flexibilité. A nos jours, la virtualisation est très connue. On entend parler de virtualisation de serveur, de Virtual box, de baremetal, mais aussi de virtualisation de poste de travail, de VDI, et de virtualisation dans les jeux-vidéos avec les émulateurs. En 2012, trois grandes sociétés se partagent le marché de la virtualisation en entreprise : VMware qui est le leader Citrix, très fort dans la virtualisation de poste de travail Microsoft, qui s'aligne sur la concurrence [1].

2.2. Définition

La virtualisation [2] est «l'ensemble des technologies matérielles et/ou logiciels qui permettent de faire fonctionner sur une seule machine plusieurs systèmes d'exploitation et/ou plusieurs applications, séparément les uns des autres, comme s'ils fonctionnaient sur des machines physiques distinctes » (Figure I.1) . Ainsi, la virtualisation permet de conserver un environnement partagé entre les différentes machines virtuelles de la même façon qu'entre des machines physiques. Cette méthode partage les ressources matérielles d'une machine entre plusieurs systèmes d'exploitation

en facilitant l’accès matériel de manière à ce que cette gestion soit transparente du point de vue des systèmes d’exploitation virtualisés. La virtualisation est une couche d'abstraction qui découple le système d'exploitation du matériel afin de délivrer une meilleure utilisation et flexibilité des ressources de traitement [3].

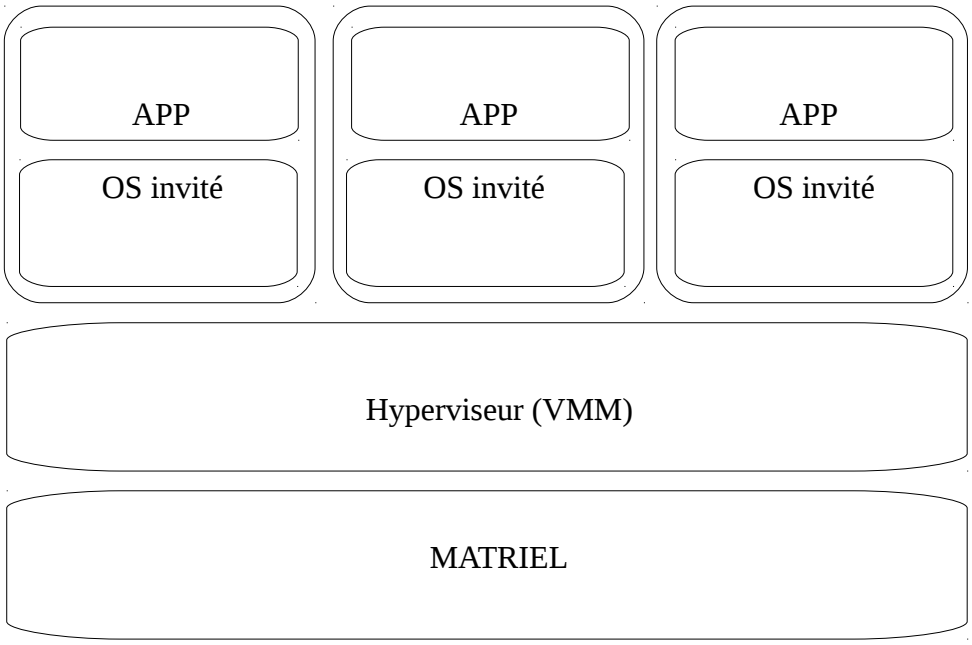


Figure I.1: *Les différentes couches d'un serveur virtualisé.*

3. Machine virtuelle (VM - Virtual Machine)

Chaque système informatique virtuel correspond à une « machine virtuelle » (VM), c’est-à-dire à un conteneur de logiciels totalement isolé, et doté d’un système d’exploitation et d’applications [4]. Le sens original de Machine Virtuelle est la création de plusieurs environnements d’exécution sur un seul ordinateur [5]. Est une illusion d'un machine réel créée par un logiciel d'émulation, ce logiciel émule la présence de ressources matérielles et logicielles telles que la mémoire, le processeur, le disque dur, permettant d'exécuter des programmes dans les mêmes conditions que celles de la machine réel.

4. Hyperviseur (VMM Virtual Machine Monitor)

Un hyperviseur ou contrôleur des machines virtuelles est une plate-forme de virtualisation, il s’agit d’un logiciel qui va créer et gérer l’exécution d’un certain nombre de machines virtuelles [6]. Il permet à plusieurs systèmes d'exploitation de travailler sur une même machine physique en même temps. La figure I.2 ci-dessous montre l'hyperviseur et la machine virtuelle:

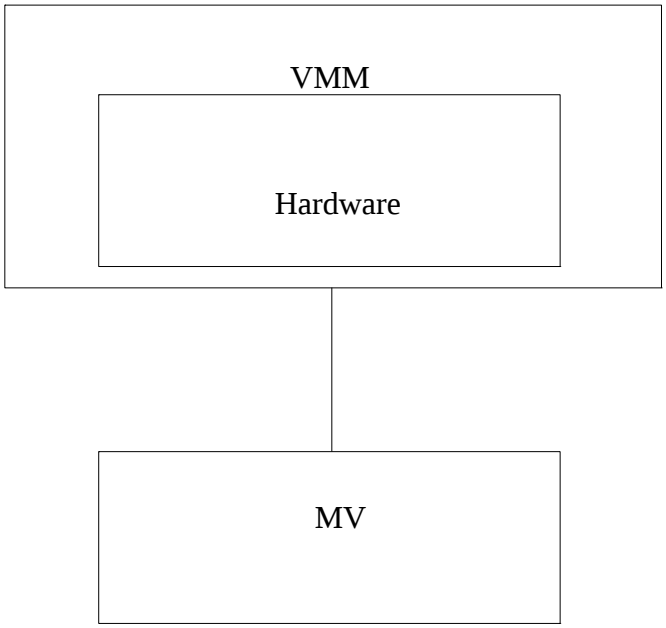


Figure I.2: Le schéma d'hyperviseur et de machine virtuelle[7].

5. Les types des hyperviseurs

5.1. Hyperviseur de type 1

C’est un programme qui s’installe directement sur la plate-forme matérielle et qui permet d’arbitrer les échanges d’entrées-sorties des ressources matérielles (processeur, mémoire RAM, disque dur, etc...) entre les systèmes d’exploitation (OS). Chaque OS est complètement isolé et indépendant des autres systèmes.

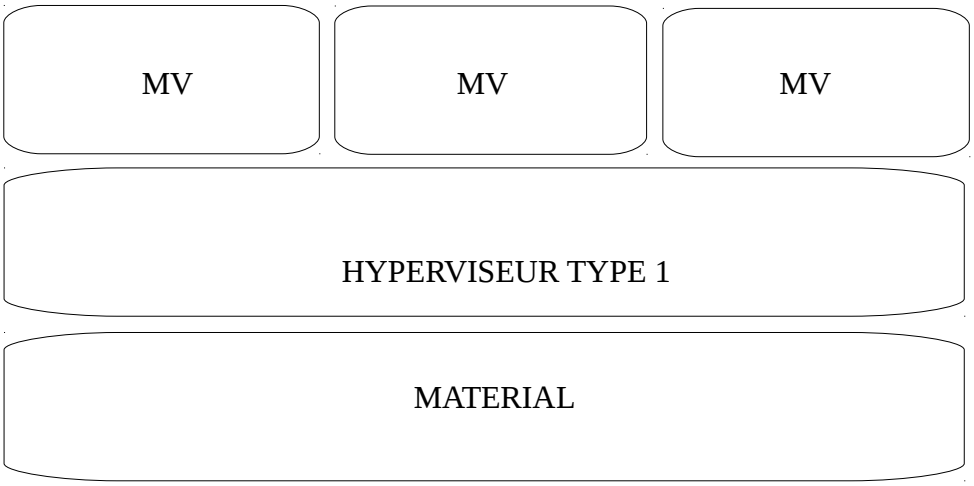


Figure I.3:Hyperviseur de type 1 [8].

5.2. Hyperviseur de Type 2

Est un logiciel de virtualisation des systèmes qui s'installe et s'exécute sur un système d'exploitation déjà présent sur la machine physique. Il fait alors la relation entre les ressources offertes par le système d'exploitation hôte et les systèmes d'exploitation virtualisés.

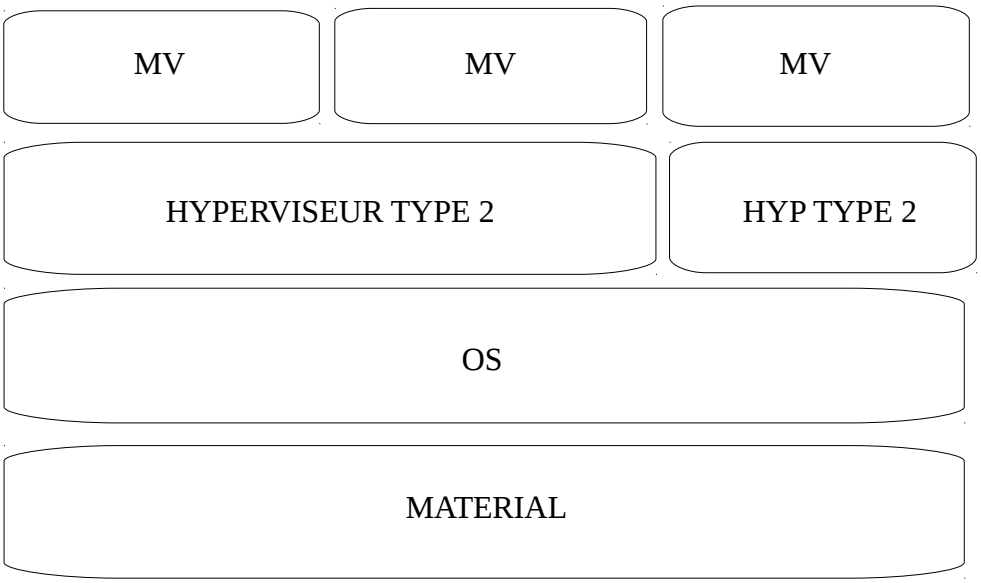


Figure I.4: Hyperviseur de type 2 [8].

6. L' expérimentation dans les environnements réels

L'environnement réel est un environnement qui utilise un support d'infrastructure réel afin de mener des expérimentations. Cette approche assure implicitement un excellent niveau de réalisme, et permet d'obtenir les meilleurs résultats en termes de précisions. L'expérimentation en environnement réel est limitée par le manque de contrôle où il est très difficile de contrôler l'ensemble de paramètres d'une expérience.

Pour réaliser une expérience, le développeur a besoin de modifier les paramètres et les conditions de son environnement expérimental pour atteindre le paramétrage optimale, alors il est nécessaire de reproduire une même expérience, gardant toujours les mêmes conditions pour chaque expérience, et ce qui ne peut pas être assuré dans un test en environnement réel, puisque les conditions peuvent être changées d'un teste à un autre.

Ce type d'approche est très coûteux en terme de temps, ressources et main-d'œuvre, ainsi la réalisation d'une expérience dans un environnement pareil ne permet pas toujours un contrôle précis des conditions d'expérimentation, la reproduction d'une même expérience est quasiment impossible.

7. La simulation

La simulation est la reproduction du comportement dynamique d'un système réel [9]. Les environnements de simulation permettent à l'utilisateur de prévoir le résultat de l'exécution d'un ensemble de Périphériques sur un réseau complexe en utilisant un modèle interne qui est spécifique au simulateur. Avec Le réseau comme une entrée et le résultat (éventuellement un état de réseau) comme une sortie, les simulateurs ne reproduisent pas nécessairement la même séquence d'événements qui aurait lieu dans le système réel, mais Plutôt d'appliquer un ensemble interne de routines de transformation qui amène le réseau à un état final qui est aussi proche que possible de celui vers lequel le système réel évoluerait.

L'inconvénient est que les dispositifs de réseau simulés peuvent avoir des fonctionnalités limitées et leur comportement ne peut pas ressembler.

Plusieurs outils et environnement permettent la mise en œuvre de simulation réseaux existent tel que NS-2, NS-3, OPNeT , OMNeT++ et autres.

8. L’émulation

L'émulation consiste à substituer un élément matériel informatique par un logiciel [10]. les environnements d'émulation visent à reproduire de près les caractéristiques et le comportement des dispositifs. Pour cette raison, ils consistent souvent en une plate-forme logicielle ou matérielle qui permet d'exécuter les mêmes morceaux de logiciel qui seraient utilisés sur les dispositifs réels. Dans un émulateur, le réseau testé subit les mêmes échanges de paquets et les changements d'état qui se produirait dans le monde réel. L'avantage réel de l'approche d'émulation sort quand l'émulateur lui-même est une pièce de logiciel, car cela permet une plus grande souplesse dans la réalisation des tests de réseau. Chaque aspect du réseau peut être influencé et surveillé comme il se pourrait dans un réseau réel. Bien que cela assure une très grande précision, les ressources de calcul nécessaires pour exécuter un émulateur sont typiquement supérieures à ceux disponibles dans le dispositif réel lui-même. Par conséquent, la performance d'un émulé est en général inférieur à celui du réel, ce qui pose souvent des limites à l'évolutivité du taille du réseau émulé.

9. Classification de quelques émulateurs

Plusieurs environnements d’émulation existent pour les réseaux informatiques sous différents systèmes d’exploitation, en licences commerciales ou gratuites, avec de petite ou grande échelle, (en termes de nombre d'entités virtuelles). Le tableau suivant représente quelques émulateurs des réseaux [18].

	L’échelle	Type d’émulation	Diapositif émulé
VirtualBox	Petit	Virtualisation complète	X86
Netkit	Moyen	Noyau mode utilisateur	Linux
VNUML	Moyen	Noyau mode utilisateur	Linux
VINI	Large	Noyau mode utilisateur	Linux
Xen	Moyen	Para-virtualisation	X86
VMware	petit	Virtualisation complète	X86

Tableau I.1: *Classification de quelques émulateurs*

L’échelle: petit, moyen et large, il est décrit a partir de nombre d'entités virtuelles (hôtes, routeurs, commutateurs, ou autres) émulé.

Un émulateur de petite échelle est généralement conçu pour exécuté quelques machines virtuelles, et leurs besoins en ressources peuvent être élevés. Un émulateur de Moyen échelle, permettent généralement à exécuté des centaines de machines virtuelles sur un seul poste de travail. Un émulateur à grande échelle est généralement conçu pour fonctionner sur une architecture distribuée (un groupe de postes de travail répartis géographiquement reliées par un réseau), ce qui permet d'effectuer des expériences très grande.

La diapositive émulée: Spécifie la machine dont les caractéristiques sont reproduites par l'émulateur. Chaque émulateur est démarré sur une plateforme, généralement un poste de travail standard. Qu’exécute un ensemble de logiciel, peut être tourné comme un routeur, un commutateur ou un autre périphérique réseau.

Type d’émulation: Spécifié la technique utilisée pour la virtualisation des ressources.

10. Les techniques de virtualisation

les techniques de virtualisation permettent à l'utilisateur de disposer d'un environnement virtuel qui Peut être exploité pour des essais, des expériences, des mesures. On a présenter les techniques suivantes:

10.1. Para-virtualisation

Para-virtualisation est une technique de virtualisation qui présente une interface logicielle à des machines virtuelles, mais pas identique à celle du matériel sous-jacent. Dans ce type de

virtualisation, l'hyperviseur et le système d'exploitation invité coopèrent. Il vise donc à modifier les systèmes d'exploitation pour qu'ils communiquent avec un hyperviseur au lieu d'une machine physique.

La para-virtualisation, comme le montre la figure I.5, consiste à modifier le noyau du système d'exploitation d'invité pour remplacer les instructions non virtualisables par des hyper-appels (hypercalls) qui communiquent directement avec la couche virtuelle de l'hyperviseur. L'hyperviseur fournit également des interfaces hyper-appels pour d'autres opérations critiques du noyau telles que la gestion de la mémoire, la gestion des interruptions…

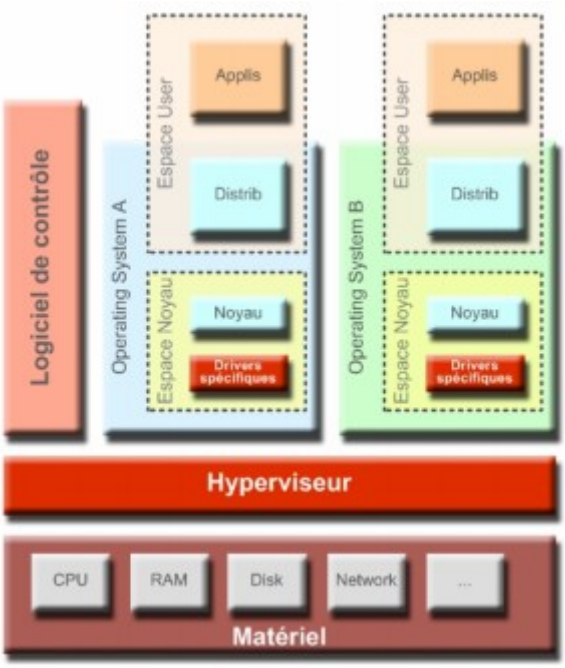


***Figure I.5**: Le technique de para-virtualisation [11].*

10.2. La virtualisation complète

La virtualisation complète est une technique de virtualisation utilisée pour fournir un certain type d'environnement de machine virtuelle, à savoir une simulation complète du matériel sous-jacent. Il permet de faire fonctionner n'importe quel système d'exploitation en tant qu'invité dans une machine virtuelle. L'hyperviseur fait créer un environnement virtuel complet, simulant un ordinateur avec du matériel virtuel.

Dans la virtualisation complète, l'hyperviseur intercepte de manière transparente tous les appels que le système d'exploitation invité peut faire aux ressources matérielles (le jeu d'instructions complet, les opérations d'entrée / sortie, les interruptions, l'accès à la mémoire et tout autre élément utilisé par le logiciel qui fonctionne sur la machine réel).

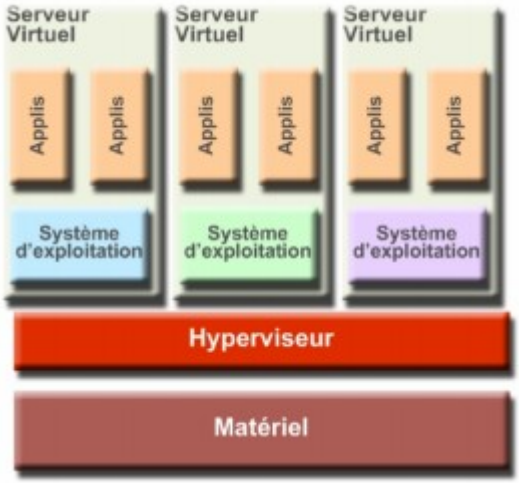


***Figure I.6**: Le technique de virtualisation complète [11].*

10.3. User Mode Linux (UML)

Le noyau est la couche de base d'un système d'exploitation[12] User Mode Linux est un noyau Linux compilé pour être exécuté dans l'espace utilisateur comme un simple programme[13]. Il permet d'exécuter plusieurs machines virtuelles et chaque machine peut être liée par un kernel spécifique [14]. Chaque machine virtuel UML a une console qui permet d'exploiter la machine virtuelle en ligne de commande, et une mémoire, qui est un bout de la mémoire de la machine hôte, et un système de fichiers sauvegardé dans un fichier de système de la machine hébergeuse.

UML lance des logiciels bogués, expérimente de nouveaux noyaux Linux ou de nouvelles distributions, sans risque de détruire la configuration principale du noyau physique.

De nombreuses technologies sont basées sur UML, par exemple Netkit, Marionnet, Cloonix-Net, GINI. On va s'intéresser dans le chapitre suivant sur l'émulateur Netkit.

11. Conclusion

Dans ce chapitre, nous avons parlé des techniques de la virtualisation qui offrent de nombreux avantages pour l'utilisateur. Tout d'abord, elles permettent de mettre en place un système de virtualisation et de réduire son infrastructure matérielle car il devient alors possible d'exécuter divers environnements sur un même système, d'une part. D'autre part, le rendement des machines physiques augmentent grâce à l'utilisation presque maximale de leurs ressources. Enfin, une infrastructure virtualisée est un bon moyen de déployer un environnement de tests lors du développement d'un logiciel critique.

Chapitre II: Netkit et la virtualisation

1. Introduction

Dans ce chapitre, on va présenter Netkit, et découvrir son architecture, ses fonctionnalités, et les étapes pour créer un réseau Virtuel (laboratoire). Enfin, un laboratoire Netkit sera conçu pour comprendre le routage statique et le routage dynamique (protocole RIPv2, deuxième version du protocole RIP).

2. Présentation de Netkit

Netkit est un système d'émulation de réseau informatique qui permet de comprendre le fonctionnement de réseaux informatique. Il permet de déployer plusieurs machines et périphériques de réseau virtuels à part entière qui peuvent être facilement interconnectés pour former un réseau sur une seule machine. Les machines virtuels présentent les mêmes caractéristiques que les vrais, y compris l'interface de configuration.

Netkit est un logiciel basé sur l'UML (User Linux Mode) [15], un port du noyau Linux conçu pour s'exécuter en tant que processus d'espace utilisateur. Les machines virtuelles Netkit sont créées sur le même hôte. Elles sont reliées à des domaines de diffusion virtuels (hub , switch , …) et peuvent communiquer entre elles. Chaque interface réseaux peut être connecté à un domaine de collision (virtuel), et chaque domaine de collision peut être connecté à plusieurs interfaces réseau. Plusieurs machines virtuelles peuvent être exécutées en même temps pour émuler donc un réseau [16]**.** L'environnement Netkit est composé d'un terminal pour chaque machine virtuelle crée, chaque machine virtuelle peut être configurée pour jouer le rôle d'un hôte, d'un routeur, commutateur de manière automatique au travers des labs (laboratoires Netkit) ou manuellement, les modifications peuvent être sauvegardées.

Netkit est un logiciel libre sous licence GPL (GNU Public License) qui est composé de différents scripts permettant le lancement et l'arrêt de machines virtuelles et l'utilisation des outils réseau[17].

3. L'architecture de Netkit

Chaque machine virtuelle possède :

- une console (Fenêtre de terminal).
- Une mémoire indépendante.
- un système de fichiers (qui est stocké dans un fichier dans le filesystem de la machine Linux réelle).
- une ou plusieurs interfaces réseau.

Il est conçu pour une installation et une utilisation faciles et ne nécessite pas de privilèges d'administration pour ces opérations.

L'approche d'émulation adoptée dans Netkit est simple. Fondamentalement, chaque périphérique qui constitue un réseau est implémenté dans Netkit en tant que machine virtuelle. Chaque machine virtuelle possède un ensemble de ressources virtuelles qui sont mappées à des portions des ressources correspondantes sur l'hôte. Comme le montre La figure II.1 [18].

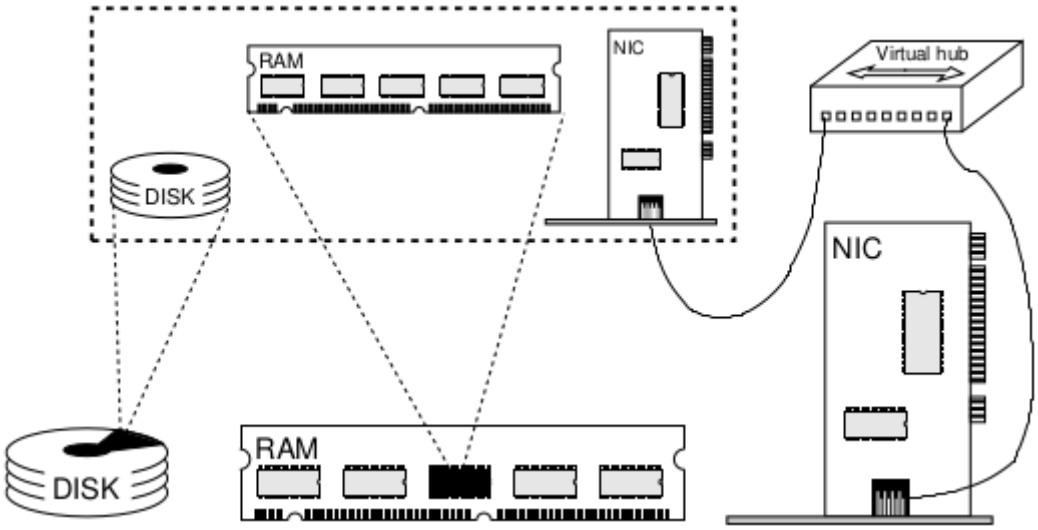


Figure II.1*: Ressources d'une machine virtuelle Netkit.*

Les machines virtuelles sont équipées d'un disque, dont l'image brute est un fichier dans la machine hôte; Ils ont leur propre zone de mémoire, dont la taille peut être établie au démarrage; Et ils peuvent être configurés avec un nombre arbitraire d'interfaces de réseau virtuel qui sont connectées à un concentrateur (hub) virtuel [18].

La zone en pointillés de la Figure II.1 contient des ressources virtualisées, tandis que tout ce qui se trouve en dehors de cette zone est un périphérique ou un processus sur l'hôte. Il est possible d'observer que le hub virtuel est un processus spécial sur l'hôte réel qui diffuse les paquets sur toutes les interfaces connectées. Si on le demande, le hub virtuel peut être connecté à une interface réseau de la machine hôte, afin qu'une machine virtuelle puisse accéder à un réseau externe tel qu'Internet.

Pour faire une distinction claire entre un dispositif émulé et la machine réelle, les mots "hôte" et "invité" sont utilisés pour distinguer le logiciel qui fonctionne sur la machine réelle et logiciel fonctionnant sur la machine virtuelle.

La figure II.2 [18] montre l'architecture de Netkit, constituée des blocs à l'intérieur de la boîte pointillée. chaque bloc représente un morceau de logiciel qui s'exécute sur le bloc en dessous et il s'est contrôlé par les outils sur sa gauche. Les machines virtuelles sont des instances UML qui s'exécutent directement sur le noyau de l'hôte et elles sont gérées par un ensemble de commandes dont les noms sont ltools et vtools. Les machines virtuelles peuvent exécuter des logiciels de routage, ainsi que d'autres outils. Les hubs virtuels sont implémentés en tant que processus s'exécutant sur le noyau hôte.

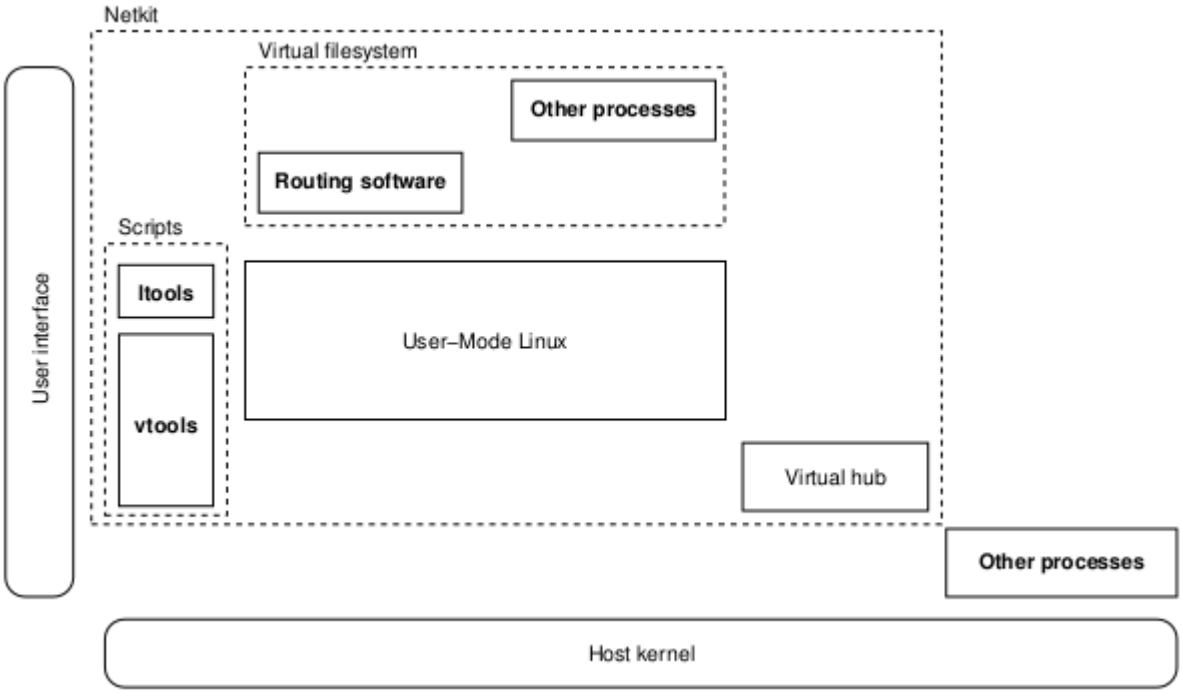


***Figure II.2**: Architecture de Netkit.*

Les blocs avec du texte en gras représentent les seuls composants qui sont présentés à l'utilisateur final. Il est possible d'observer qu'il n'est pas nécessaire pour lui d'interagir directement avec les noyaux UML ou les hubs virtuels. D'autre part, Les machines virtuelles peuvent être contrôlées en utilisant deux interfaces accessibles à l'utilisateur les ltools et les vtools. Les commandes vtools pour créer/gérer une seule machine virtuelle mais les commandes ltools pour gérer un "laboratoire" (avec plusieurs machines virtuelles), alors les commandes ltools exploitent donc les vtools afin de mettre en œuvre leurs fonctionnalités.

4.Technologies de réseaux supportées

Les machines virtuelles Netkit sont connectées par le hub virtuel. En pratique, un hub fonctionne comme une sorte de câble qui relie plusieurs invités. Sachant qu'un invité doit toujours se connecter à un hub virtuel et il ne peut pas être directement connecté à un autre invité. La figure II.3 [18] montre un exemple de fonctionnement d'un réseau Netkit.

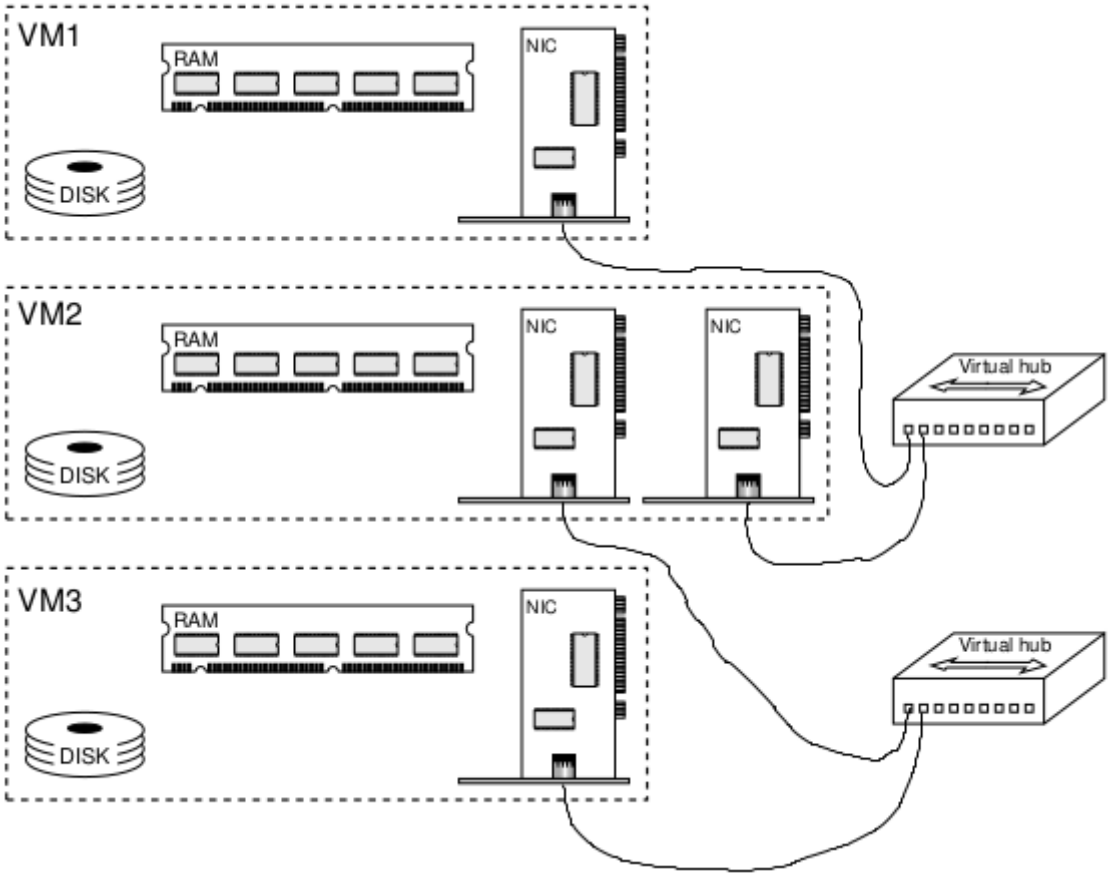


Figure II.3*: Exemple de réseau émulé par Netkit.*

Les VM1, VM2 et VM3 sont des machines virtuelles qui costituent une topologie simple constitué de deux domaines de collision: l'un incluant VM1 et VM2 et l'autre incluant VM2 et VM3. VM2 peut être configuré pour fonctionner comme un commutateur, comme un routeur, comme pare-feu, comme proxy Web, etc.

Dans cette topologie, VM2 achemine le trafic uniquement sur les ports connectés au segment LAN contenant l'hôte de destination. Pour ce faire, VM2 a été équipé de deux Réseau.

Les machines virtuelles peuvent être interconnectées en utilisant des hubs virtuels, à savoir les logiciels exécutés sur l'hôte qui émule les domaines de collision Ethernet. En option, un hub virtuel peut être configuré pour accéder à un réseau externe, par exemple pour connecter des machines virtuelles à Internet. La figure II.4 [18] montre un exemple dans lequel les machines virtuelles vm1 et vm2 sont connectées au concentrateur virtuel A. La machine virtuelle vm2 a deux interfaces de réseau virtuel et est également connectée au concentrateur virtuel B avec vm3. En exécutant un logiciel approprié, vm2 peut permettre à vm1 de communiquer avec vm3. Les machines vm1 et vm2 peuvent également accéder à un réseau externe via le concentrateur virtuel A qui est connecté à L'interface TAP spéciale sur l'hôte.

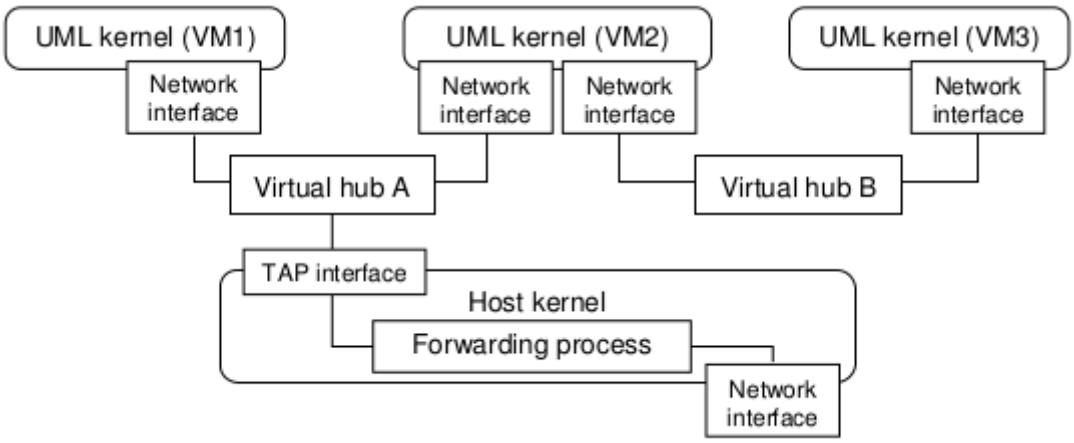


***Figure II.4**: Comment les machines virtuelles Netkit sont connectés, peut être avec une connexion à un réseau externe.*

4.1. L'interface TAP

Un accès internet est parfois indispensable à partir des VMs pour ajouter par exemple des paquets manquants ou pour réaliser des tests. L'accès au réseau physique n'est possible qu'à partir de l'interface physique de l'hôte qui héberge les VMs. Un domaine de collision particulier **TAP** est créé à cet effet. La fonction **TUN/TAP** du noyau Linux est la solution retenue afin de faire communiquer des machines virtuelles avec un hôte. **TUN/TAP** sont des pilotes réseaux qui simulent des interfaces réseaux virtuelles complètement émulées sous forme logicielle. **TAP** simule une interface Ethernet et fonctionne comme une interface de niveau 2 (couche liaison). TUN simule un TUNnel et fonctionne au niveau 3 (couche réseau – IP) du modèle OSI. **TAP** crée un pont entre les interfaces, TUN assure le routage.

4.2. Uml-switch

Les fonctionnalités d'un dispositif de réseau émulé dépendent d'un logiciel installé dans la machine virtuelle qui l'implémente. Comme nous l'avons dit Netkit permet de déployer plusieurs machines et périphériques de réseau virtuels à part entière qui peuvent être facilement interconnectés pour former un réseau, uml_switch est un processus qui implémente un concentrateur (hub) ou commutateur (switch) virtuel [19], est un démon pour la gestion d'un réseau virtuel entre les machines virtuelles , sans connexion à un réseau du système hôte [20]. Chaque machine virtuelle possède un ou plusieurs interface réseau, ces interfaces peuvent être rattachées à un socket UNIX sur l'hôte. À son tour, uml_switch utilise la même socket et établie la connexion avec les machines virtuelles qui sont connectés à ce socket. La figure II.5 [18] montre comment les machines virtuelles sont en réseau.

Netkit utilise uml_switch comme domaines de collision virtuelle et le lancement des processus uml_switch peut être automatiquement selon les besoins de l'utilisateur.

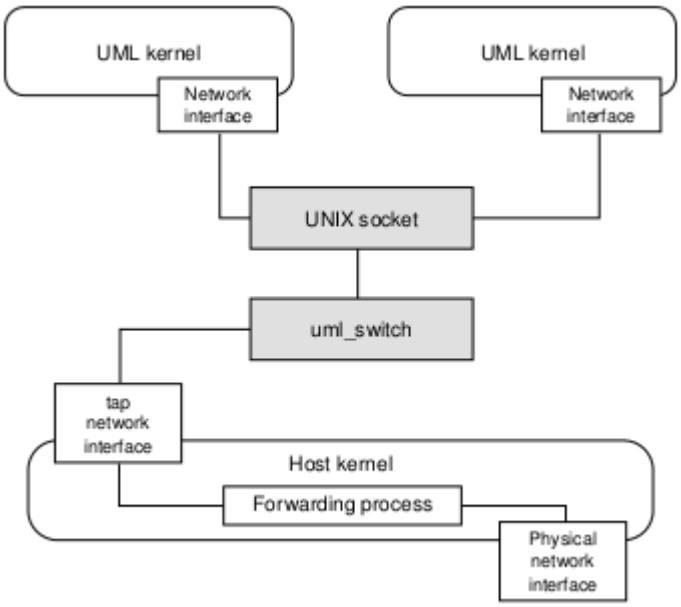


***Figure II.5**: l'uml_switch et UNIX socket*

En raison de son architecture, Netkit peut être utile dans plusieurs contextes par exemple l'enseignement (éducation) qui donne aux étudiants l'impression de ce qui se passe réellement à l'intérieur d'un réseau.

Netkit est doté d'un ensemble d'expériences réseau prêtes à l'emploi qui mettent en œuvre des études de cas des protocoles de routage (TCP, RIP, BGP, etc.) aux services applicatifs (DNS, e-mail, etc.) [21].

Une autre application consiste à préparer des réseaux virtuels pour éviter la nécessité d'effectuer des tests potentiellement nuisibles sur un réseau réel, ainsi que la capacité de réaliser des expériences dans un environnement sûr. Netkit est également une pratique pour tester une configuration avant de la déployer.

5. User Mode Linux

5.1. Le fonctionnement d'UML

Le fonctionnement habituel d'un processus sous Linux se fait par l'intermédiaire du noyau, qui est le lien avec le matériel. L'idée avec User-Mode-Linux, c'est de placer le noyau en tant que simple programme. Les processus virtuels feront donc leurs appels systèmes vers ce programme émulant un noyau [22].

Chaque processus crée par une machine virtuelle est d'abord intercepté et traité par le kernel UML puis passe au kernel de la machine physique [23]. La figure ci-dessous montre l'architecture de UML.

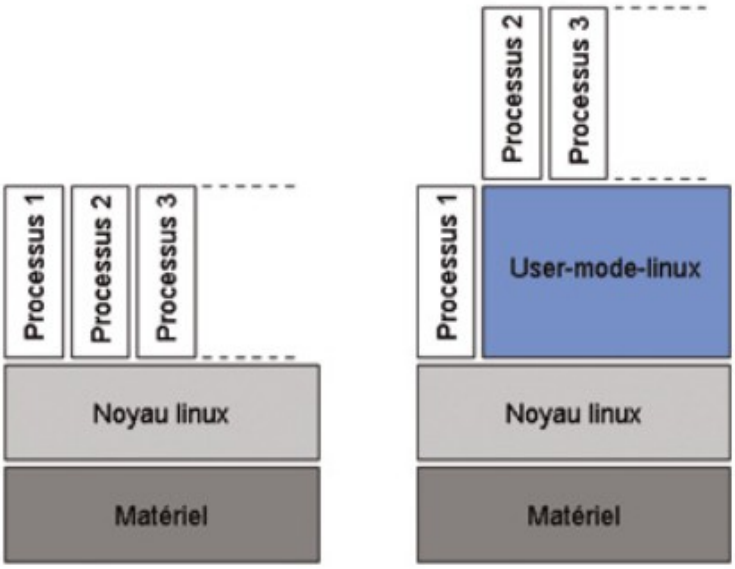


***Figure II.6**: Architecture de UML [22].*

UML permet à plusieurs machines virtuelles de se partager un même fichier (root-fs) commun en Lecture seule. Les opérations d'écriture de chacune des machines virtuelles sont déportées dans des fichiers séparés propres à chaque machine virtuelle. Ces fichiers ne contenant que les différences avec le root-fs, sont dénommés fichiers COW (abréviation de Copy On Write) [23].

UML a introduit les fichiers Copy On Write, Plusieurs machines virtuelles se partagent un rootfs commun accessible en lecture seule, Chaque machine virtuelle dispose d'un espace de débordement dit Copy On Write dans lequel sont déportées les surcharges apportées au rootfs de référence. Comme le montre la figure ci-dessous.

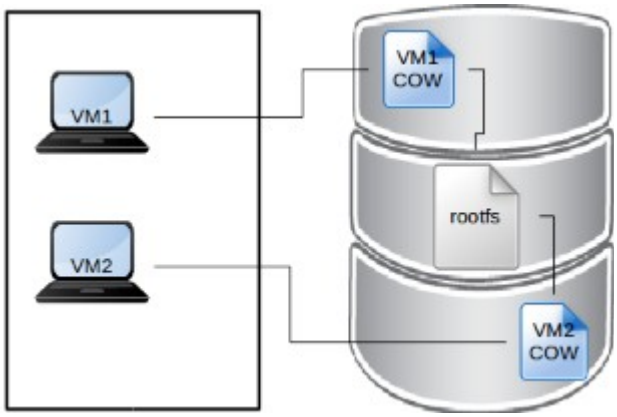


***Figure II.7**: Les fichier Rootfs et COW.*

5.2. La relation entre UML et Netkit

Les machines virtuelles Netkit sont basées sur le noyau UML (User Mode Linux), c'est a dire Netkit utilise UML comme un noyau, mais l'installation de UML n'est pas nécessaire. La machine est réglée pour fonctionner à l'intérieur d'UML et pour s'interfacer avec les commandes de Netkit [25]. Le démarrage d'une machine virtuelle signifie le démarrage d'une instance UML, ce qui nécessite souvent traiter des lignes de commande quelque peu longues et complexes. Pour cette raison, Netkit supporte la configuration et la gestion simple des machines virtuelles au moyen d'une interface composée de plusieurs scripts. Les scripts Netkit se chargent de lancer automatiquement des processus UML selon les besoins de l'utilisateur.

6. La virtualisation avec Netkit

6.1. Installation de Netkit

6.1.1. Pré-requis

Les machines virtuelles Netkit peuvent être configurées de manière flexible, de sorte que nous pouvons décider du nombre de ressources système sur la machine hôte qui doivent leur être allouées. Les exigences du système ne sont pas très strictes. Netkit peut fonctionner sur un ordinateur personnel avec:

matérielles :

- Architecture i386 32bits.
- CPU supérieur ou égal à 600MHz.
- RAM supérieur ou égal a 10 Mo pour chaque machine virtuelle (selon la configuration des machines virtuelles).
- Espace disque 600 Mo (1-20 Mo pour chaque machine virtuelle selon l'utilisation des machines virtuelles).

Logiciels :

- systèmes d'exploitation Linux.
- indépendant de la distribution Linux.

6.1.2. Les fichiers de Netkit

La distribution de Netkit complète consiste en trois modules logiciels :

- **netkit-2.8.tar.bz2**

  Le Netkit "core", qui contient des commandes, la documentation et d'autres outils nécessaires pour son fonctionnement.

- **netkit-filesystem-i386-F5.2.tar.bz2**

  Ce paquet fournit une image de système de fichiers à utiliser avec les machines virtuelles Netkit.

- **netkit-kernel-i386-K2.8.tar.bz2**

  Ce paquet fournit un noyau UML (User Mode Linux) pour une utilisation avec les machines virtuelles Netkit.

6.2. La configuration

Après l'installation des paquets de Netkit, il faut configuré les variables d'environnement avant le lancement des machines virtuelles.

Les variables d'environnement sont des valeur dynamique, chargées en mémoire, peuvent être utilisées par plusieurs processus fonctionnant simultanément. Elles servent à communiquer des informations entre programmes qui ne se trouvent pas sur la même ligne hiérarchique, et ont donc besoin d'une convention pour se communiquer mutuellement leurs choix. Les variables d'environnement à configurer pour Netkit sont :

- NETKIT_HOME (contient le répertoire où Netkit a été installé).
- PATH (pour accéder aux commandes Netkit standard.)
- MANPATH (pour accéder à la page de manuel).

Ces variables doivent être configurées avant le lancement des machines virtuelles avec la commande export. Il faut de configurer le fichier ~/.bashrc pour que ces variables soient initialisées automatiquement à l'ouverture de session. Pour cela, nous ajoutons les variables dans le fichier, Comme le montre la figure ci-dessous.

```
export NETKIT_HOME=/home/mohamed/Bureau/paquets/netkit
export MANPATH=:$NETKIT_HOME/man
export PATH=$NETKIT_HOME/bin:$PATH
```

***Figure II.8**: La configuration des variables d'environnement*

6.3. Lancement des machines virtuelles

Avant de démarrer les machines, vérifiez que le système est correctement configuré. Pour passer à la vérification, il convient de se déplacer dans le répertoire de Netkit et de lancer le script de vérification dont le nom est **check_configuration.sh**. Ce script vérifie si le système est configuré correctement pour faire fonctionner Netkit. Comme on peut le voir sur la figure suivante:

```
mohamed@debian:~/Bureau/paquets/netkit$ ./check_configuration.sh
>  Checking path correctness... passed.
>  Checking environment... passed.
>  Checking for availability of man pages... passed.
>  Checking for proper directories in the PATH... passed.
>  Checking for availability of auxiliary tools:
     awk          : ok
     basename     : ok
     date         : ok
     dirname      : ok
     find         : ok
     getopt       : ok
     grep         : ok
     head         : ok
     id           : ok
     kill         : ok
     ls           : ok
     lsof         : ok
     ps           : ok
     readlink     : ok
     wc           : ok
     port-helper  : ok
     tunctl       : ok
     uml_mconsole : ok
     uml_switch   : ok
passed.
>  Checking for availability of terminal emulator applications:
     xterm          : found
     konsole        : not found
     gnome-terminal : found
passed.
>  Checking filesystem type... passed.
>  Checking whether 32-bit executables can run... passed.

[ READY ] Congratulations! Your Netkit setup is now complete!
          Enjoy Netkit!
```

***Figure II.9**: Vérification de la configuration.*

Pour tester si Netkit fonctionne correctement, il est possible de démarrer une machine virtuelle simple par la commande vstart comme le montre la figure suivante :

```
mohamed@debian:~/Bureau$ vstart virtual_machine

============= Starting virtual machine "virtual_machine" =============
   Kernel:     /home/mohamed/Bureau/paquets/netkit/kernel/netkit-kernel
   Modules:    /home/mohamed/Bureau/paquets/netkit/kernel/modules
   Memory:     32 MB
   Model fs:   /home/mohamed/Bureau/paquets/netkit/fs/netkit-fs
   Filesystem: /home/mohamed/Bureau/virtual_machine.disk (new)
   Hostfs at:  /home/mohamed

Running ==> xterm -e /home/mohamed/Bureau/paquets/netkit/kernel/netkit-
kernel modules=/home/mohamed/Bureau/paquets/netkit/kernel/modules name=
virtual_machine title=virtual_machine umid=virtual_machine mem=36M ubd0=
/home/mohamed/Bureau/virtual_machine.disk,/home/mohamed/Bureau/paquets/
netkit/fs/netkit-fs root=98:1 uml_dir=/home/mohamed/.netkit/mconsole
hosthome=/home/mohamed quiet con0=fd:0,fd:1 con1=null SELINUX_INIT=0
```

***Figure II.10**: Démarrage d'une machine virtuelle.*

La figure II.11 montre la machine virtuelle Netkit créée. Sur le disque dure de la machine physique on voit apparaître un fichier nommé **nom_mv**.disk qui représente le disque dure de la machine virtuelle, ceci est le COW système de fichiers par défaut utilisé par les machines virtuelles, ce fichier est automatiquement créé dans le répertoire courant dès que la machine virtuelle nom_mv est démarré. Lors de démarrage la machine virtuelle remarquer une information particulier, la **phase 1** dans la figure II.11 qui permettre de monter le dossier « home » de l'utilisateur de la machine hôte sous /hosthome ce qui permet de accéder depuis la machine virtuelle. Ce terminal représente une machine, on peut donc réaliser toute les commandes classiques.

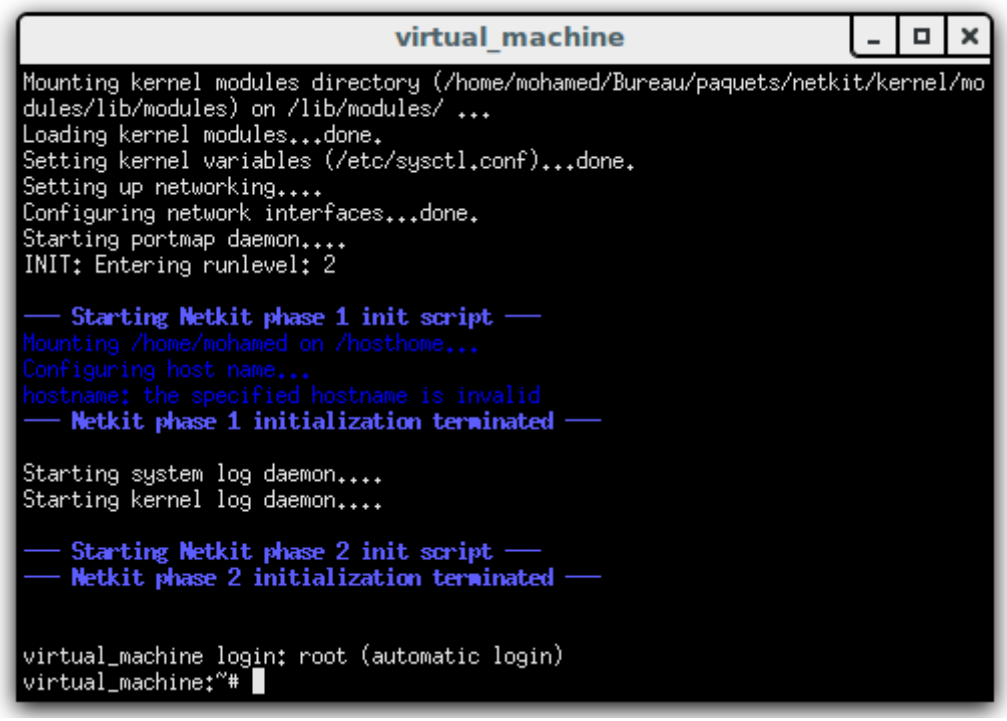


***Figure II.11**: Une machine virtuelle créée.*

6.4. Utilisation de Netkit

La fonctionnalité le plus intéressante de Netkit est l'interface utilisateur, qui comprend des outils permettant de configurer rapidement et facilement des expériences réseau complexes. Les machines virtuelles Netkit peuvent être gérées au moyen d'un ensemble de commandes composé de :

- les **vcommandes** (vtools) préfixées par 'v' .
- les **lcommandes** (ltools) préfixées par 'l'.

Toutes ces commandes doivent être exécutées sur la machine hôte.

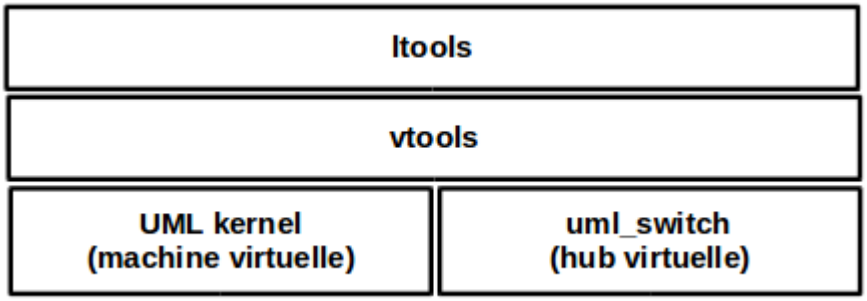


***Figure II.12**: les interfaces utilisateur de Netkit.*

6.4.1. Les « V » commandes

Ces commandes permettent de démarrer, d'arrêter et de configurer une machine virtuelle isolée, et ils fournissent les fonctionnalités suivantes:

vstart

La commande vstart peut être utilisée pour démarrer une nouvelle machine virtuelle. Les noms de machines virtuelles doivent être uniques pour chaque utilisateur. Pourtant, différents utilisateurs peuvent exécuter des machines portant le même nom.

Il permet de définir des paramètres tels que la quantité de mémoire disponible, les interfaces réseau et les domaines de collision… **.**

Par exemple, on va démarrer deux machines virtuelles mv1 et mv2, avec une interfaces réseau chacune, eth0 de mv1 et eth1 de mv2, et une quantité de mémoire (la taille en Méga octet) pour chaque machine, les deux interfaces sont attachés au même domaine de collision Domaine_Collision.

```
vstart mv1 --eth0=Domaine-Collision --mem=10
vstart mv2 --eth1=Domaine-Collision --mem=16
```

Une fois le démarrage terminé, les interfaces réseau de mv1 et mv2 peuvent être configurées en utilisant la commande **ifconfig**

vlist

La commande vlist affiche plusieurs informations sur les machines virtuelles en cours d'exécution. Si aucun argument n'est fourni, il affiche une liste des machines virtuelles qui ont été démarrées par l'utilisateur actuel. Les entrées de liste ont le format suivant: USER, VHOST, PID, SIZE, INTERFACES.

USER : est le nom de l'utilisateur qui a démarré la machine virtuelle.

VHOST : est le nom de la machine virtuelle.

PID : est le ID de la machine virtuelle.

SIZE : est la quantité réelle de mémoire consommée par la machine, en Ko.

INTERFACES : est une liste (éventuellement vide) des interfaces réseau de La machine virtuelle, ainsi que les domaines collision virtuelles (hubs).

On va lister les machines virtuelles crées précédemment par la commande **vlist** comme le montre la figure suivante :

```
mohamed@debian:~/Bureau$ vlist
USER             VHOST          PID     SIZE  INTERFACES
mohamed          mv1          29987    12844  eth0 @ Domaine-Collision
mohamed          mv2          31689    12824  eth1 @ Domaine-Collision

Total virtual machines:       2    (you),        2    (all users).
Total consumed memory:    25668 KB (you),    25668 KB (all users).
```

Figure II.13*: Les machines virtuelles en cours d'exécution.*

Si un nom d'un machine virtuelle est fourni comme argument, vlist fournit des informations détaillées sur la machine virtuelle, Comme le montre la figure II.14.

Information comptable :

PID: Machine virtuelle PID.
Owner: Nom de l'utilisateur qui a démarré la machine virtuelle.
Used mem: Quantité de mémoire consommée par la machine virtuelle.

Paramètres d'émulation :

Kernel: Nom du noyau UML utilisé par la machine virtuelle.
Memory: Quantité de mémoire disponible à l'intérieur de la machine virtuelle.
Model fs: Nom du modèle (backing) système de fichiers utilisé par la machine virtuelle.
Filesystem: Nom du système de fichiers (COW) utilisé par la machine virtuelle.
Interfaces: Liste des interfaces réseau de la machine virtuelle et les domaines de collision auxquels ils sont rattachés.
Hostfs at: répertoire du système de fichiers hôte qui est mis à disposition dans la machine virtuelle sous '/ hosthome '.
Console 1, Console 2: Appareils auxquels sont rattachés les consoles primaire et secondaire de la machine virtuelle.
Other args: Des paramètres supplémentaires ou des arguments qui ont été ajoutés à la ligne de commande du noyau machine virtuelle.
Mconsole: Nom du fichier socket temporaire qui sera utilisé pour envoyer des directives à la machine virtuelle (par exemple, pour arrêter la machine virtuelle).

```
mohamed@debian:~/Bureau$ vlist mv1

============= Information for virtual machine "mv1" =============
 --- Accounting information ---
   PID:         29987
   Owner:       mohamed
   Used mem:    12844 KB
 --- Emulation parameters ---
   Kernel:      xterm
   Modules:     /home/mohamed/Bureau/paquets/netkit/kernel/modules
   Memory:      10 MB
   Model fs:    /home/mohamed/Bureau/paquets/netkit/fs/netkit-fs
   Filesystem: /home/mohamed/Bureau/mv1.disk
   Interfaces: eth0 @ Domaine-Collision  (/home/mohamed/.netkit/hubs/
               vhub_mohamed_Domaine-Collision.cnct)
   Hostfs at:  /home/mohamed
   Console 1:  stdin/stdout
   Console 2:  disabled
   Other args: umid=mv1 root=98:1 uml_dir=/home/mohamed/.netkit/
               mconsole quiet SELINUX_INIT=0

   Mconsole:   /home/mohamed/.netkit/mconsole/mv1/mconsole
```

***Figure II.14**: Informations détaillés sur une machine virtuelles.*

vconfig :

La commande vconfig peut être utilisée pour attacher des interfaces réseau à des machines virtuelles en cours d'exécution. La machine concernée est identifiée par un ID-MACHINE, soit le PID ou le nom d'une machine virtuelle en cours d'exécution. Cela est utile pour modifier la configuration d'un scénario déjà en cours ou simplement pour éviter le redémarrage d'une machine parce qu'une de ses interfaces a été oubliée. Une ou plusieurs interfaces peuvent être attachées ou détachées sur une machine virtuelle en cours d'exécution. La commande suivante ajoute à mv1 une interface eth2 attachée au domaine de collision Domaine-Collision-I.

```
vconfig mv1 --eth2=Domaine-Collision-I
```

vhalt :

La commande vhalt peut être utilisée pour fermer correctement les machines virtuelles en cours d'exécution. L'utilisation de vhalt a exactement le même effet que d'exécution de la commande 'halt' à l'intérieur de la machine virtuelle.

La figure ci-dessous montre l'utilisation de vhalt pour arrêter mv1.

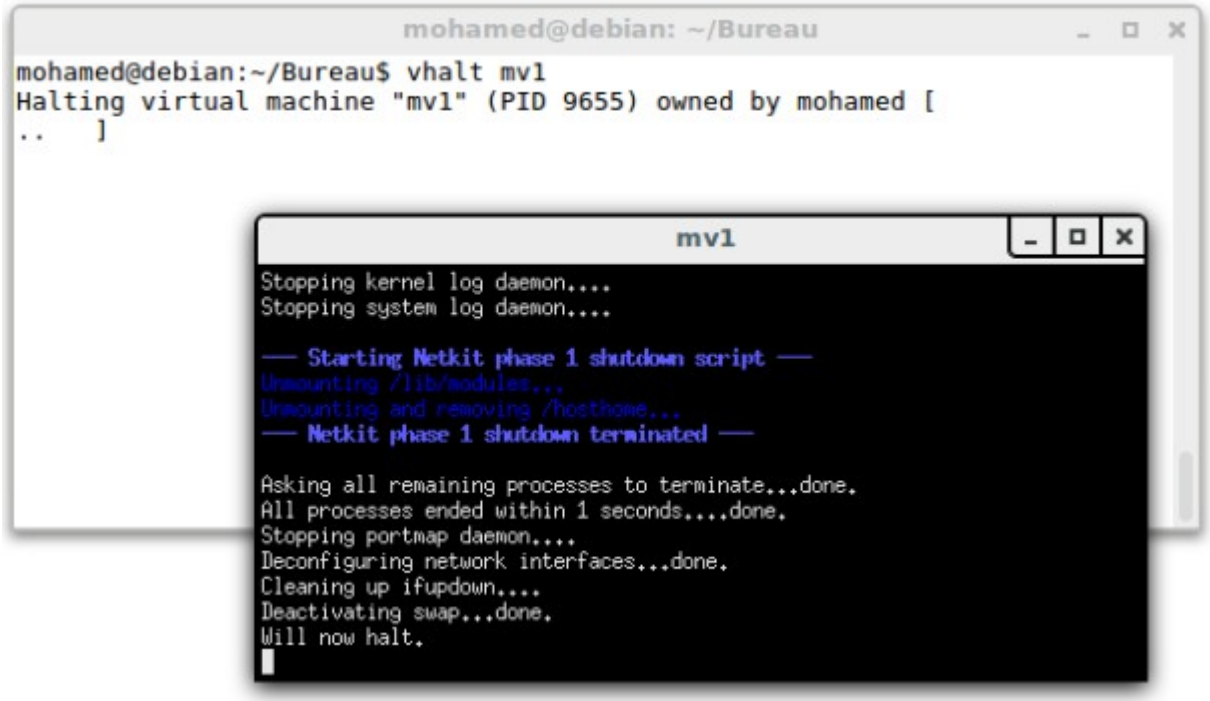


***Figure II.15**: Arrêt d'une machine virtuelle en utilisant vhalt.*

vcrash :

La commande vcrash peut être utilisée pour arrêter forcément les machines virtuelles en cours d'exécution. Il permet à la fois d'arrêter immédiatement les machines (c'est-à-dire sans avoir à passer par leur séquence d'arrêt). Cette commande est assez similaire à vhalt, sauf qu'elle est plus rapide et son effet est équivaut à débrancher abruptement le cordon d'alimentation de la machine virtuelle. Ceci est réalisé en demandant d'abord au noyau UML de s'arrêter à travers un socket de gestion UNIX spécial.

vclean :

La commande vclean peut être utilisée pour effectuer certaines opérations de nettoyage sur la machine hôte. Pour exécuter des machines virtuelles, Netkit lance plusieurs processus et effectue parfois certaines modifications de configuration (par exemple, lors de la mise en place des domaines de collision "tap"). vclean peut être invoqué pour exécuter plusieurs opérations: il peut simplement supprimer des uml_switch non utilisés, tuer toutes les machines virtuelles en cours d'exécution et supprimer les tunnels se connectant à un réseau externe. vclean peut être considéré comme le «bouton panique» de Netkit. La figure ci-dessous montre l'utilisation de vclean pour tuer les machines virtuelles lancées par n'importe quel utilisateur.

```
mohamed@debian:~/Bureau$ vclean --clean-all
Killing virtual machines owned by user mohamed:
   PIDs: 6676 6683 6684 6685 7037 7221 7229 7265 7267 29987 29989 30005
30006 30007 30008 30379 30573 30591 30637 30638 30640 30658 31689 31691
31715 31716 31717 31718 32089 32281 32299 32343 32346 32348 32353
   Killing... done.
Killing virtual hubs owned by user mohamed:
   None is currently running.
************* Removing tap configurations ************
   This will affect tap configurations for user mohamed.
******** This operation requires root privileges ********
Running ==> /home/mohamed/Bureau/paquets/netkit/bin/manage_tuntap stop
mohamed's password:
Disabling IP forwarding...            done.
Bringing down tap devices and tunnels:
Done.
Resetting permissions for /dev/net/tun...   done.
************** Abandoning root privileges ***************
```

Figure II.16*: Utilisation de vclean pour tuer les machines virtuels lancées.*

6.4.2. Les « L » commandes

Ces commandes permettent de configurer, de lancer ou d'arrêter facilement un scénario complexe de manière simple. Le « l » signifie "laboratoire" (abréviation de "lab"), et ils fournissent les fonctionnalités suivantes:

lstart :

Pour faciliter la configuration d'expériences réseau complexes avec Netkit, il est possible de les décrire complètement dans des fichiers de configuration spéciaux, afin que l'expérience puisse être lancée plus tard avec la commande lstart. En particulier, il démarre un ensemble de machines virtuelles qui font partie d'un laboratoire Netkit et les configure en fonction des paramètres contenus dans la description du laboratoire.

lhalt :

La commande lhalt ferme toutes les machines virtuelles d'un laboratoire Netkit. lhalt utilise vhalt pour arrêter les machines virtuelles. Par défaut, vhalt arrête toutes les machines virtuelles du laboratoire.

lcarsh :

La commande lcrash peut être utilisée pour fermer forcément toutes les machines virtuelles d'un laboratoire Netkit. lcrash utilise la commande vcrash pour arrêter les machines virtuelles.

linfo :

Cette commande peut être utilisée pour afficher les informations suivantes sur un laboratoire Netkit sans le lancer: des informations descriptives (extraites de lab.conf) et le nombre de machines virtuelles qui composent le laboratoire, si le laboratoire prend en charge le démarrage parallèle (au moyen d'un fichier lab.conf).

lclean :

lclean effectue simplement un nettoyage des fichiers temporaires restés après avoir exécuté un laboratoire Netkit. Il prend soin de supprimer tous les fichiers temporaires y compris les systèmes de fichiers de machines virtuelles (.disk). Par conséquent, toute modification apportée aux systèmes de fichiers des machines virtuelles est perdue. Bien sûr, toute partie persistante du labo est conservée, y compris les fichiers qui sont automatiquement copiés à l'intérieur des machines virtuelles pendant la phase de démarrage. Si elle est appelée sans arguments, lclean supprime les fichiers temporaires de toutes les machines virtuelles qui composent le laboratoire (c'est-à-dire, nom-machine est remplacé par le nom de chaque machine virtuelle).

ltest :

ltest est un outil permettant de récupérer et d'enregistrer des informations sur l'état de fonctionnement des machines virtuelles dans un laboratoire Netkit. Il permet de tester la configuration du labo et de vérifier que celui-çi a été monté proprement. Si le test est exécuté pour la première fois, le dernier crée une «signature» des résultats du test qui est utilisé comme une empreinte digitale d'un laboratoire de comportements corrects. Les prochaines fois que le test est exécuté, les résultats des tests sont comparés avec la signature pour vérifier que le laboratoire se comporte toujours comme prévu. Après avoir exécuté le test, le laboratoire est automatiquement arrêté, tous les fichiers temporaires sont supprimés et le résultat du test est imprimé en sortie. Chaque fois qu'un test est exécuté, un sous-répertoire `_test ' est créé à l'intérieur de répertoire actuelle cd lab (s'il n'existe pas déjà) et les résultats du test sont stockés à l'intérieur. Les résultats existants des tests précédents sont écrasés. Le sous-répertoire `_test / signature 'et` _test / results' contiendra respectivement la signature du test de laboratoire et le résultat du dernier test exécuté.

6.5. Création d'un réseau virtuelle Netkit

La création d'un réseau va consister à créer des machines et de préciser leur connectivité physique. Une machine virtuelle peut être aussi bien un hôte ou un routeur. Une machine avec 2 interfaces réseaux devient un routeur. En fait, un routeur est une machine avec une table de routage

correctement configurée.

La mise en place d'un réseau virtuelle Netkit sera de deux façons :

6.5.1. Création d'un réseau depuis un terminal

Dans ce cas les commandes utilisées sont les V_commandes.

6.5.1.1. La communication entre les machines virtuelles

Comme nous l'avons dit plus haut, La création d'une machine virtuelle s'effectue par la commande vstart <nom_machine>. Ici, on crée deux machines virtuelles pc1 et pc2, avec une interfaces réseau chacune, les deux interfaces sont attachés au même domaine de collision nommé « Hub ».

```
vstart pc1 --eth1=Hub
vstart pc2 --eth2=Hub
```

La figure suivante montre la topologie de réseau :

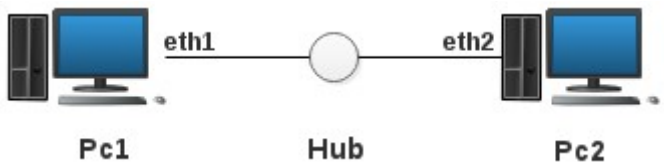
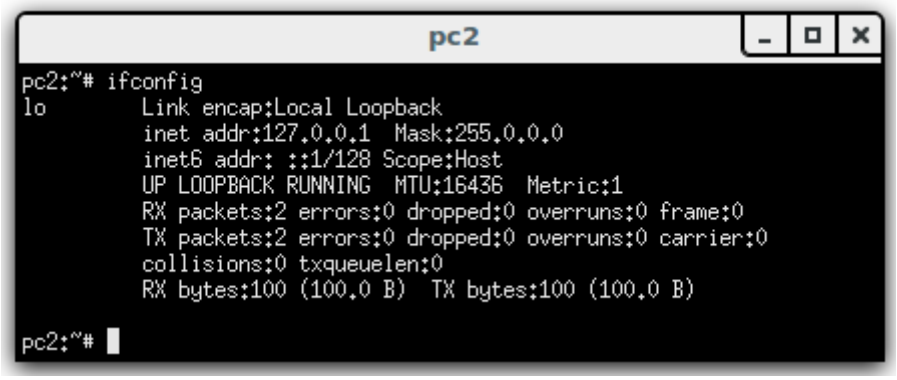


***Figure II.17**: La topologie de réseau.*

A partir de l'une des machines créées, on tape la commande ifconfig comme le montre ci-dessous.



La carte réseau eth2 n'apparaît pas, seule la carte de loopback (lo) est affichée. Pour la visualiser, on utilise le paramètre -a (ifconfig -a). La carte eth2 existe bien mais sans adresse IP, il faut ajouter une adresse IP statique de l'interface eth1 et eth2.

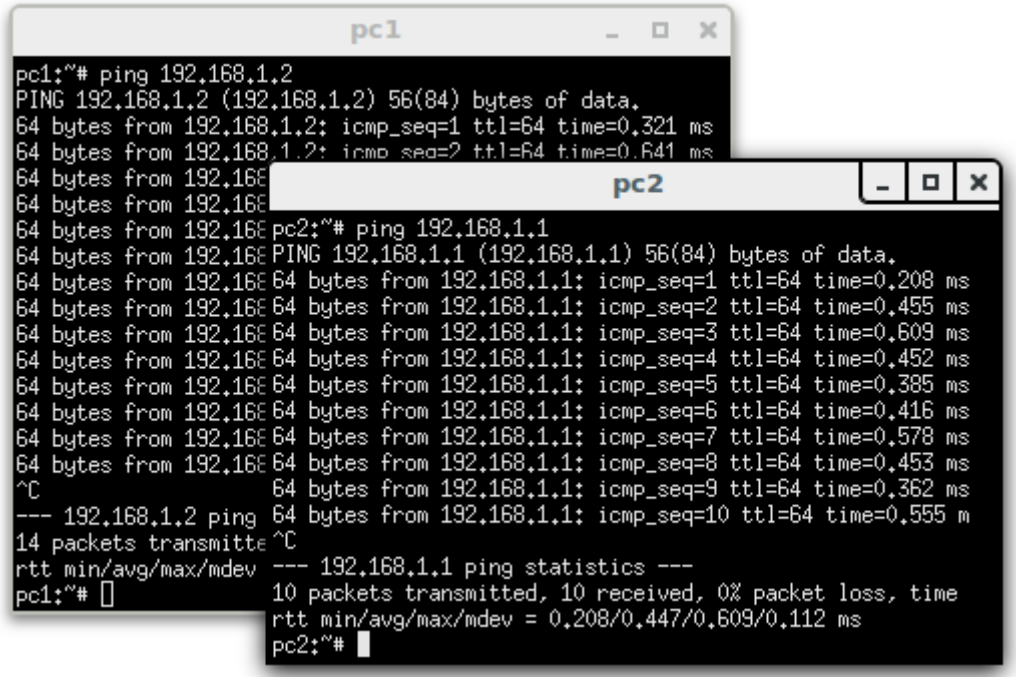
A partir de pc1 :

```
ifconfig eth1 192.168.1.1 netmask 255.255.255.0
```

A partir de pc2 :

```
ifconfig eth2 192.168.1.2 netmask 255.255.255.0
```

Maintenant les machines peuvent communiquer entre eux, on exécute la commande "ping" de pc1 vers pc2, et inversement de pc2 vers pc1 pour tester si ça fonctionne:



Considérons un autre réseau simple qui contient un routeur et deux ordinateurs :

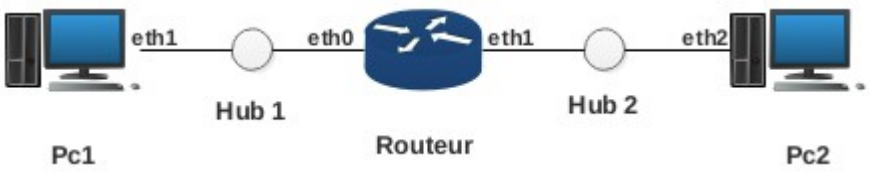


***Figure II.18**: Un topologie simple contenir un routeur et deux pc.*

La création des trois machines par les commande suivantes :

```
vstart pc1 --eth1=Hub1
vstart pc2 --eth2=Hub2
vstart routeur --eth0=Hub1 --eth1=Hub2
```

Après le démarrage des machines il faut configurer les adresses IP des interfaces réseaux.

A partir de pc1 :

```
ifconfig eth1 10.0.0.1 netmask 255.0.0.0
```

A partir de pc2 :

```
ifconfig eth2 192.168.1.2 netmask 255.255.255.0
```

A partir de routeur :

```
ifconfig eth0 10.0.0.4 netmask 255.0.0.0
ifconfig eth1 192.168.1.5 netmask 255.0.0.0
```

Maintenant les machines pc1 et pc2 relier avec le routeur mais il faut ajouter une route par défaut permettant d'aiguiller les paquets vers le passerelle appropriée. La route par default en pc1 et pc2 peut être configurée avec la commandes :

```
route add default gw 1.0.0.4 dev eth1
route add default gw 192.168.1.5 dev eth2
```

Les deux machines peuvent communiquer enter eux, sur la machine pc1 on va lancer un « ping » vers pc2 et le laisser tourner. Entre temps, sur la machine routeur on va lancer « tcpdump » pour capturer le trafic , comme montre la figure ci-dessous :

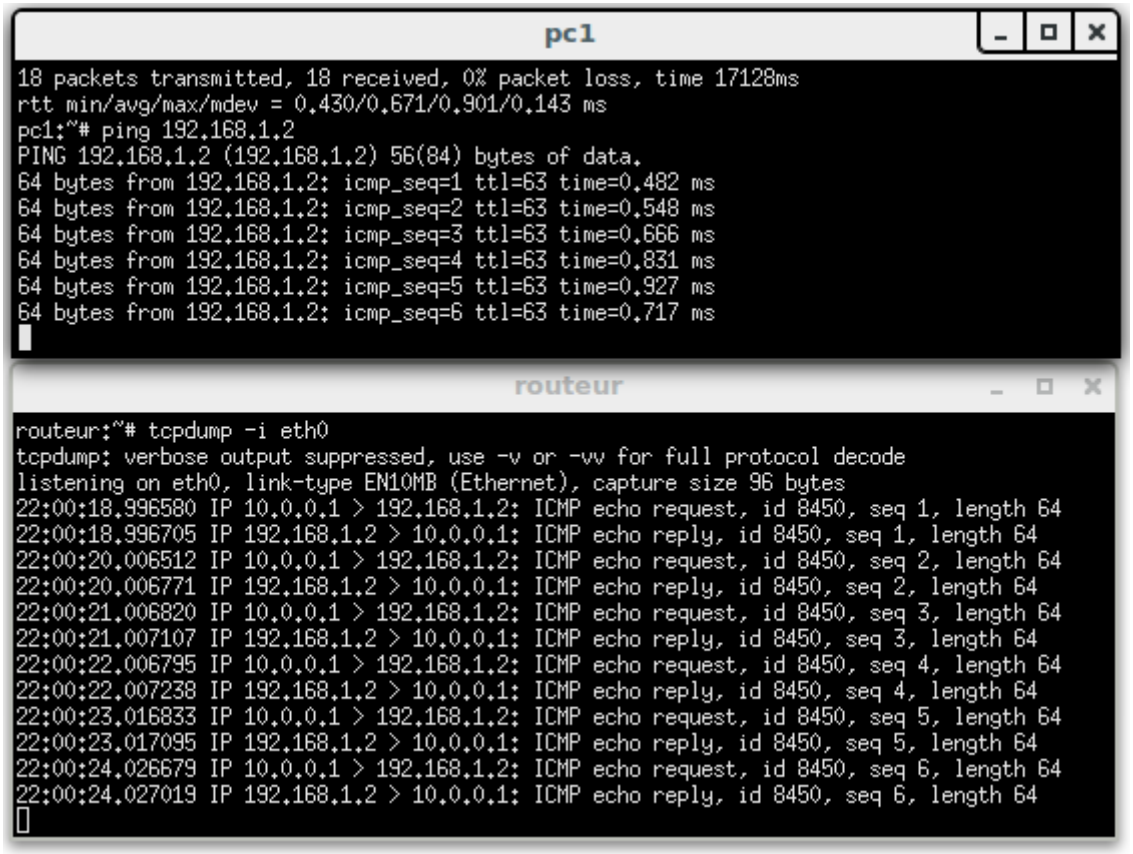


***Figure II.19**: teste de la communication entre les machines*

6.5.1.2. La communication entre la machine hôte et la machine virtuelle

Chaque machine virtuelle possède un répertoire personnel /hosthome qui pointe directement sur le répertoire personnel de l'utilisateur ayant lancé la machine virtuelle, tous les fichiers placés dans le répertoire personnel sur la machine hôte sont accessibles depuis les machines virtuelles par le répertoire /hosthome . Ceci permet de transférer simplement des fichiers entre l'hôte et les MVs.

pour connecter à l'internet et au réseau de la machine réelle il faut lancer la machine virtuelle avec une interface « **tap** ». Le domaine de collision TAP permet à la machine virtuelle de communiquer avec le réseau «réel» auquel est connectée la machine hôte. Une interface virtuelle, nommée **nk_tap_<nom-utilisateur>**, est aussi créée sur la machine hôte lors du démarrage de la machine virtuelle. Il faut définir une adresse pour chacune de ces deux interfaces. Les adresses doivent appartenir au même réseau et ne pas être utilisées sur le réseau «réel». Voici un exemple :

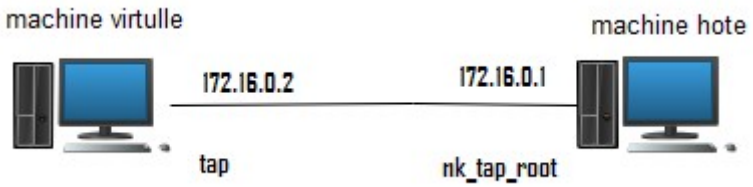


Figure II.20: *La communication a l'internet et au réseau de la machine réelle.*

On va créer une machine virtuelle « mv » pour communiquer à l'internet et au réseau de la machine réelle. Lors de la création de la machine virtuelle, il faut ajouter les paramètres en gras :

```
vstart mv  --eth1=tap,172.16.0.1,172.16.0.2
```

On définit ici une interface eth1 qui connecte la machine virtuelle à la machine réelle. La première adresse IP mentionnée est celle de l'interface côté machine hôte, la deuxième celle de l'interface côté mv. La création d'une interface TAP nécessite des droits administrateur. Il faut fournir le mot de passe du compte d'utilisateur pour exécuter la commande vstart en mode superutilisateur. Après avoir démarré la machine virtuelle, une nouvelle carte réseau créée sur la machine hôte:

```
nk_tap_mohamed Link encap:Ethernet  HWaddr de:11:ec:46:85:51
          inet adr:172.16.0.1  Bcast:172.16.255.255  Masque:255.255.0.0
          adr inet6: fe80::dc11:ecff:fe46:8551/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
          TX packets:57 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:500
          RX bytes:748 (748.0 B)  TX bytes:8026 (7.8 KiB)
```

La nouvelle carte réseau sur la machine virtuelle Netkit « mv » :

```
mv:~# ifconfig
eth1      Link encap:Ethernet  HWaddr 1e:ab:88:ea:af:05
          inet addr:172.16.0.2  Bcast:172.16.255.255  Mask:255.255.0.0
          inet6 addr: fe80::1cab:88ff:feea:af05/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:28 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4307 (4.2 KiB)  TX bytes:468 (468.0 B)
          Interrupt:5
```

Maintenant la machine virtuelle peut communiquer avec la machine hôte :

```
mv:~# ping -c 2 172.16.0.1
PING 172.16.0.1 (172.16.0.1) 56(84) bytes of data.
64 bytes from 172.16.0.1: icmp_seq=1 ttl=64 time=0.260 ms
64 bytes from 172.16.0.1: icmp_seq=2 ttl=64 time=0.364 ms

--- 172.16.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1013ms
rtt min/avg/max/mdev = 0.260/0.312/0.364/0.052 ms
```

Figure II.21*: La communication enter la machine virtuelle et la machine hôte.*

Et communiquer avec l'internet, on va tester avec le navigateur web de mode texte « lynx ».

```
mv:~# lynx google.com
```

On obtient :

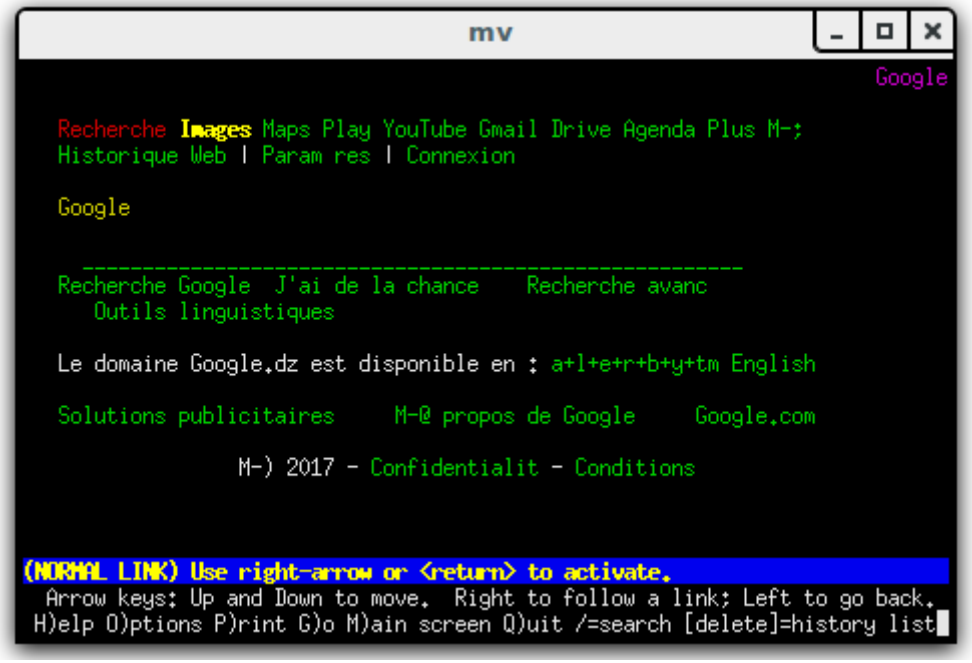


***Figure II.22**: La page web de Google de mode texte.*

6.5.2. Création d'un réseau à l'aide des laboratoires Netkit

Un laboratoire est un ensemble de machines virtuelles interconnectées constituant un réseau informatique qui peut être complexe. L'utilisation du laboratoire permet de configurer et lancer automatiquement toutes les machines virtuelles, ce laboratoire permet de concevoir, mais aussi de conserver une architecture de réseau complexe. Les principaux avantages par rapport à l'utilisation de V_commandes sont : Un démarrage et une configuration plus rapide de l'ensemble des machines virtuelles, et la possibilité de sauvegarder une topologie de réseau et sa configuration.

Le laboratoire peut être mis en œuvre de deux manières :
En écrivant un script de laboratoire qui fait appel à la commande vstart pour chaque machine virtuelle pour le démarrage.
En configurant un laboratoire Netkit standard qui peut être lancé en utilisant les L_commandes.

6.5.2.1. Les composants d'un laboratoire Netkit

La création d'un Lab s'effectue dans un répertoire distinct. Ce répertoire contiendra :

Un fichier de configuration (**lab.conf**) qui décrit les machines virtuelles qui seront lancées, leurs interfaces et les domaines de collision.

Un répertoire pour chaque machine (nom du répertoire=nom de la machine) qui contiendra les commandes et les fichiers propres à la machine. Les éléments dans ce répertoire sont montés à partir de la racine de la machine virtuelle.

Un fichier de configuration par machine appelé **<nom_machine>.startup**. Ce fichier contient des

commandes à exécuter au démarrage de la machine virtuelle. Et le script **<nom_MV>.shutdown** est exécuté à l'arrêt de la mv. Il faut veiller à garder le même nom pour une machine dans les fichiers et les répertoires.

Un dossier **shared** qui sera recopié à la racine de chaque machine virtuelle.

Un fichier **lab.dep** décrivant les relations de dépendance sur l'ordre de démarrage des machines virtuelles.

Le fichier lab.conf

Ce fichier décrit les configurations des MVs qui composent un laboratoire et la topologie réseau qui connecte les machines virtuelles du laboratoire. Il peu également contenir des paramètres optionnels informatifs qui sont affichés dans la console de la machine virtuelle au démarrage du lab :

```
Lab directory: « le répertoire du Lab »
Version:       « précise la version du laboratoire »
Author:        « le nom du réalisateur du laboratoire »
Email:         « permet de laisser un E-mail »
Web:           « permet de préciser l'adresse où trouver le lab »
Description:   « une description du lab »
```

Le fichier lab.conf, se termine par la description de la connexion des interfaces réseau de la manière suivante :

```
nom_MV[Arg]=Valeur
```

Si Arg (lire i) est un nombre, alors Valeur est le nom du domaine de collision sur lequel l'interface ethi est attaché. Si Arg est une chaîne, alors c'est le nom d'une option de vstart et Valeur est un argument de cette option.

Si l'interface eth0 des machines pc1 et pc2 sont connectées sur le même domaine de collision, et pc2 avec une quantité de mémoire 128 M octet ceci se déclare de cette manière :

```
pc1[0]=hub1
pc2[0]=hub1
pc2[mem]=128
```

par exemple la machine pc1 est connectée avec la machine hôte ou l'internet le domaine de collision 'tap' est créé de cette manière :

```
pc1[1]=tap,adresse_machine_hôte,adresse_machine_virtuelle
```

6.6. Laboratoire Netkit

A l'aide d'un laboratoire Netkit nous allons créer le scénario suivant :

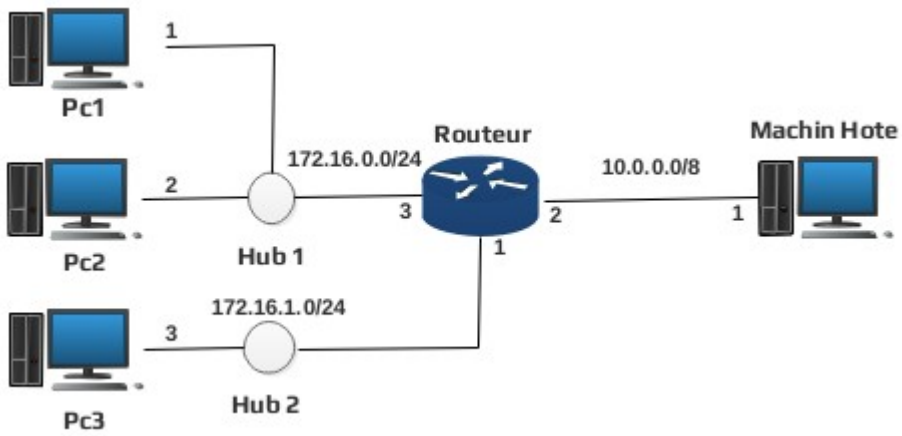
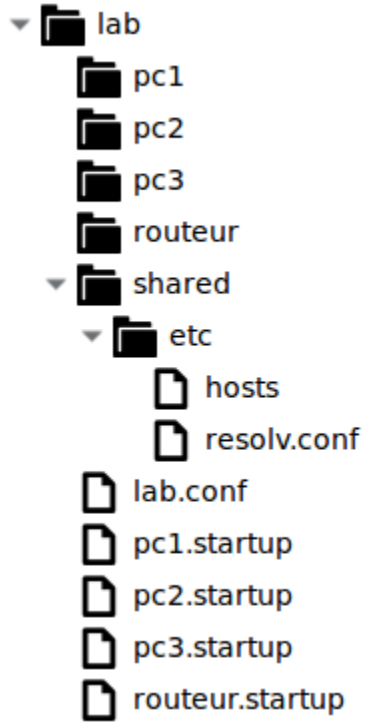


***Figure II.23**: La topologie de réseau.*

Pour cet exemple on a suivi les étapes suivantes :

1. Créer un répertoire nommé lab
2. Accéder à mon_lab
3. Créer pour chaque machine son propre répertoire.
4. Créer le fichier de description de topologie Lab.conf.
5. Créer le fichier de démarrage pour chaque machine.
6. Lancer le Laboratoire.

Voici l'arborescence des dossiers :



- Le fichier **lab.conf** :

```
LAB_DESCRIPTION="laboratoire Netkit"
LAB_VERSION="0.1"
LAB_AUTHOR="salmi et chehrouri"
LAB_EMAIL="master@gmail.com"
LAB_WEB="http://www.lagh-univ.dz"

routeur[2]=tap,10.0.0.1,10.0.0.2
pc1[0]="Hub1"
pc2[1]="Hub1"
routeur[1]="Hub1"
pc3[1]="Hub2"
routeur[0]="Hub2"
```

- Le fichier **pc1.startup** :

```
/sbin/ifconfig eth0 172.16.0.1 netmask 255.255.0.0 up
route add default gw 172.16.0.3 dev eth0
```

- Le fichier **pc2.startup** :

```
/sbin/ifconfig eth1 172.16.0.2 netmask 255.255.0.0 up
route add default gw 172.16.0.3 dev eth1
```

- Le fichier **pc3.startup** :

```
/sbin/ifconfig eth1 172.17.1.3 netmask 255.255.0.0 up
route add default gw 172.17.1.1 dev eth1
```

- Le fichier **routeur.startup** :

```
/sbin/ifconfig eth1 172.16.0.3 netmask 255.255.0.0 up
/sbin/ifconfig eth0 172.17.1.1 netmask 255.255.0.0 up
```

- Le fichier **hosts** :

```
172.16.0.1 pc1
172.16.0.2 pc2
172.17.1.3 pc3
10.0.0.1 machineHote
```

Le fichiers /etc/hosts offre un moyen de résoudre localement des noms de machines.Les noms étant généralement plus simples à retenir que les adresses IP, il est en général plus pratique de pouvoir utiliser un nom plutôt qu'une adresse IP, par exemple quand on fait un ping. Des correspondances entre IP et noms sont stockées dans ce fichier. Pour notre exemple, il peut être intéressant de créer une fichier /shared/etc/hosts dans le lab avec tous les noms utilisés, ils seront utilisables sur tous les postes.

- Le fichier **resolv.conf :**

```
nameserver 208.67.222.222
Nameserver 208.67.220.220
```

La résolution de nom est généralement réalisée par les DNS. Pour, cela il faut indiquer l'adresse des

serveurs de nom (DNS) dans le fichier /shared/etc/resolv.conf.

Lancer maintenant le Lab avec la commande lstart saisie dans le terminale de machine hôte. lstart lance les MVs de Lab en séquence (en série), pour le faire en parallèle on ajoute l'option –p. Le terminale de la machine hôte affiche les lignes suivantes (figure II.24):

```
======================== Starting lab ============================
Lab directory: /home/mohamed/Bureau/netkit/***mémoire***+/lab
Version:       0.1
Author:        salmi et chehrouri
Email:         master@gmail.com
Web:           http://www.lagh-univ.dz
Description:
laboratoire Netkit
================================================================
You chose to use parallel startup.
Starting "pc3"...
Starting "pc2"...
Starting "pc1"...
Starting "routeur"...
mohamed's password:

The lab has been started.
================================================================
```

***Figure II.24**: Le lancement de lab.*

Les terminaux des machines virtuelle du Lab s'affichent dans la figure II.25.

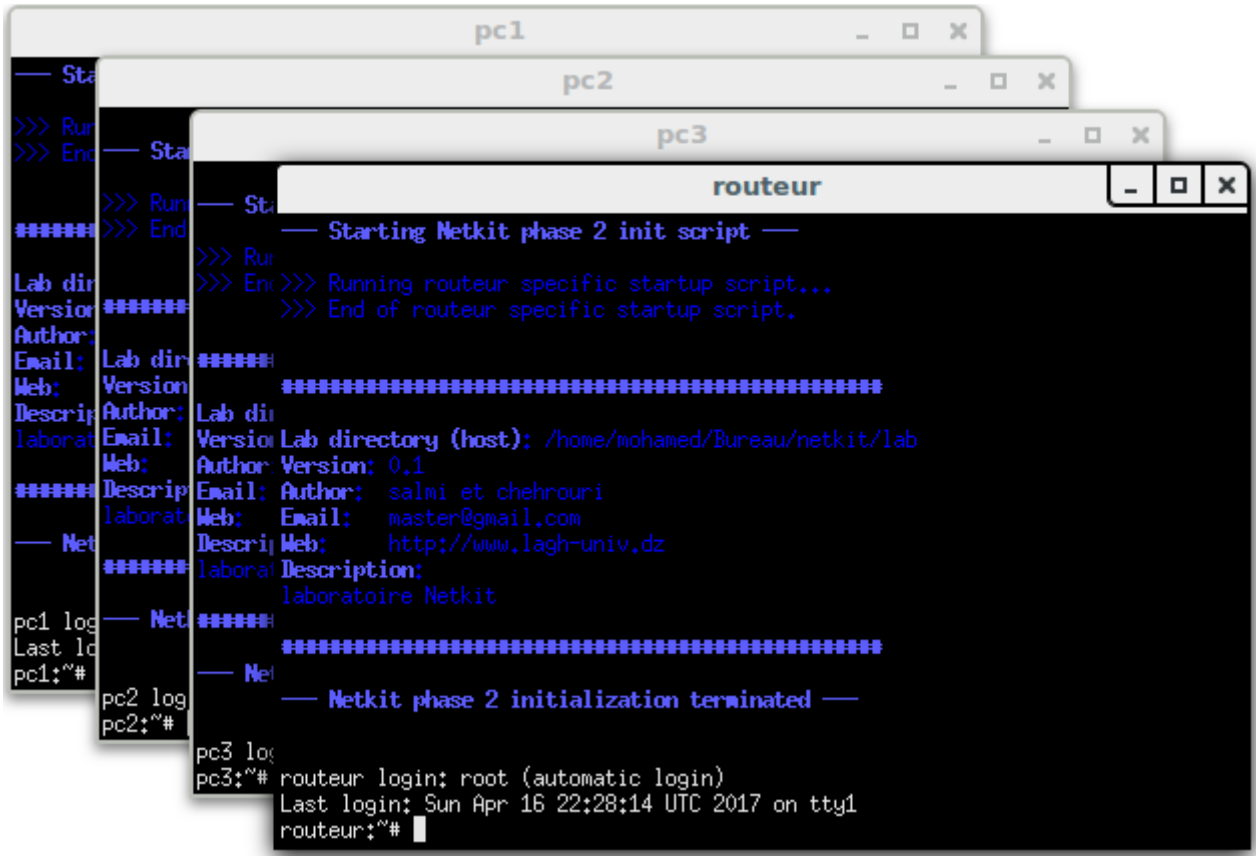


***Figure II.25 :** Les machines virtuelle du lab.*

Maintenant il faut ajouter une route statique à la machine réel (hôte) afin de lui indiquer comment accéder aux réseaux créés après la machine virtuelle (routeur). Voici les commandes à exécuter dans le terminal de la machine hôte :

```
sudo route add -net 172.16.0.0 netmask 255.255.0.0 gw 10.0.0.2
sudo route add -net 172.17.0.0 netmask 255.255.0.0 gw 10.0.0.2
```

Les commandes sont à interpréter de la manière suivante :

Pour atteindre le réseau 172.16.0.0 et le réseau 172.17.0.0 (celui sur lequel se trouve les machines virtuelles pc1 et pc2 et pc3), il faut envoyer les paquets 10.0.0.2 qui sont les interfaces de la routeur. Cette route statique sera supprimée lors de l'arrêt du lab ou du redémarrage de la machine hôte.

7. tcpdump :Capture et analyse de paquet

C'est un outil en ligne de commande qui nous permet de capturer et d'analyser tous les paquets qui sont transis par une carte réseau. Il est disponible sur la plupart des systèmes d'exploitations (Linux, Mac, Windows...). Il permet d'enregistrer des informations dans un fichier pour analyse ultérieure.

Avec un simple ping d'une machine virtuelle a une autre par exemple, on peut capturer et analyser Les paquets qui passent par le routeur. Un point de contrôle sur l'interface eth1 du routeur permet d'observer le dialogue entres les deux machines « request/reply ». Pour réaliser une capture, on va utiliser le tcpdump spécifiant l'interface où il doit être le point de contrôle, ici c'est eth1, à l'aide de la ligne de commande suivante dans le console du routeur:

```
tcpdump –i eth1 –w /hostlab/routeur/routeur.pcap
```

L'option -w de tcpdump, permet d'afficher les résultats du capture dans un fichier spécifié, alors on va sauvegarder les résultats du capture dans un fichier d'extension .pcap qui est interprété par le capteur de trafics Wireshark qui fournie une interface graphique et des détails très importants sur chaque trafic. Le routeur capture tous les trafics arrivés après le lancement d'un ping sur la machine pc2 vers google.com, le routeur affiche le nombre de trafics reçue sur l'interface eth1 et il les sauvegarde dans le fichier hostlab/routeur/routeur.pcap. Le répertoire hostlab est le répertoire lab.

On lance le wireshark sur la machine hôte et on ouvre le fichier hostlab/routeur/routeur.pcap on obtient l'interface présentée dans la figure II.26:

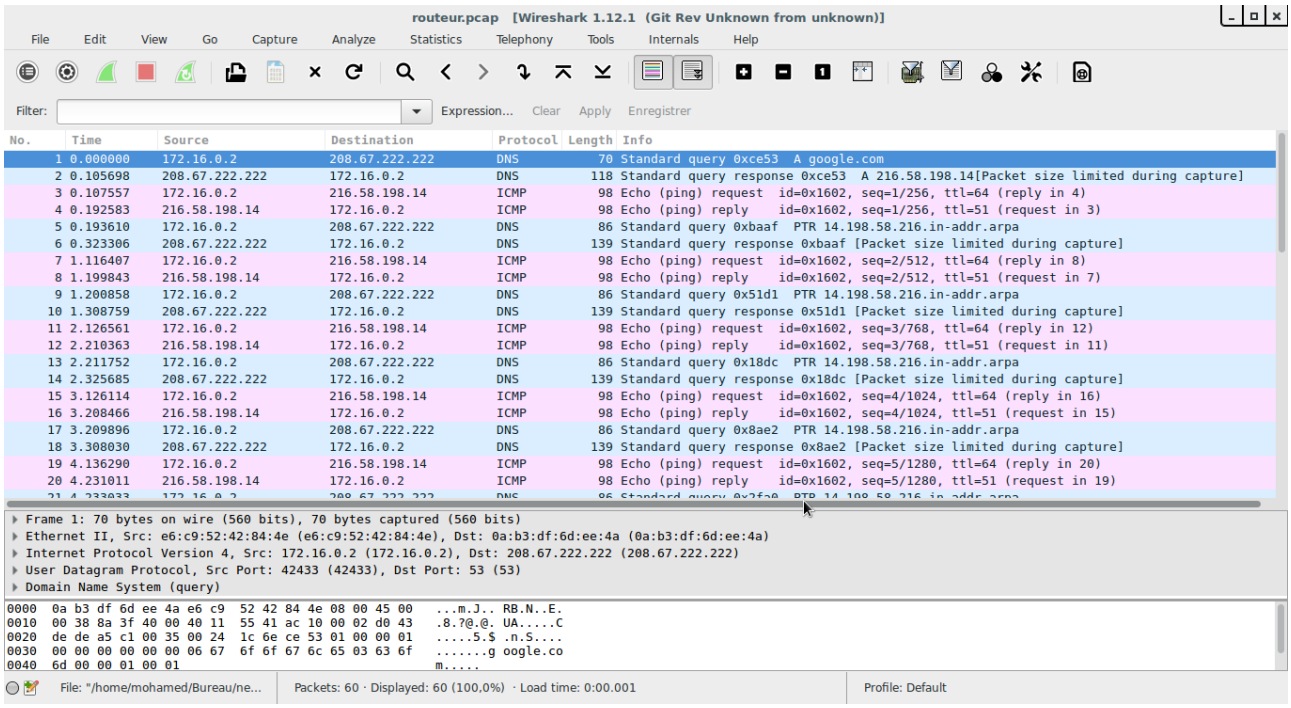


***Figure II.26**: L'interface de Wireshark.*

8. Le routage

8.1. Principe

Le routage est une fonction de la couche 3 du modèle OSI [26]. C'est un processus permettant à des paquets de trouver un chemin pour atteindre leur destination, Les routeurs utilisent une base de données appelée la table de routage qui est une véritable cartographie des itinéraires à suivre en fonction de l'adresse visée, définissant les différentes routes. Pour remplir sa table de routage, le routeur utilise deux procédés, le routage statique et le routage dynamique.

8.2. Les types de routage

8.2.1. Routage statique

Les informations sont mises à jour manuellement à chaque modification topologique de l'inter réseau. Ces modifications de la topologie réseau obligent l'administrateur à ajouter et supprimer des routes statiques pour tenir compte de ces modifications [27] .

8.2.1.1. Mise en œuvre

L'objectif de ce laboratoire est de connaître le déroulement du routage statique pour cela on a mis en place un réseau virtuel qui permet au Pc d'accéder à un serveur web à partir les routeurs. Nous configurons les routeurs en utilisant le routage statique nous plaçons un seul pc et un seul serveur

web et 5 routeurs dans notre laboratoire. La figure ci-dessous montre la topologie de réseau.

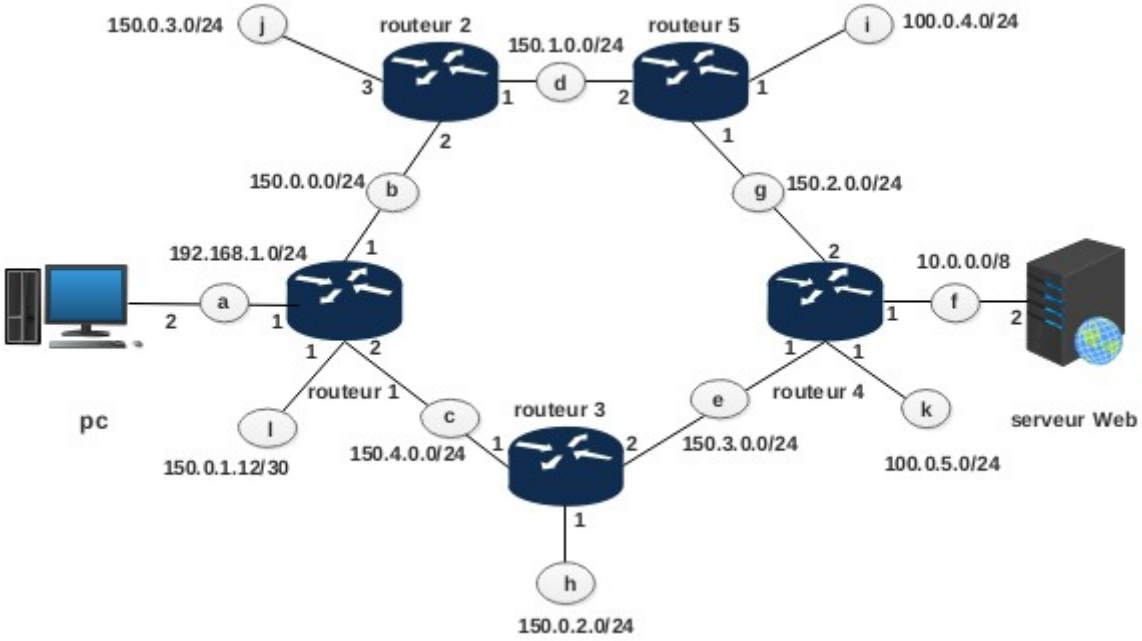


***Figure II.27**: La topologie de réseau.*

Configuration du fichier **lab.conf**

```
LAB_DESCRIPTION="Mise en oeuver du routage statique"
LAB_VERSION="0.1"
LAB_AUTHOR="chehrouri et salmi"
LAB_EMAIL="master@gmail.com"
LAB_WEB="http://www.lagh-univ.dz"

pc[0]="a"
r1[0]="a"
r1[1]="b"
r1[2]="c"
r1[3]="l"
r2[0]="d"
r2[1]="b"
r2[2]="j"
r3[0]="h"
r3[1]="e"
r3[2]="c"
r4[0]="g"
r4[1]="e"
r4[2]="f"
r4[3]="k"
r5[0]="d"
r5[1]="g"
r5[2]="i"
serverWeb[2]="f"
```

Configuration des **fichiers .startup**

- Le fichier **Pc.startup:**

```
ifconfig eth0 192.168.1.2 up
route add default gw 192.168.1.1 dev eth0
```

Dans ce fichier nous déterminons l'adresse **IP (**192.168.1.2) à l'interface eth0 du pc et nous déterminons une route par défaut (192.168.1.1 adresse de Gateway) à suivre par le PC.

- *Le fichier **r1.startup:***

```
ifconfig eth0 192.168.1.1 up
ifconfig eth1 150.0.0.1 up
ifconfig eth2 150.4.0.2 up
ifconfig eth3 150.0.1.1 up
route add -net 150.3.0.0 netmask 255.255.255.0 gw 150.4.0.1 dev eth2
route add -net 10.0.0.0 netmask 255.0.0.0 gw 150.4.0.1 dev eth2
```

Dans ce fichier nous déterminons l'adresse IP 192.168.1.1 à l'interface eth0, l'adresse IP *150.0.0.1* à l'interface eth1, l'adresse IP *150.4.0.2* à l'interface eth2 et l'adresse IP *150.0.1.1* à l'interface eth3 du routeur 1 et nous déterminons la route suivie par ce routeur.

La première route permet au routeur d'accéder à un sous-réseau 150.3.0.0/24 via adresse IP *150.4.0.1*.

La seconde route permet au routeur d'accéder à un autre sous-réseau *10.0.0.0/8* à partir l'adresse IP *150.4.0.1*.

Concernant les autres routeurs nous appliquons les mêmes principes comme ceux du routeur 1. En appliquant des changements qui dépendent de nos besoins.

- Le fichier **r2.startup:**

```
ifconfig eth0 150.1.0.1 up
ifconfig eth1 150.0.0.2 up
ifconfig eth2 150.0.3.3 up
```

- Le fichier **r3.startup:**

```
ifconfig eth0 150.0.2.1 up
ifconfig eth1 150.3.0.2 up
ifconfig eth2 150.4.0.1 up
route add -net 10.0.0.0 netmask 255.255.255.0 gw 150.3.0.1 dev eth1
route add -net 192.168.1.0 netmask 255.255.255.0 gw 150.4.0.2 dev eth2
```

- Le fichier **r4.startup:**

```
ifconfig eth0 150.2.0.2 up
ifconfig eth1 150.3.0.1  up
ifconfig eth2 10.0.0.1 up
ifconfig eth3 100.0.5.1 up
route add -net 150.4.0.0 netmask 255.255.255.0 gw 150.3.0.2 dev eth1
route add -net 192.168.1.0 netmask 255.255.255.0 gw 150.3.0.2 dev eth1
```

- Le fichier **r5.startup:**

```
ifconfig eth0 150.1.0.2 up
ifconfig eth1 150.2.0.1 up
ifconfig eth2 100.0.4.1 up
```

- Le fichier **serverWeb.startup:**

```
ifconfig eth2 10.0.0.2 up
route add default gw 10.0.0.1 dev eth2
```

À l'aide de la commande route -n nous obtenons les tables de routage de chaque machine, par exemple on va afficher la table de routage de routeur 1 :

```
r1:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
150.3.0.0       150.4.0.1       255.255.255.0   UG    0      0        0 eth2
150.4.0.0       0.0.0.0         255.255.0.0     U     0      0        0 eth2
150.0.0.0       0.0.0.0         255.255.0.0     U     0      0        0 eth1
150.0.0.0       0.0.0.0         255.255.0.0     U     0      0        0 eth3
10.0.0.0        150.4.0.1       255.0.0.0       UG    0      0        0 eth2
```

8.2.1.2. Test de connectivité entre le pc et le serveur web

L'échange de requêtes ICMP (Ping) entre le pc et le serveur web se passe via les routeurs et se connecte avec succès (la figure II.28), à cause des tables de routage qui se trouvent dans les routeurs contiennent des chemins vers la destination menée.

```
Pc:~# ping -c 3 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=61 time=72.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=61 time=1.33 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=61 time=0.718 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2016ms
rtt min/avg/max/mdev = 0.718/24.807/72.371/33.633 ms
```

***Figure II.28**: Test de la connectivité.*

Nous utilisons la commande tcpdump au niveau du routeur 3 pour L'analyse du trafic. Le pc (192.168.1.2) envoie un paquet qui contient le ***request*** puis le serveur web (10.0.0.2) répond par un autre paquet de ***reply***.

```
r3:~# tcpdump -i eth1
22:31:06.150523 IP 192.168.1.2 > 10.0.0.2: ICMP echo request, id 4610,
seq 1, length 64
22:31:06.159087 IP 10.0.0.2 > 192.168.1.2: ICMP echo reply, id 4610,
seq 1, length 64
22:31:07.149520 IP 192.168.1.2 > 10.0.0.2: ICMP echo request, id 4610,
seq 2, length 64
22:31:07.149888 IP 10.0.0.2 > 192.168.1.2: ICMP echo reply, id 4610,
seq 2, length 64
22:31:08.150842 IP 192.168.1.2 > 10.0.0.2: ICMP echo request, id 4610,
seq 3, length 64
22:31:08.151162 IP 10.0.0.2 > 192.168.1.2: ICMP echo reply, id 4610,
seq 3, length 64
```

Figure II.29*: L'analyse du trafic.*

Pour le traçage du chemin entre le pc et le serveur web nous utilisons le command ***traceroute*** pour afficher ce chemin.

```
pc:~# traceroute 10.0.0.2
traceroute to 10.0.0.2 (10.0.0.2), 64 hops max, 40 byte packets
 1  192.168.1.1 (192.168.1.1)  11 ms  0 ms  0 ms
 2  150.4.0.1 (150.4.0.1)  11 ms  0 ms  0 ms
 3  150.3.0.1 (150.3.0.1)  11 ms  0 ms  0 ms
 4  10.0.0.2 (10.0.0.2)  11 ms  0 ms  0 ms
```

Figure II.30*: Traçage du chemin depuis pc vers serverWeb.*

8.2.1.3. Détection du Problème de routage statique

Nous provoquons une panne sur le routeur 3 (arrêter de l'interface eth2), par la commande suivante :

```
r3:~# ifconfig eth2 down
```

Ensuite, nous testons la connectivité entre le pc et le serveur web.

```
pc:~# ping -c 2 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 192.168.1.1 icmp_seq=1 Destination Host Unreachable
From 192.168.1.1 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time
1016ms, pipe 2
```

Figure II.31*: Test de la connectivité.*

nous constatons que le pc n’est pas connecté au serveur web.

On va tracer la route entre le pc et le serveurWeb.

```
pc:~# traceroute 10.0.0.2
traceroute to 10.0.0.2 (10.0.0.2), 64 hops max, 40 byte packets
 1  192.168.1.1 (192.168.1.1)  1 ms  0 ms  0 ms
 2  * 192.168.1.1 (192.168.1.1)  43 ms !H *
```

Figure II.32*: Traçage du chemin depuis pc vers serverWeb.*

Pour résoudre ce problème, l’administrateur du réseau cherche le routeur endommagé et le configure manuellement pour déterminer une nouvelle route vers le serveur web tout au suivant la nouvelle topologie du réseau.

8.2.2. Routage dynamique

Les informations relatives à la route sont mises à jour automatiquement entre les routeurs [27]. Ces derniers utilisent des protocoles de routage pour chercher le chemin le plus efficace d'une unité à une autre, il existe deux familles de protocoles de routage [26]:

Les protocoles IGP (Interior Gateway Protocol):

Acheminent les données au sein d'un système autonome par exemple protocole RIP et RIPv2 protocole OSPF …etc.

Les protocoles EGP (Exterior Gateway Protocol):

Acheminent les données entre les systèmes autonomes. Le protocole BGP est un exemple de ce type de protocole.

8.2.2.1. Mise en œuvre

L’objectif de ce laboratoire est de connaître le déroulement du routage dynamique pour se faire, on a créé un réseau Virtuel qui permet au pc d’accéder à un serveur web à partir les routeurs. Nous configurons les routeurs en utilisant le protocole IGP RIPv2 et nous plaçons la même topologie que le laboratoire du routage statique. (figure II.27).

8.2.2.2. Définition le protocole RIP

Le protocole RIP [26] est un protocole de routage à vecteur de distance qui utilise le nombre de sauts comme métrique pour déterminer la direction et la distance vers n'importe quelle liaison de l'inter réseau. S’il existe plusieurs chemins vers une destination, le protocole RIP sélectionne celui qui comporte le moins de sauts. Toutefois, le nombre de sauts étant la seule métrique de routage utilisée

par ce protocole, il ne sélectionne pas toujours le chemin le plus rapide. En outre, le protocole RIP ne peut acheminer un paquet au-delà de 15 sauts. La version 2 (RIPv2) fournit un routage par préfixe et envoie les informations de masque de sous-réseau dans ses mises à jour de routage. On parle ici de routage sans classe. Avec les protocoles de routage sans classe, les sous-réseaux d'un même réseau peuvent comporter des masques différents. Cette technique fait référence à l'utilisation de masques de sous-réseau de longueur variable (VLSM).

Configuration du fichier **lab.conf**

```
LAB_DESCRIPTION="Mise en oeuver du routage dynamique"
LAB_VERSION="0.1"
LAB_AUTHOR="chehrouri et salmi"
LAB_EMAIL="master@gmail.com"
LAB_WEB="http://www.lagh-univ.dz"

pc[0]="a"
r1[0]="a"
r1[1]="b"
r1[2]="c"
r1[3]="l"
r2[0]="d"
r2[1]="b"
r2[2]="j"
r3[0]="h"
r3[1]="e"
r3[2]="c"
r4[0]="g"
r4[1]="e"
r4[2]="f"
r4[3]="k"
r5[0]="d"
r5[1]="g"
r5[2]="i"
serverWeb[2]="f"
```

Configuration des **fichiers .startup**

- Le fichier **pc.startup:**

```
ifconfig eth0 192.168.1.2 up
route add default gw 192.168.1.1 dev eth0
```

- Le fichier **r1.startup:**

```
ifconfig eth0 192.168.1.1 up
ifconfig eth1 150.0.0.1 up
ifconfig eth2 150.4.0.2 up
ifconfig eth3 150.0.1.1 up
/etc/init.d/zebra start
```

Dans ce fichier nous déterminons l'adresse IP 192.168.1.1 à l'interface eth0, l'adresse IP 150.0.0.1 à l'interface eth1, l'adresse IP 150.4.0.2 à l'interface eth2 et l'adresse IP 150.0.1.1 à l'interface eth3. Puis nous démarrons le service zebra (un logiciel qui gère le routage).

Concernant les autres routeurs nous appliquons les mêmes principes comme ceux du routeur 1, en appliquant des changements qui dépendent à nos besoins.

- Le fichier **r2.startup:**

```
ifconfig eth0 150.1.0.1 up
ifconfig eth1 150.0.0.2 up
ifconfig eth2 150.0.3.3 up
/etc/init.d/zebra start
```

- Le fichier **r3.startup:**

```
ifconfig eth0 150.0.2.1 up
ifconfig eth1 150.3.0.2 up
ifconfig eth2 150.4.0.1 up
/etc/init.d/zebra start
```

- Le fichier **r4.startup:**

```
ifconfig eth0 150.2.0.2 up
ifconfig eth1 150.3.0.1  up
ifconfig eth2 10.0.0.1 up
ifconfig eth3 100.0.5.1 up
/etc/init.d/zebra start
```

- Le fichier **r5.startup:**

```
ifconfig eth0 150.1.0.2 up
ifconfig eth1 150.2.0.1 up
ifconfig eth2 100.0.4.1 up
/etc/init.d/zebra start
```

- Le fichier **serverWeb.startup:**

```
ifconfig eth2 10.0.0.2 up
route add default gw 10.0.0.1 dev eth2
```

8.2.2.3. Implémentation du protocole RIPv2

Nous implémentons ce protocole avec le logiciel de routage **zebra**.

Pour configurer ce dernier dans notre laboratoire, nous ajoutons dans chaque dossier des routeurs les fichiers suivants:

- **/etc/zebra/deamons:**

```
zebra=yes
bgpd=no
ospfd=no
ospf6d=no
ripd=yes
ripngd=no
```

Ce fichier spécifie la liste des démons à utiliser, nous acceptons zebra et le protocole de routage **rip.**

- **/etc/zebra/ripd.conf**

```
hostname ripd
password zebra
enable password zebra
 router rip
 redistribute connected
 Network 150.0.0.0/8
```

Dans ce fichier nous insérons la configuration suivante:

hostname ripd: identifier le hostname.

password zebra: ajouter le mot passe de zebra.

enable password zebra: activer le mot passe.

router rip: définir le protocole de routage RIP.

redistribute connected: diffuser la table de routage sur les voisins qui utilisent le même protocole de routage RIP.

Network 150.0.0.0/8: déclarer les réseaux.

- **/etc/zebra/zebra.conf**

```
hostname zebra !! identifier le hostname
password zebra !! ajouter le mot passe de zebra
enable password zebra !! activer le mot passe
```

8.2.2.4. Test de connectivité entre le pc et le serveur web

Le Ping entre les pc et le serveur web se passe via les routeurs et se connecte avec succès à cause des tables de routage qui contiennent des chemins vers la destination menée (figure II.33).

```
pc:~# ping -c 3 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=61 time=0.823 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=61 time=1.09 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=61 time=0.868 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2026ms
rtt min/avg/max/mdev = 0.823/0.930/1.099/0.120 ms
```

***Figure II.33**: Test de la connectivité.*

L'analyse du trafic se fait au niveau du routeur 3, nous utilisons toujours la commande tcpdump. A chaque instant, le routeur envoie une requête RIP qui contient un chemin donc il fait la mise à jour automatique. Après que nous tapons une commande PING dans le PC, ce dernier envoie un paquet qui contient le request puis le serveur web (10.0.0.2) répond par un autre paquet de reply, suivant un chemin choisi automatiquement par les routeurs à partir le protocole RIP (figure II.34).

```
r3:~# tcpdump -i eth1
10:43:15.353245 IP 150.3.0.2.520 > 224.0.0.9.520: RIPv2, Response,
length: 84
10:43:30.370763 IP 192.168.1.2 > 10.0.0.2: ICMP echo request, id 3586,
seq 1, length 64
10:43:30.371024 IP 10.0.0.2 > 192.168.1.2: ICMP echo reply, id 3586,
seq 1, length 64
10:43:31.388716 IP 192.168.1.2 > 10.0.0.2: ICMP echo request, id 3586,
seq 2, length 64
10:43:31.389330 IP 10.0.0.2 > 192.168.1.2: ICMP echo reply, id 3586,
seq 2, length 64
10:43:32.398341 IP 192.168.1.2 > 10.0.0.2: ICMP echo request, id 3586,
seq 3, length 64
10:43:32.398599 IP 10.0.0.2 > 192.168.1.2: ICMP echo reply, id 3586,
seq 3, length 64
10:43:35.361090 arp who-has 150.3.0.2 tell 150.3.0.1
10:43:35.361116 arp reply 150.3.0.2 is-at d2:90:43:d2:95:19
10:43:41.360227 IP 150.3.0.2.520 > 224.0.0.9.520: RIPv2, Response,
length: 84
10:43:47.331774 IP 150.3.0.1.520 > 224.0.0.9.520: RIPv2, Response,
length: 84
```

***Figure II.34:** L'analyse du trafic.*

Maintenant, on va tracer le chemin entre pc et serverWeb.

```
pc:~# traceroute 10.0.0.2
traceroute to 10.0.0.2 (10.0.0.2), 64 hops max, 40 byte packets
 1  192.168.1.1 (192.168.1.1)  5 ms  0 ms  0 ms
 2  150.4.0.1 (150.4.0.1)  1 ms  0 ms  0 ms
 3  150.3.0.1 (150.3.0.1)  1 ms  1 ms  1 ms
 4  10.0.0.2 (10.0.0.2)  1 ms  1 ms  1 ms
```

8.2.2.5. Avantage du routage dynamique

Cette fois-ci, nous provoquons la même panne voulue sur le routeur 3 (arrêter l'interface eth2), par la commande suivante :

```
r3:~# ifconfig eth2 down
```

Ensuite, nous testons la connectivité entre le pc et le serveur web en utilisant la commande traceroute 10.0.0.2 au niveau du PC.

```
pc:~# traceroute 10.0.0.2
traceroute to 10.0.0.2 (10.0.0.2), 64 hops max, 40 byte packets
 1  192.168.1.1 (192.168.1.1)  0 ms  0 ms  0 ms
 2  * 192.168.1.1 (192.168.1.1)  42 ms !H *
```

***Figure II.35**: Traçage du chemin.*

Nous constatons que le pc ne se connecte pas avec le serveur web.

Après un moment donné, les routeurs misent à jour les tables de routage et ils prennent un nouveau chemin vers la destination, donc le PC se connecte avec le serveur web.

Le traçage du nouveau chemin entre le pc et le serveur web:

```
pc:~# traceroute 10.0.0.2
traceroute to 10.0.0.2 (10.0.0.2), 64 hops max, 40 byte packets
 1  192.168.1.1 (192.168.1.1)  0 ms  0 ms  0 ms
 2  150.0.0.2 (150.0.0.2)  0 ms  0 ms  0 ms
 3  150.1.0.2 (150.1.0.2)  0 ms  0 ms  0 ms
 4  150.2.0.2 (150.2.0.2)  0 ms  0 ms  0 ms
 5  10.0.0.2 (10.0.0.2)  0 ms  0 ms  0 ms
```

Nous observons que les paquets traversent un autre chemin.

9. Conclusion

Dans ce chapitre on a présenté la virtualisation avec Netkit. Comme exemple, on a présenté le routage statique et dynamique avec Netkit.

Chapitre III: Exemple de virtualisation d'un réseau d'entreprise avec Netkit

1. Introduction

Une des objectifs de notre étude est la réalisation d'un laboratoire virtuel avec Netkit.

Dans ce chapitre, on va mettre en place un réseau d'entreprise possédant un pare-feu (firewall) et une zone démilitarisée (DMZ) et une zone locale sécurisée (LAN).

2. Étude d'un pare-feu

2.1. Principe

Un pare-feu (en anglais firewall) est un logiciel et/ou un matériel permettant de faire respecter la politique de sécurité du réseau, celle-ci définissant quels sont les types de communication autorisés sur ce réseau informatique. Il mesure la prévention des applications et des paquets [28]. Les buts d'un firewall sont multiples. La fonction première d'un firewall est de bloquer les paquets non désirés afin de protéger le réseau local, voire la figure III.1. Les firewalls plus évolués permettent cependant de faire bien plus que du filtrage de paquets, ils peuvent notamment impersonnaliser l'émetteur, transférer les paquets vers une autre machine ou un autre port, équilibrer la charge entre différents serveurs ou transférer les paquets d'une interface vers une autre en suivant des règles établies au préalable.

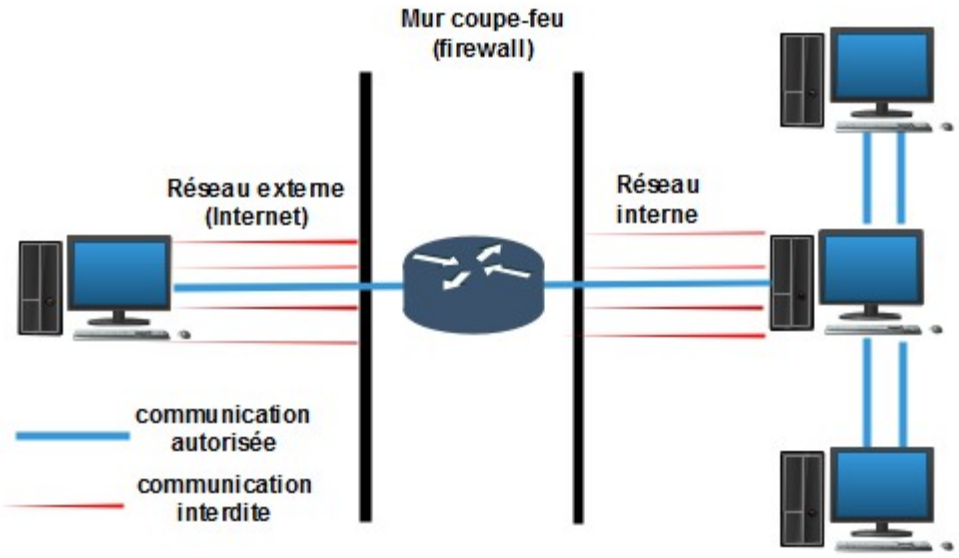


Figure III.1: *Le fonction principale de firewall.*

2.2. L'emplacement de pare-feu

En règle générale, on place le pare-feu entre le réseau externe (typiquement, une connexion internet) et le réseau de l'entreprise. Cependant, les bons administrateurs réseaux préfèrent pratiquer la sécurisation du réseau par niveau. L'idée de la sécurisation par niveau est de découper le réseau

en couches par groupe de machines qui ont le même niveau de sécurité. Typiquement, un réseau se compose de deux zones : une zone démilitarisée (DMZ) et une zone de réseau local. La zone démilitarisée contient les différents serveurs accessibles de l'extérieur du réseau d'entreprise (serveur web, dns, smtp...). Le réseau local contient les serveurs propres à l'entreprise et les machines clientes de l'entreprise. Pour sécuriser un tel réseau, placer un firewall possédant trois cartes réseaux, une connectée à la zone démilitarisée, une connectée au réseau local et la dernière connectée au reste de monde.

2.3. Le pare-feu Netfilter

Netfilter est un sous-système du noyau Linux qui permet d'inspecter les paquets IP et de les filtrer selon certaines règles. Il utilise le mécanisme de listes d'accès qui sont regroupées dans des “chaînes“, elles mêmes contenues dans des tables. Chaque paquet IP qui arrive, traverse, et sort de Linux, est analysé et traité (accepté, rejeté, refusé, modifié, redirigé) en fonction des règles qui lui sont applicables [29].

2.4. Le fonctionnement du pare-feu Netfilter

2.4.1. Le masquage et le translation d'adresses

Ces techniques sont utilisées principalement dans deux cas, pour connecter plus de stations qu'il n'y a d'adresses IP disponibles, ou masquer les adresses réelles des stations derrière le routeur.

2.4.1.1. La translation d'adresses (NAT)

Traducteur d'adresse (*Network Address Translator*) réseau :mécanisme de conversion d'adresses IP privées (internes) en adresses IP publiques (sur internet), mis en place pour répondre à la pénurie d'adresses IP avec le protocole ipv4. Le routeur NAT remplace donc, dans l'en-tête du paquet TCP/IP, l'adresse de l'émetteur par l'adresse IP externe. L'inverse est fait lorsqu'une trame correspondant à cette connexion doit être routée vers l'adresse interne. Le principe du NAT consiste donc à utiliser une passerelle de connexion à internet, possédant au moins une interface réseau connectée sur le réseau interne et au moins une interface réseau connectée à Internet (possédant une adresse IP publique), pour connecter l'ensemble des machines du réseau.

Espaces d'adressages

L'organisme gérant l'espace d'adressage publique est IANA. La RFC 1918 définit un espace d'adressage privé permettant à toute organisation d'attribuer des adresses IP aux machines de son réseau interne sans risque d'entrer en conflit avec une adresse IP publique allouée par l'IANA. Ces

adresses dites non-routables correspondent aux plages d'adresses suivantes:

Classe A : plage de 10.0.0.0 à 10.255.255.255.

Classe B : plage de 172.16.0.0 à 172.31.255.255.

Classe C : plage de 192.168.0.0 à 192.168.255.55.

Toutes les machines sur un réseau interne qui sont connectées à Internet via un routeur et n'ont pas d'adresse IP publique doivent utiliser une adresse dans l'une de ces plages.

Il y a deux types de translation d'adresses (NAT) :

Translation statique :

La translation statique qui consiste à associer un ensemble d'adresses privées internes à un ensemble de même taille d'adresses externes. Ces NAT sont dites *statiques* car l'association entre une adresse interne et son homologue externe est statique (première adresse interne avec première externe…). La table d'association est assez simple, de type un pour un et ne contient que des adresses. Ces NAT servent à donner accès à des serveurs en interne à partir de l'extérieur. Il permet donc de rendre une machine présente dans un LAN ou une DMZ disponible depuis internet. Cela ne va pas dans le sens premier de la création du NAT qui est d'économiser des adresses IPv4 car l'association d'une IP publique vers une IP privée est en un à un.

La figure III.2 montre l'illustration d'un translation statique "un à un", c'est à dire une association IP à IP de tous les ports du routeur coté publique vers une IP privée :

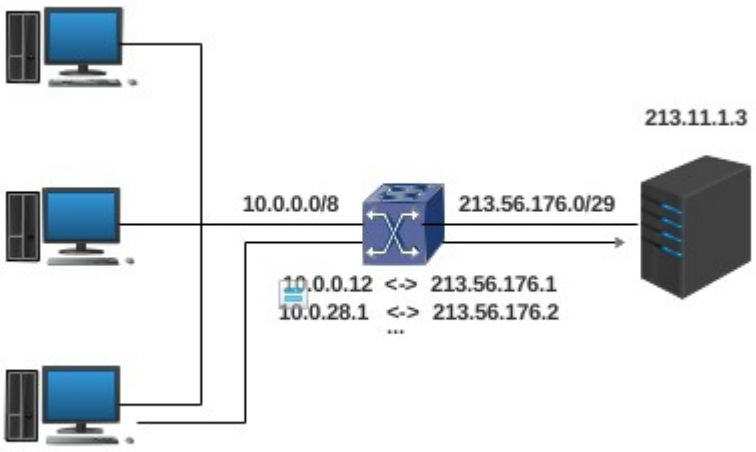


***Figure III.2**: Le mécanisme de translation statique.*

Translation dynamique :

La translation d'adresse dynamique fonctionne elle dans l'autre sens et c'est le but premier de la création du NAT. Si l'on dispose par exemple d'une plage IP de 8 adresses comme 141.15.47.0/29 sur internet mais que l'on a 350 machines dans notre LAN, nous ne pourrons pas les rendre toutes disponibles sur internet en même temps car à force il n'y aurait plus assez d'adresses IPv4. À l'aide de translation dynamique nous pouvons de traduire les 500 adresses IP internes dans le lot des 8 adresses que nous avons sur internet. Le routeur vont utiliser dynamiquement différentes correspondances IP : ports pour faire suivre les paquets et les échanges qui transitent sur les translations interne et externe. C'est la raison pour laquelle le terme de «mascarade IP» est parfois utilisé pour désigner le mécanisme de translation d'adresse dynamique.

2.4.1.2. Le masquage d'adresses IP

L'IP masquerade, consiste à masquer les adresse IP du réseau interne et à n'utiliser que l'adresse du routeur sur le réseau d'interconnexion. Cette technique est intéressante pour connecter tout un réseau local sur l'accès d'un prestataire qui ne fournit qu'une seule adresse IP. Le routeur tient, pour chaque connexion initiée depuis le réseau interne, la correspondance entre les (adresse IP interne, port utilisateur) et (adresse IP externe , port translaté). Le routeur va alors avoir une table de translation qui va être générée via un mécanisme de PAT (Port Address Translation). La figure.III.3 montre le mécanisme de masquage et le translation d'adresses.

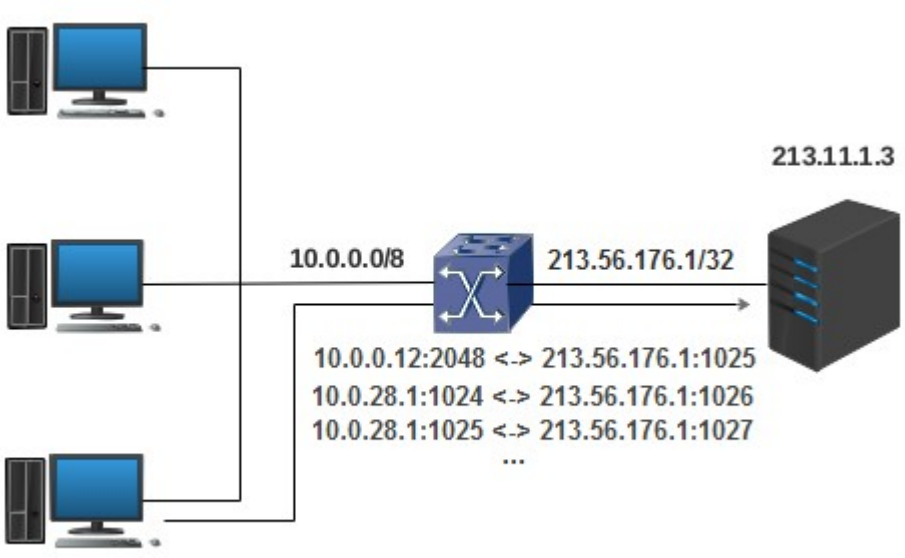


***Figure III.3**: Le mécanisme de masquage et le translation d'adresses.*

2.5. Le filtrage

Un firewall est une machine qui sert comme intermédiaire pour l'accès à un réseau (interne), permettant ou interdisant certain type d'accès avec le respect de la politique de sécurité. Pour atteindre cet objectif, le firewall réalise alors un filtrage des paquets en se basant sur un ensemble de règles définies par l'administrateur de sécurité. Il propose donc un véritable contrôle sur le trafic réseau du système informatique. Il permet d'analyser, de sécuriser et de gérer le trafic réseau. Différentes catégories de firewall on été développées afin de réaliser ces objectifs de sécurité. Le filtrage s'effectue en analysant les champs (source/destinataire) des paquets qui transitent à travers le routeur. On peut ainsi filtrer sur les adresse IP, le protocole, les ports, etc. En général les règles de filtrage sont analysées de manière séquentielle, dès qu'une règle correspond au paquet analysé, elle est appliquée.

2.6. DMZ (zone démilitarisée)

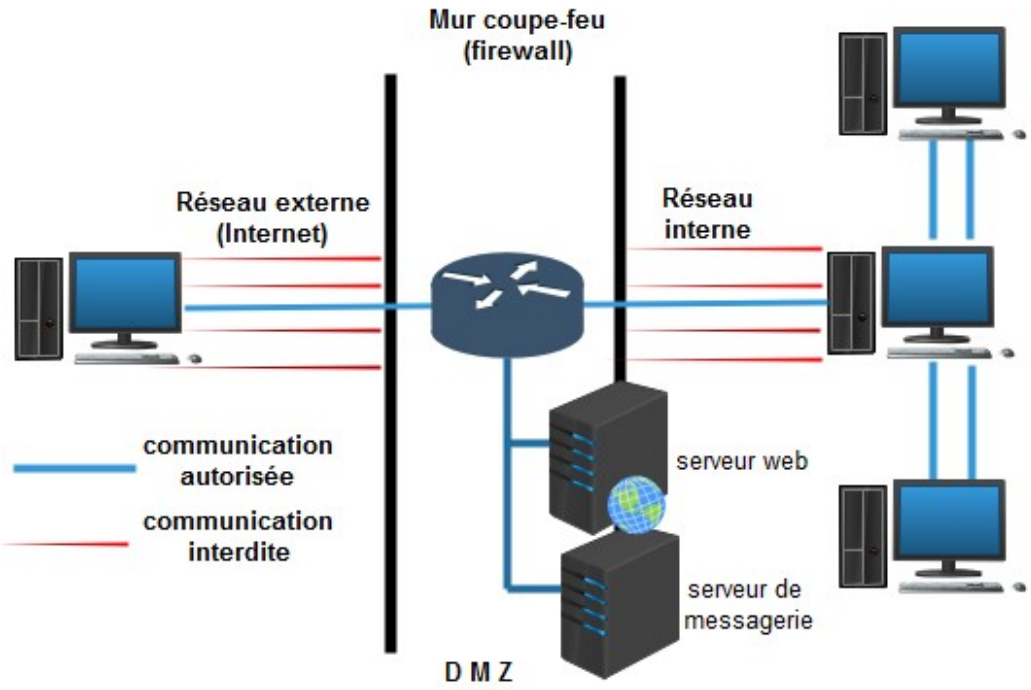


***Figure III.4**: L'architecture de DMZ.*

Les systèmes pare-feu permettent de définir des règles d'accès entre deux réseaux. Les entreprises ont généralement plusieurs sous-réseaux avec des politiques de sécurité différentes. C'est la raison pour laquelle il est nécessaire de mettre en place des architectures des systèmes pare-feu permettant d'isoler les différents réseaux de l'entreprise. Une zone démilitarisée (DMZ) est un sous-réseau se trouvant entre le réseau local et le réseau extérieur. La figure ci-dessous montre la position d‘une DMZ au sein d‘un réseau.

2.7. Iptables

Iptables est l'outil qui permet la manipulation des chaînes et des tables de Netfilter. Il fournit à Linux les fonctions de pare-feu, par filtrer les paquets entrant, sortant ou transitant grâce à des règles définies dans des tables.

2.7.1. Les tables

Il existe trois tables qui vont servir à contenir des règles de filtrage:

2.7.1.1. La table FILTRE

C'est la table par défaut, elle contient toute les règles des filtrage, Cette table contient trois chaînes de base :

- INPUT : (entrée) permet le traitement des paquets immédiatement avant qu'ils ne soient délivrés au processus local.
- OUTPUT : (sortie) Ici, ce ne sont que les paquets émis par l'hôte local qui seront filtrés.
- FORWARD : (transfert) permet le traitement des paquets qui arrivent vers une passerelle, arrivant de l'une des interfaces réseaux et ressortant immédiatement par une autre.

La figure suivante indique la manière dont les paquets traversent le système lors du filtrage de paquets.

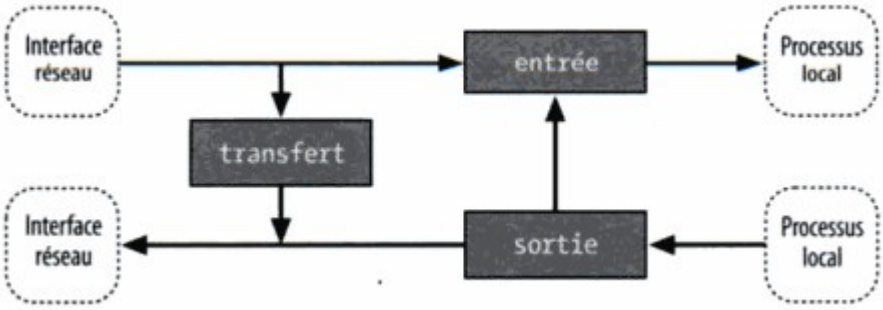


***Figure III.5**: Flux de paquets pour la table filtre [30].*

2.7.1.2. La table NAT

Cette table permet de traduire le champ de l'adresse source ou destination d'un paquet. Cette table contient trois chaînes de base :

- PREROUTING : (pré-routage) elle permet de faire la translation d'adresse de la destination après leur arrivée par l'interface réseau.
- OUTPUT : (sortie) celle-ci va permettre de modifier la destination de paquets générés localement (par la passerelle elle-même).

- POSTROUTING : (post-routage) elle permet de faire de la translation d'adresse de la source juste avant qu'ils ne sortent par l'interface réseau.

La figure suivante indique la manière dont les paquets traversent le système lors de la traduction d'adresses réseaux.

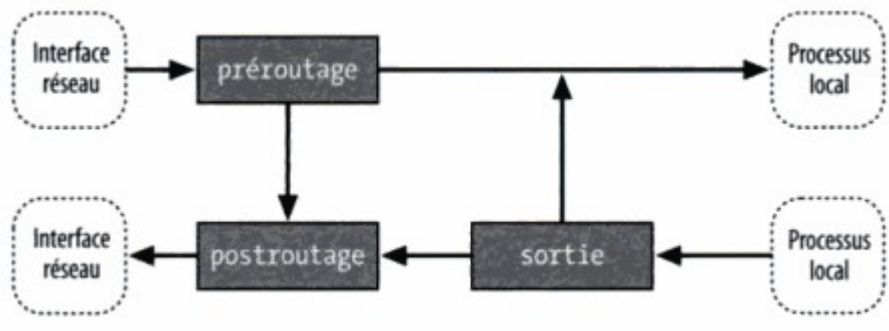


***Figure III.6**: Flux des paquets de la table NAT [30].*

2.7.1.3. La table MANGLE:

La table mangle sert à modifier spécialement les paquets. Cette table possède les cinq chaîne de base :

- PREROUTING : pour modifier les paquets entrants avant le routage.
- OUTPUT : pour modifier les paquets générés localement avant le routage.
- INPUT : pour les paquets entrant dans la case elle-même.
- FORWARD : pour modifier les paquets en cours de route.
- POSTROUTING : pour modifier les paquets comme ils sont sur le point de sortir.

La figure suivante indique la manière dont les paquets traversent lors de la modification des paquets.

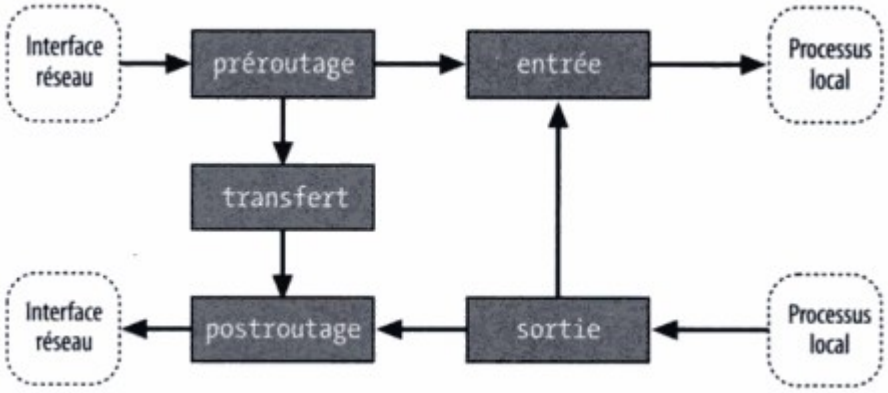


***Figure III.7**: Flux des paquets pour la table MANGLE [30].*

2.7.2. Flux de paquets et chaînes

les paquets sont présentés aux différentes tables et chaînes dans un ordre précis, elle traversent les chaînes et sont présentés aux règles de la chaîne suivant leur ordre d'arrivée. Le point de départ ou d'arrivée peut être une interface réseau ou un processus local, on distingue quatre possibilités, pour chaque possibilité les tableau ci-dessous indiquent l'ordre suivant lequel les paquets sont présentés aux tables et aux chaînes par défaut.

1. Le paquet transite d'une interface réseau vers une autre :

Table	Chaîne
MANGLE	PREROUTING
NAT	PREROUTING
MANGLE	FORWARD
FILTER	FORWARD
MANGLE	POSTROUTING
NAT	POSTROUTING

2. Le paquet transite d'une interface réseau vers un processus local :

Table	Chaîne
MANGLE	PREROUTING
NAT	PREROUTING
MANGLE	INPUT
FILTER	INPUT

3. Le paquet transite d'un processus local vers une interface réseau :

Table	Chaîne
MANGLE	OUTPUT
NAT	OUTPUT
FILTER	OUTPUT
MANGLE	POSTROUTING
NAT	POSTROUTING

4. Le paquet transite d'un processus local vers un autre :

Table	Chaîne
MANGLE	OUTPUT
NAT	OUTPUT
FILTER	OUTPUT
FILTER	INPUT
MANGLE	INPUT

2.7.3. Les chaînes

Sont des ensembles de règles que nous allons écrire dans chaque table. Ces chaînes vont permettre d'identifier des paquets qui correspondent à certains critères. Par défaut, toute table possède des chaînes initialement vides.

2.7.4. Les cibles

La cible indique à la règle que faire avec un paquet qui est parfaitement apparié avec la section correspondance de la règle. Les cibles sont:

- ACCEPT : Permet d'accepter un paquet grâce à la règle vérifiée.
- DROP : Rejet d'un paquet sans message d'erreur si la règle est vérifiée
- REJECT : Rejet avec un retour de paquet d'erreur à l'expéditeur.

D'autres cibles deviennent accessibles, comme SNAT, DNAT, MASQUERADE...

2.7.5. Configuration d'iptables

Maintenant, nous allons décrire la syntaxe de la commande iptables. C'est à l'aide de cette commande que nous configurons les règles du firewall. La plupart des commandes d'iptables peuvent être regroupées en sous-commandes, règles de correspondance et options. Nous allons examiner chacune de ces catégories.

Les commandes d'iptables :

Syntaxe	*Fonction*
iptables -h	aide en ligne
iptables -L	lister les chaînes et règles actives
iptables -N chain	créer une nouvelle chaîne utilisateur

iptables -X chain	supprimer chaîne utilisateur
iptables -F [chain]	vider une chaîne (ou toutes)
iptables -P chain cible	traitement par défaut pour une règle
iptables -A chain règle	ajouter une règle à la chaîne
iptables -I chain [numéro] règle	insérer une chaîne en position numéro
iptables -D chain [numéro] [règle]	effacer une règle
iptables -R chain [numéro] [règle]	remplacer une règle
iptables -C chain	tester un paquet dans une règle
iptables -Z [chain]	remettre à zéro les compteurs

Les options :

Paramètre	*Fonction*
-i interface	Spécifier une interface d'entrée
-o interface	Spécifier une interface de sortie
-t table	table concernée (table filtre par défaut)
-s [!] X.X.X.X/lg	adresse source, longueur du masque le ! signifie la négation
-d [!] X.X.X.X/lg	adresse destination (et longueur du masque)
-p protocole	tcp / udp / icmp / all
--source-port [!] [port[:port]]	port source (ou intervalle de ports)
--destination-port [!] [port[:port]]	port destination (ou intervalle de ports)
--tcp-flags	Spécifier un flag tcp : SYN ACK FIN RST URG PSH ALL NONE
-j cible	ACCEPT/DROP/QUEUE/RETURN/ REDIRECT/MASQUERADE/DNAT/ SNAT/LOG

3. Étude d'un Système de noms de domaine (DNS)

3.1. Principe

Pour pouvoir communiquer, chaque machine présente sur un réseau doit avoir un adresse IP unique. Cependant pour le utilisateur, il est impensable de retenir les adresses IP de chaque ordinateur. C'est pourquoi des mécanisme de résolution de noms ont été mis en place. Le mécanisme de résolution de noms de faire la relation entre nom d'ordinateur et adresse IP. Il faut savoir que l'on peut remplir à la main un fichier, qui s'appelle /etc/hosts, avec les correspondances « adresse IP et nom DNS des machines ». Donc si l'on a un petit réseau local de quelques machines, on peut remplir ce fichier à la main. Mais dans le cas d'un réseau beaucoup plus conséquent, il vaut mieux installer un serveur DNS.

Le serveur DNS est particulièrement utilisé en zone Internet (zone public) et en zone Intranet (zone privée de l'entreprise). Le premier objectif du DNS est de résoudre les noms hiérarchiques pour trouver leurs adresses IP et vice versa. Par exemple, quand taper dans le navigateur l'adresse : www.google.fr, celui-ci va tout d'abord faire une requête à un serveur DNS en lui demandant : Quelle est l'adresse de www.google.fr ? Et le serveur DNS lui donne l'adresse IP et le navigateur va alors se connecter à cette adresse IP et afficher le site, Comme le montre la figure ci-dessous.

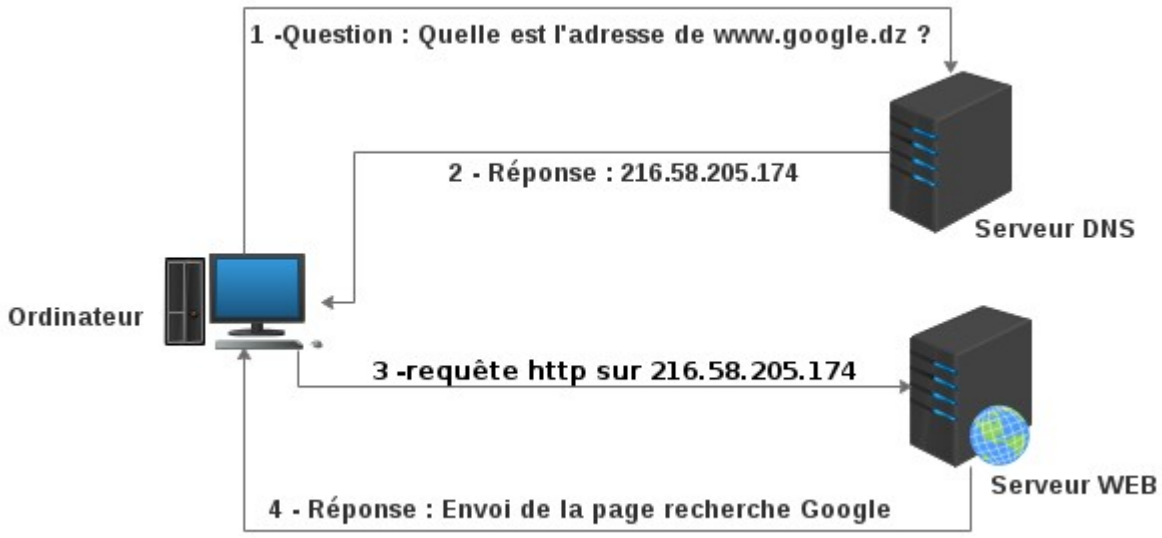


***Figure III.8**: Principe d'une requête DNS.*

3.2. Quelques notions fondamentales

3.2.1. L'espace nom de domaine

Les données ou nom de domaine inscrits dans un serveur DNS sont organisés en arborescence. Chaque nœud de l'arbre appelé domaine, reçoit une étiquette. Le nom de domaine de nœud est la

concaténation de toutes les étiquette des nœuds sur le chemin du domaine racine [30]. Le domaine racine représente le sommet de la hiérarchie, symbolisé par un point. Le nom de domaine est séparé en parties appelées zone.

3.2.2. Notion de zone

La zone est organisation administrative des domaines. C'est une sous-arborescence de la base de données DNS, administrée en tant que entité individuelle distincte [32]. Le rôle d'une zone est principalement de simplifier l'administration des domaines. Le domaine ".com" peut être découpé en plusieurs zones, z1.com, z2.com . . . zn.com. La base de données DNS est distribué sur plusieurs serveurs stockant les différentes zones et du même coup répartir le trafic issu des requêtes DNS envoyées par les client (systèmes et applications). Un fichier de zone est le fichier, stocké sur le disque dur local du serveur DNS, qui contient toutes les informations de configuration d'une zone et des enregistrements de ressources contenus dans celle-ci. Il existe deux zones en fonction de leurs rôles vis-à-vis des requêtes émises par les clients:

- **Zone de recherche directe**

Les zones de recherche directe prennent en charge la fonction principale de DNS. Ils s'appuient sur des noms de domaines DNS. Est une zone qui nous permet de faire des recherches directe, c'est-à-dire trouver le nom d'une machine à partir de son adresse IP. La zone de recherche directe contient généralement des enregistrements de ressources de type A (hôte).

- **Zone de recherche inversée**

Les zones de recherche inversée s'appuient sur le nom de domaine in-addr.arpa. Est une zone "spéciale" qui nous permet de faire des recherches inverses, c'est-à-dire trouver le nom d'une machine à partir de son adresse IP. Pour trouver une machine d'adresse IP 192.168.1.1 par exemple, la requête va être de trouver en premier *in-adr.arpa* (root), puis *192.inadr.arpa*, puis *168.192.in-adr.arpa*, etc. La zone de recherche directe contient généralement des enregistrements de ressources de type PTR (pointeur).

3.2.3. Notion de domaine

Dans le système DNS, on appelle domaine toute arborescence ou sous arborescence se trouvant dans l'espace de noms de domaine [33]. Par exemple .com est un domaine, il contient toute la partie hiérarchique inférieure de l'arbre sous-jacente au nœud .com.

3.3. Type de serveur DNS

3.3.1. Serveur de nom primaire

Un serveur de noms primaire (maître) dispose toutes les données de la zone ou du domaine. Il s'agit d'une zone pouvant être accédée en écriture (ajout) et en modification (mise à jour et suppression).

3.3.2. Serveur de nom secondaire

Un serveur de noms secondaire (esclave) obtient les informations à partir d'un serveur primaire ou d'un autre serveur esclave. Si le serveur de noms primaire n'est pas ou est difficilement accessible pour une raison quelconque, il est possible de retomber sur le serveur secondaire. Ce serveur secondaire est une copie du serveur de noms primaire. La transmission des informations de zone est définie par le terme de « transfert de zone ». Lors du démarrage d'un serveur de noms secondaire, celui-ci établit une connexion vers son serveur de nom maître et démarre le transfert de zone.

3.3.3. Serveur cache

Le serveur cache va permettre de réduire le trafic réseau sortant. Au démarrage, un serveur cache ne peut fournir aucune information, car le serveur sera presque vide, celui-ci va demander aux serveurs externes les réponses afin de remplir sa base DNS. Mais comme le serveur va stocker les réponses, au bout d'un moment, cela va accélérer les réponses DNS, car c'est le serveur local et non pas les serveurs externes qui vont répondre à ces requêtes DNS.

3.4. Le logiciel de serveur de noms le plus utilisé

Le paquetage qui régit l'installation du serveur DNS sous Linux s'appelle BIND. Il existe déjà en plusieurs versions. Il faut aussi dire qu'il existe d'autres paquets qui implémentent le DNS, bind est le plus connu et le plus utilisé sur internet. BIND (Berkeley Internet Name Domain) est un logiciel open source utilisé pour la résolution de noms Internet. Il permet de publier les informations DNS sur Internet et de résoudre les requêtes DNS pour les utilisateurs.

4.Étude d'un service Web

4.1. Principe

Un serveur web est spécifiquement un serveur-multi-service utilisé pour publier des sites web sur Internet ou un Intranet. Est un ordinateur qui stocke les fichiers qui composent les sites web (par exemple les documents HTML, les images…) et qui les envoie à la machine de l'utilisateur qui visite le site. Cet ordinateur est généralement accessible via un nom de domaine. L'expression serveur Web désigne également le logiciel utilisé sur le serveur pour exécuter les requêtes HTTP. HTTP signifie Hypertext Transfer Protocol (protocole de transfert hypertexte) est un protocole de

communication client-server définie la communication entre un client (Les client HTTP les plus connus sont les navigateurs Web) et un serveur contenant les données sur le World Wide Web. ll peut fonctionner sur n'importe quelle connexion fiable, dans les faits on utilise le protocole TCP comme couche de transport. Il utilise alors par défaut le port 80. À chaque fois qu'un navigateur a besoin d'un fichier hébergé sur un serveur web, le navigateur envoie une requête HTTP à le serveur web. Cette requête demande un document (exemple: page HTML, image...). Le serveur cherche les informations, puis il est peut-être amené à interpréter les résultats, pour finalement envoyer la réponse. Cette réponse contient le document demandé, également grâce à HTTP. La communication se déroule de la manière décrite sur le schéma suivant:

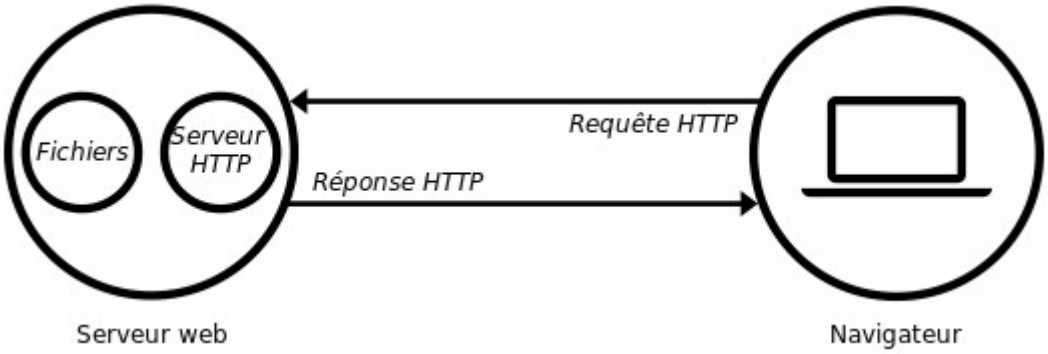


Figure III.9*: Schéma d'une requête HTTP.*

Aujourd'hui, de nombreux serveurs HTTP existent. Parmi les plus quelle utilisés sont: Apache et Nginx, IIS. . .

Apache

Apache est un logiciel de serveur HTTP (le logiciel qui « envoie » des pages .html à un navigateur) produit par l'Apache Software Foundation. C'est un logiciel Open Source le plus populaire la plus couramment utilisée serveur web dans le monde utilisé principalement sur les hébergements Internet. Il est disponible pour presque tous les principaux systèmes d'exploitation de type Unix.

5. Mise en œuvre

Le but de ce Travail est de mettre en place un réseau d'entreprise possédant un pare-feu (firewall) et une zone démilitarisée (DMZ) et une zone locale sécurisée (LAN). La figure suivante présente l'architecture du réseau.

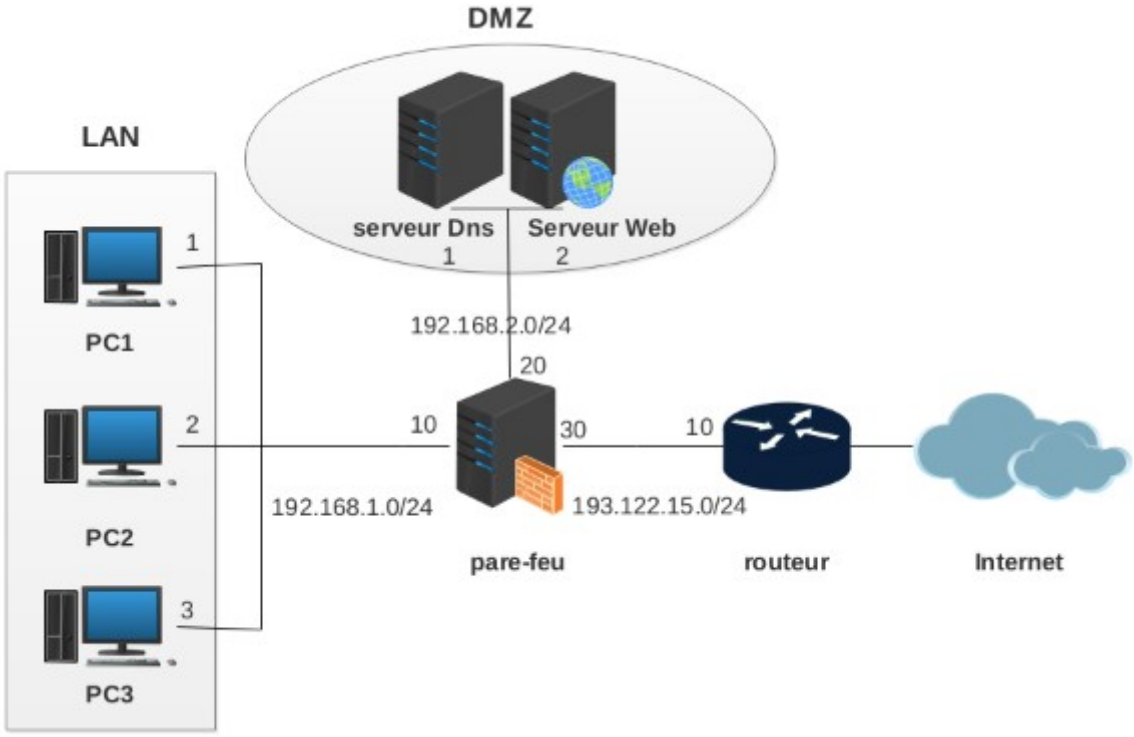


***Figure III.10**: L'architecture du réseau proposé.*

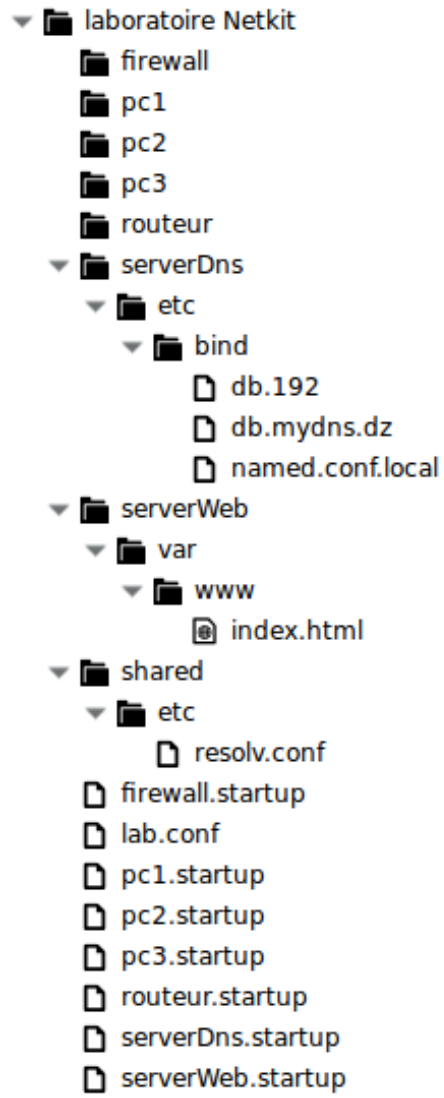
La zone DMZ se compose des serveurs qui sont accessibles du monde extérieur. Dans notre réseau, nous avons deux serveurs, un serveur web (192.168.2.2) qui assure les services web, et un serveur de noms (Serveur Dns, 192.168.2.1) qui assure les services du résolution de noms.

La zone LAN se compose des clients. Les clients se dénomment PC1 et PC2 et PC3 et utilisent une plage d'adresse de 192.168.3.1, 192.168.3.2 et 192.168.3.3. Ils sont autorisés à accéder aux serveurs de la DMZ et ont le droit d'accéder à internet.

Entre le réseau extérieur et le réseau d'entreprise, on va placer un Pare-feu entre la DMZ et le LAN.

L'adresse IP qui est attribuée au le pare-feu pour l'interface réseau exposée au réseau extérieur est fixe (il s'agit de 193.122.15.10). Les clients extérieurs au réseau d'entreprise ne verront aucune autre interface que cette dernière.

Voici l'arborescence des dossiers :



Configuration du fichier **lab.conf**

```
LAB_DESCRIPTION="Mise en place du pare-feu"
LAB_VERSION="0.1"
LAB_AUTHOR="chehrouri et salmi"
LAB_EMAIL="master@gmail.com"
LAB_WEB="http://www.lagh-univ.dz"

routeur[0]="c"
pc1[1]="a"
pc2[2]="a"
pc3[3]="a"
firewall[0]="a"
firewall[1]="b"
firewall[2]="c"
serverDns[0]="b"
serverWeb[0]="b"
routeur[1]=tap,10.0.0.1,10.0.0.2
```

Configuration des fichiers **.startup**

- Le fichier pc1.startup

```
/sbin/ifconfig eth1 192.168.3.1 netmask 255.255.255.0 up
route add default gw 192.168.3.10 dev eth1
```

- Le fichier pc2.startup

```
/sbin/ifconfig eth2 192.168.3.2 netmask 255.255.255.0 up
route add default gw 192.168.3.10 dev eth2
```

- Le fichier pc3.startup

```
/sbin/ifconfig eth3 192.168.3.3 netmask 255.255.255.0 up
route add default gw 192.168.3.10 dev eth3
```

- Le fichier routeur.startup

```
/sbin/ifconfig eth0 193.122.15.10 netmask 255.255.0.0 up
route add -net 192.168.0.0/16 gw 193.122.15.30
```

- Le fichier serverDns.startup

```
/sbin/ifconfig eth0 192.168.2.1 netmask 255.255.255.0 up
route add default gw 192.168.2.20 dev eth0
/etc/init.d/bind9 start
```

- Le fichier serverWeb.startup

```
/sbin/ifconfig eth0 192.168.2.2 netmask 255.255.255.0 up
route add default gw 192.168.2.20 dev eth0
/etc/init.d/apache2 start
```

Après le lancement de laboratoire avec la commande lstart, Il faut ajouter une route statique à la machine hôte pour indiquer comment accéder au réseau créés après les machines virtuelles :

```
sudo route add -net 193.122.15.0/24 gw 10.0.0.2
sudo route add -net 192.168.3.0/24 gw 10.0.0.2
sudo route add -net 192.168.2.0/24 gw 10.0.0.2
```

Maintenant nous pouvons accéder de n’importe quelle machine à toutes les autres machines.

5.1. La configuration de serveur DNS

Le fichier principal de configuration du serveur BIND est named.conf. Il se situe dans le répertoire /etc/bind/. À la fin du ce fichier de configuration, un fichier est inclut (include /etc/bind/named.conf.local), c'est dans ce fichier que se trouverons nos déclarations de zones. On va maintenant ajouter notre zone de résolution de domaine mydns.dz.

```
zone "mydns.dz" {
        type master;
        file "/etc/bind/db.mydns.dz";
};
```

Dans le même fichier /etc/bind/named.conf.local on ajoutons la zone de résolution inverse.

```
zone "168.192.in-addr.arpa" {
     type master;
     file "/etc/bind/db.192";
};
```

Dans la déclaration, **mydns.dz** correspond au nom de la zone, **file** Précise dans quel fichier se trouvent les données de configuration de la zone, **type master** indique que le serveur a autorité complète sur la ou les zones définies en dessous.

Voici le fichier de résolution de zone /etc/bind/db.mydns.dz

```
$TTL  604800
@     IN    SOA   ns.mydns.dz. root.mydns.dz. (
                        2       ; Serial
                   604800          ; Refresh
                    86400          ; Retry
                  2419200          ; Expire
                   604800 )  ; Negative Cache TTL

@                 IN     NS     ns.mydns.dz.
Ns                IN     A      192.168.2.1
www               IN    CNAME   ns
serverweb         IN     A      192.168.2.2
pc1               IN     A      192.168.3.1
pc2               IN     A      192.168.3.2
pc3               IN     A      192.168.3.3
FI0               IN     A      192.168.3.10
FI1               IN     A      192.168.2.20
FI2               IN     A      193.122.15.30
router            IN     A      193.122.15.10
```

On créé le fichier de résolution inverse /etc/bind/192.db:

```
   $TTL    604800
   @ IN  SOA  ns.mydns.dz. root.mydns.dz. (
                        1  ; Serial
                   604800  ; Refresh
                    86400  ; Retry
                  2419200  ; Expire
                   604800  ; Negative Cache TTL

 @      IN       NS       ns.
 1.2    IN       PTR      ns.mydns.com.
 2.2    IN       PTR      serverweb.mydns.com
 1.3    IN       PTR      pc1.mydns.com
 2.3    IN       PTR      pc2.mydns.com
 3.3    IN       PTR      pc3.mydns.com
 10.3   IN       PTR      FI0.mydns.com
 20.2   IN       PTR      FI1.mydns.com
 30.15  IN       PTR      FI2.mydns.com
 10.15  IN       PTR      router.mydns.com
```

SOA: (Start Of Authority) permet de définir les informations relatives à la zone. En l'occurrence le nom du serveur DNS primaire "**ns.mydns.dz.**" et l'adresse mail du contact technique (**root.mydns.dz.** ,en remplaçant le @ de l'émail par un point). Il est compose de plusieurs champs :

Serial: Numéro de version du fichier, sert à la synchronisation. A chaque fois que le fichier de zone est modifier, il faut incrémenter ce numéro.

Refresh: Intervalle de temps pour le rafraîchissement. Temps d'attente du serveur secondaire avant de contrôler si le serveur primaire a modifier de sa zone.

Retry: Temps d'attente du serveur avant de faire à nouveau une demande parce que le serveur primaire n'a pas répondu à une requête. Utilisé par les serveurs secondaires.

Expire: Le temps pendant les informations reçues d'un serveur primaire reste valide. Si ce délai est dépasse et que le serveur secondaire n'a pas pu contacter le serveur primaire, il va alors considérer que les données qu'il a en cache ne sont plus fiable.

Negative Cach TTL: Demande que d'autres serveurs de noms placent en cache les informations pour cette zone pendant au moins cette durée.

Le reste du fichier, les enregistrements de la zone:

IN: Signifie internet, c'est à dire que la zone après le IN est destinée à internet.

A: Permet la conversion d'un nom DNS en adresse IP.

NS: Indique le serveur de noms responsable d'un domaine.

CNAME: Donne un alias à une machine.

PTR: Utilisé dans le fichier de résolution inverse, permet d'associer l'adresse IP à un nom.

Inscription du serveur

Pour indiquer à notre machine que nous avons un serveur DNS et que nous pouvons nous en servir, il faut renseigner l'adresse IP de notre serveur DNS dans le fichier /etc/resolv.conf des MVs :

```
search mydns.dz
Nameserver 192.168.2.1
```

Test de la configuration

La commande « dig » : Elle permet d'interroger directement le serveur DNS de son choix et d'obtenir de nombreuses informations, en plus de la résolution de noms et la résolution inverse.

```
firewall:~# dig pc1.mydns.dz

; <<>> DiG 9.5.0-P2 <<>> pc1.mydns.dz
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17715
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1,
 ADDITIONAL: 1

;; QUESTION SECTION:
;pc1.mydns.dz.              IN    A

;; ANSWER SECTION:
pc1.mydns.dz.         604800     IN    A     192.168.3.1

;; AUTHORITY SECTION:
mydns.dz.        604800     IN    NS    ns.mydns.dz.

;; ADDITIONAL SECTION:
ns.mydns.dz.          604800     IN    A     192.168.2.1

;; Query time: 3 msec
;; SERVER: 192.168.2.1#53(192.168.2.1)
;; WHEN: Sun May 14 06:29:11 2017
;; MSG SIZE  rcvd: 79
```

```
firewall:~# dig -x 192.168.2.2

; <<>> DiG 9.5.0-P2 <<>> -x 192.168.2.2
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23425
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1,
 ADDITIONAL: 0

;; QUESTION SECTION:
;2.2.168.192.in-addr.arpa.  IN    PTR

;; ANSWER SECTION:
2.2.168.192.in-addr.arpa. 604800 IN   PTR
serverWeb.mydns.com.168.192.in-addr.arpa.

;; AUTHORITY SECTION:
168.192.in-addr.arpa. 604800     IN    NS    ns.

;; Query time: 3 msec
;; SERVER: 192.168.2.1#53(192.168.2.1)
;; WHEN: Sun May 14 06:29:45 2017
;; MSG SIZE  rcvd: 92
```

La commande « nslookup » :permet de retrouver l'adresse IP d'une machine à part de son nom DNS, et l'inverse.

```
firewall:~# nslookup 192.168.2.2
Server:         192.168.2.1
Address:   192.168.2.1#53

2.2.168.192.in-addr.arpa    name =
serverWeb.mydns.com.168.192.in-addr.arpa.

firewall:~# nslookup pc2
Server:         192.168.2.1
Address:   192.168.2.1#53

Name: pc2.mydns.dz
Address: 192.168.3.2
```

5.2. La configuration de serveur web

Maintenant que nous avons confirmation du bon fonctionnement du réseau, nous pouvons configurer notre serveur http dans la zone DMZ.

La page par défaut du serveur http est index.html du dossier par défaut /var/www. Nous avons remplacé le code de cette page par le code suivant :

```
<html>
<head> <title> My server web</title> </head>
<body>
<center>
<h2>* Welcome to my page web *</h2>
</center>
</body>
</html>
```

pour connaître le port utilisé pour communiquer avec le serveur http, Nous devons regarder le fichier /etc/services à l'aide de la commande: more /etc/services | grep http .

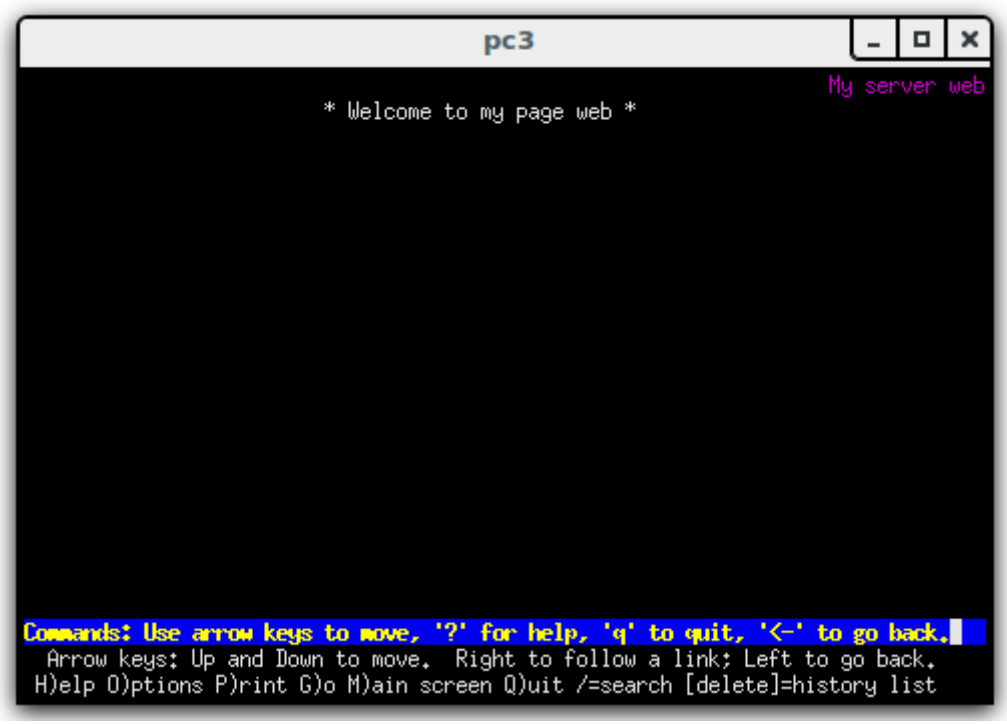
```
www         80/tcp          http        # WorldWideWeb HTTP
```

Le port utilisé est donc le port 80. Nous allons donc modifier ce port par 8080 en éditant le fichier /etc/apache2/ports.conf. Maintenant, il faut redémarrer le service apache2 avec la commande /etc/init.d/apache2 restart.

On peut tester le fonctionnement du serveur apache à partir de la machine pc3 à l'aide de la navigateur de mode texte.

```
pc3:~# lynx serverweb.mydns.dz
```

Voici notre page web:



5.3. Configuration du firewall

Le buts essentiels de pare-feu est protéger le réseau d'entreprise des attaques provenant de réseau extérieur. Une fois le pare-feu complètement configuré, les clients extérieurs à l'entreprise communiqueront exclusivement avec l'interface extérieure du réseau (193.122.15.30), En plus de ne pas divulguer la topologie du réseau d'entreprise.

Nous allons vider les chaînes existantes dans le pare-feu. Nous devons être pleinement confiants des valeurs initiales des règles de pare-feu que nous avons.

```
iptables -t filter -F
iptables -t nat -F
iptables -t mangle -F
```

Maintenant, nous allons modifier les règles par défaut du pare-feu pour rejeter toutes flux entrées, sorties ou traversées de le pare-feu.

```
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

Si on consulte alors la table de routage par défaut (table filtrage) à l'aide de commande iptables -L, on obtient les information suivante:

```
firewall:~# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
```

On observe que la politique par défaut des chaînes est le rejet. On peut s'assurer que cette configuration fonctionne en tentant d'envoyer des requêtes ICMP ping entre le LAN et la DMZ, et entre le réseau externe et LAN.

Depuis pc3 vers serveurWeb:

```
Pc3:~# Ping -c 2 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.

--- 192.168.2.2 ping statistics ---
2 packets transmitted,0 received,100% packet loss,time 1010ms
```

Depuis le routeur externe vers pc2:

```
routeur:~# Ping -c 2 192.168.3.2
PING 192.168.3.2 (192.168.3.2) 56(84) bytes of data.

--- 192.168.3.2 ping statistics ---
2 packets transmitted,0 received,100% packet loss,time 1009ms
```

Aucune communication n'est alors possible entre les zones DMZ, LAN et Internet. Il est donc nécessaire d'ajouter des règles qui permettent un certain accès entre les machines.

5.3.1. Étape : filtrage des paquets entre LAN et DMZ

Il nous faut donc autoriser les trafics ICMP (requis pour le ping), HTTP (sur le port TCP 8080) et DNS (sur le port UDP 53).

Autoriser des paquets ICMP

On configure le pare-feu pour les ordinateurs du réseau locale peuvent envoyer des paquets ICMP à l'interface LAN du pare-feu (eth0, 192.168.3.10), et le pare-feu peut leur répondre.

```
iptables -A INPUT -p icmp -i eth0 -j ACCEPT
```

On fait la même chose pour l'interface DMZ du pare-feu (eth1, 192.168.2.20).

```
iptables -A INPUT -p icmp -i eth1 -j ACCEPT
```

Ici, nous ajoutons cette règle pour permettre au pare-feu d'envoyer des paquets ICMP.

```
iptables -A OUTPUT -p icmp -j ACCEPT
```

Pour permettre le passage des paquets ICMP entre DMZ et LAN, il faut configurer la chaîne FORWARD pour qu'elle route correctement ce type de paquets.

```
iptables -A FORWARD -s 192.168.3.0/24 -d 0.0.0.0/0 -p icmp --icmp-type
echo-request -j ACCEPT
iptables -A FORWARD -s 0.0.0.0/0 -d 192.168.3.0/24 -p icmp --icmp-type
echo-reply -j ACCEPT
```

La première règle permet d'envoyer des paquets ICMP (echo-request) du LAN à tous les réseaux après le pare-feu (DMZ et réseau externe), mais la deuxième règle permet uniquement les réponses, nous indiquons que les paquets de type ICMP pouvant passer sont uniquement des echo-reply. Cela signifie que le réseau externe pourra répondre à une requête ICMP venant de LAN mais ne pourra envoyer un ping à destination de LAN.

On peut s'assurer que cette configuration fonctionne en essayant d'envoyer des requêtes ping ICMP entre le LAN et la DMZ et l'inverse.

```
pc1:~# ping -c 2 192.168.2.2
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=63 time=0.682 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=63 time=0.395 ms

--- 192.168.2.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1018ms
rtt min/avg/max/mdev = 0.395/0.538/0.682/0.145 ms

serverWeb:~# ping -c 2 192.168.3.1
PING 192.168.3.1 (192.168.3.1) 56(84) bytes of data.

--- 192.168.3.1 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1015ms
```

Autoriser l'accès au DNS

Les règles suivantes pour que le pare-feu accède au service DNS dans DMZ:

```
iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
iptables -A INPUT -p udp --sport 53 -j ACCEPT
```

Les règles suivantes permettent aux ordinateurs LAN d'accéder au serveur DNS:

```
iptables -A FORWARD -s 192.168.3.0/24 -p udp --dport 53 -j ACCEPT
iptables -A FORWARD -d 192.168.3.0/24 -p udp --sport 53 -j ACCEPT
```

Autoriser l'accès au serveur HTTP

On souhaite maintenant permettre l'accès au serveur http du DMZ pour les machines du LAN. L'adresse de notre serveur http est 192.168.2.2 et le port utilisé est 8080. Pour que les machines du LAN puissent accéder à ce service en utilisant l'adresse IP et le numéro de port de la machine, il faut les règles suivante:

```
iptables -A FORWARD -i eth0 -o eth1 -s 192.168.3.0/24 -d 192.168.2.2 -p
tcp --dport 8080 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -s 192.168.2.2 -d 192.168.3.0/24 -p
tcp ! --syn --sport 8080 -j ACCEPT
```

Ces règles autorisent le trafic depuis LAN vers le port 8080 serveur http du DMZ, et le trafic du port 8080 du serveur http à LAN (autoriser seulement le passage de paquets ! --syn ,alors serveur http ne pourra pas demander de connexion http). Pour nous connecter avec succès au serveur http de dmz depuis lan. Nous utilisons alors le navigateur en mode texte « lynx » avec la commande lynx 192.168.2.2:8080.

Maintenant nous avons utilisés la translation d'adresse (NAT) pour masquer les adresses du LAN et à n'utiliser que l'adresse du routeur (coté DMZ et coté externe).

```
iptables -t nat -A POSTROUTING -s 192.168.3.0/24 -j MASQUERADE
```

Nous pouvons nous assurer que ces configurations fonctionnent en essayant de connecter le serveur HTTP à partir du LAN (pc3), puis nous surveillons le trafic sur le serveur http.

```
pc1:~# lynx serverweb:8080
```

On obtient :

Le capture de trafic sur le serveur http:

Source	Destination	Protocol	Length	Info
192.168.2.20	192.168.2.2	TCP	74	38837→8080 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=1172380 TSecr=(
192.168.2.2	192.168.2.20	TCP	74	8080→38837 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=117(
192.168.2.20	192.168.2.2	TCP	66	38837→8080 [ACK] Seq=1 Ack=1 Win=5840 Len=0 TSval=1172380 TSecr=1170952
192.168.2.20	192.168.2.2	HTTP	289	GET / HTTP/1.0 [Packet size limited during capture]
192.168.2.2	192.168.2.20	TCP	66	8080→38837 [ACK] Seq=1 Ack=224 Win=6864 Len=0 TSval=1170952 TSecr=1172380
192.168.2.2	192.168.2.20	HTTP	484	HTTP/1.1 200 OK [Packet size limited during capture]
192.168.2.2	192.168.2.20	TCP	66	8080→38837 [FIN, ACK] Seq=419 Ack=224 Win=6864 Len=0 TSval=1170952 TSecr=117238(
192.168.2.20	192.168.2.2	TCP	66	38837→8080 [ACK] Seq=224 Ack=419 Win=6912 Len=0 TSval=1172380 TSecr=1170952
192.168.2.20	192.168.2.2	TCP	66	38837→8080 [FIN, ACK] Seq=224 Ack=420 Win=6912 Len=0 TSval=1172380 TSecr=117095(
192.168.2.2	192.168.2.20	TCP	66	8080→38837 [ACK] Seq=420 Ack=225 Win=6864 Len=0 TSval=1170953 TSecr=1172380

On peut voir que l'adresse source qui apparaît est celle de l'interface de pare-feu connectée à la DMZ et non à l'adresse de la machine pc3 du LAN.

5.3.2. Étape : filtrage des paquets entre LAN et réseau externe

Autoriser des paquets ICMP et l'accès DNS

Nous n'avons pas besoin d'ajouter des règles pour permettre le passage des paquets ICMP et l'accès au service DNS puisque ces règles ont déjà été ajoutées à l'étape précédente (LAN ↔ DMZ).

Autoriser l'accès au serveur HTTP

Les règles ci-dessous permettent d'accéder aux services http externes pour les machines LAN avec le port 80.

```
iptables -A FORWARD -i eth0 -o eth2 -s 192.168.3.0/24 -p tcp --dport 80
-j ACCEPT
iptables -A FORWARD -i eth2 -o eth0 -d 192.168.3.0/24 -p tcp ! --syn
 --sport 80 -j ACCEPT
```

5.3.3. Étape : filtrage des paquets entre DMZ et réseau externe

Autoriser des paquets ICMP

Pour permettre aux machines DMZ d'envoyer des paquets ICMP au réseau externe et les machines sur le réseau externe pour envoyer des réponses uniquement, les règles suivantes doivent être ajoutées:

```
iptables -A FORWARD -s 192.168.2.0/24 -d ! 192.168.3.0/24 -p icmp
--icmp-type echo-request -j ACCEPT
iptables -A FORWARD -d 192.168.2.0/24 -p icmp --icmp-type echo-reply -j
ACCEPT
```

Autoriser l'accès au services DNS

```
iptables -A FORWARD -p udp --dport 53 -j ACCEPT
iptables -A FORWARD -p udp --sport 53 -j ACCEPT
```

Autoriser l'accès au serveur HTTP

Nous voulons permettre au LAN d'accéder au serveur DMZ HTTP. L'accès n'est pas directement mais seulement l'adresse IP de l'interface pare-feu côté externe (eth2, 193.122.15.30) est connue. Lorsque une machine externe connecter au serveur http de DMZ, il effectue une requête http au DMZ. La DMZ répond à l'émetteur, le pare-feu effectuant une translation d'adresse et de port pour masquer l'origine.

On autorisons l'échange http par les règles suivante :

```
iptables -A FORWARD -i eth2 -o eth1 -d 192.168.2.2 -p tcp --dport 8080
-j ACCEPT
iptables -A FORWARD -i eth1 -o eth2 -s 192.168.2.2 -p tcp --sport 8080
-j ACCEPT
```

Nous configurons alors la règle effectuant la translation d'adresse et de port à partir dmz vers l'extérieur. Il s'agit de SNAT sachant que c'est l'adresse source qui est modifiée et de POSTROUTING pour effectuer le routage avant la translation. Nous modifions donc la source et le

port au passage du pare-feu :

```
iptables -t nat -A POSTROUTING -o eth2 -s 192.168.2.2 -p tcp –sport 8080
-j SNAT --to-source 193.122.15.30:80
```

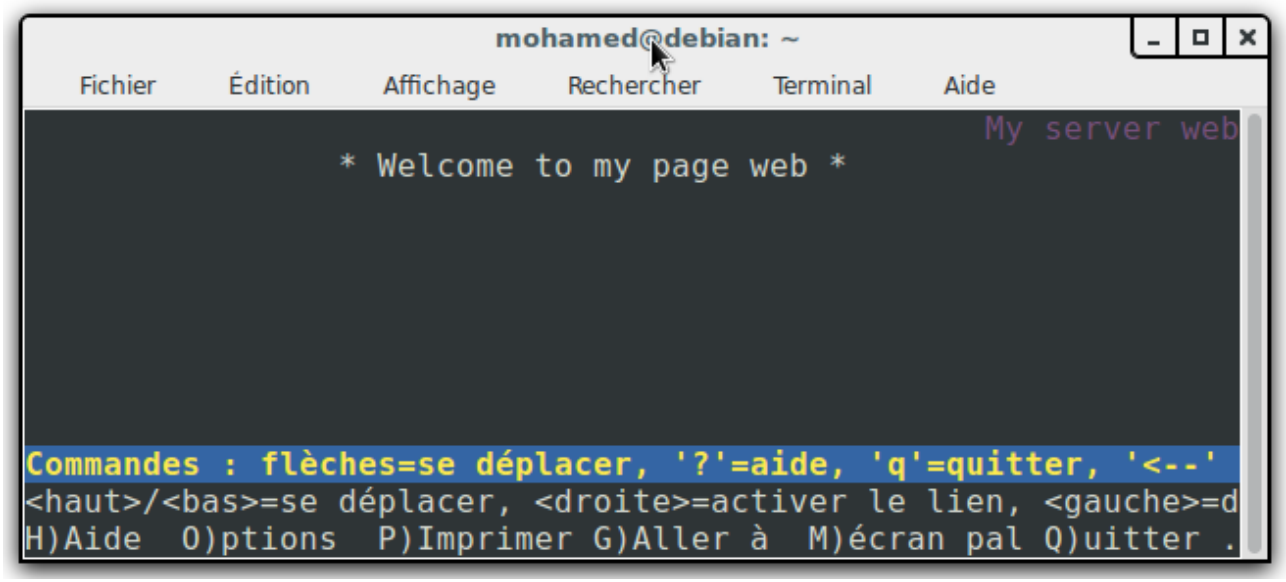
Nous ajoutons maintenant la règle inverse pour que tous paquets provenant d'extérieur à destination du pare-feu sur le port 80 soit envoyé au dmz sur le port 8080. Il s'agit donc de DNAT, la destination étant modifiée, et de PREROUTING, la translation effectuée avant le routage.

```
iptables -t nat -A PREROUTING -i eth2 -p tcp –dport 80 -j DNAT
--to-destination 192.168.2.2:8080
```

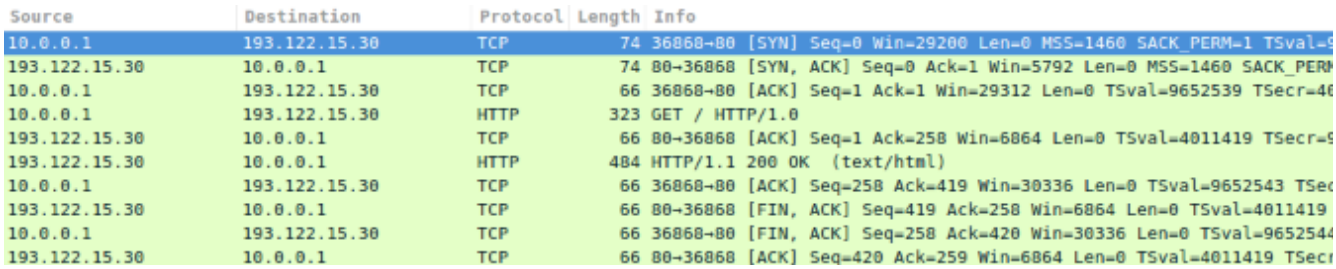
Nous pouvons désormais accéder au serveur http du dmz à partir notre machine hôte par l'adresse du pare-feu coté externe 193.122.15.30 (pas l'adresse de serveur http). Entre temps, nous capturons le trafic.

```
mohamed@debian:~$ lynx 193.122.15.30
```

On obtient :



Le capture de trafic :



Source	Destination	Protocol	Length	Info
10.0.0.1	193.122.15.30	TCP	74	36868→80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=9
193.122.15.30	10.0.0.1	TCP	74	80→36868 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM
10.0.0.1	193.122.15.30	TCP	66	36868→80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=9652539 TSecr=40
10.0.0.1	193.122.15.30	HTTP	323	GET / HTTP/1.0
193.122.15.30	10.0.0.1	TCP	66	80→36868 [ACK] Seq=1 Ack=258 Win=6864 Len=0 TSval=4011419 TSecr=9
193.122.15.30	10.0.0.1	HTTP	484	HTTP/1.1 200 OK (text/html)
10.0.0.1	193.122.15.30	TCP	66	36868→80 [ACK] Seq=258 Ack=419 Win=30336 Len=0 TSval=9652543 TSec
193.122.15.30	10.0.0.1	TCP	66	80→36868 [FIN, ACK] Seq=419 Ack=258 Win=6864 Len=0 TSval=4011419
10.0.0.1	193.122.15.30	TCP	66	36868→80 [FIN, ACK] Seq=258 Ack=420 Win=30336 Len=0 TSval=9652544
193.122.15.30	10.0.0.1	TCP	66	80→36868 [ACK] Seq=420 Ack=259 Win=6864 Len=0 TSval=4011419 TSecr

Nous remarquons l’échange s’effectue correctement.

6. Conclusion

Au cours de ce chapitre nous avons présenté une création de réseau virtuelle avec Netkit dans une seule machine physique.

Notre réseau comprendre la création de réseau d'entreprise possédant un pare-feu (firewall) et une zone DMZ (qui contient un serveur web et un serveur dns) et une zone locale sécurisée LAN.

Conclusion générale

La virtualisation est un domaine en pleine croissance, qui évolue très rapidement. Les utilisateurs peuvent s'en servir pour différents usages, aux besoins de leur fin. Les différentes solutions de virtualisation existantes utilisent des technologies variées.

Dans ce travail, nous avons étudié la virtualisation avec Netkit.

Il nous a permis de:

- Comprend Netkit et son utilisation.
- La notion du noyau d'un système linux.
- La création de laboratoires virtuels avec Netkit.

D’après notre étude, on peut tirer les avantages suivants :

- Netkit est très facile à apprendre et à utiliser.
- La machine virtuelle sous Netkit exploite un système GNU/Linux, donc elle a les avantages de GNU/Linux, puisque elle supporte la plupart des technologies de cet environnement.
- Netkit est Légé et facile à installer et à exécuter.
- Netkit est livré avec un ensemble d'expériences de réseaux prêts à être utilisé qui mettent en œuvre des études des protocoles de routage (TCP, RIP, BGP, etc) à des services de niveau application (DNS, e-mail, etc).
- Netkit permet de faire des expériences des réseaux complexes facilement.

Enfin, nous souhaitons que :

- Netkit sera utilisé dans les salles de TP de notre département pour profiter de sa puissance.
- La création d'un laboratoire virtuel utilisant Netkit pour réaliser des émulations.

Références

[1] **EXAKIS | NANTES**, 'Point sur la virtualisation', mathieuc@exakis.com, 2013.

[2] http://www.marche-public.fr/Terminologie/Entrees/virtualisation.htm , Le dernier accès Mars 2017.

[3] http://democritique.org/IT/Virtualisation.svg.xhtml , Le dernier accès Mars 2017.

[4] http://www.vmware.com/fr/solutions/virtualization.html , Le dernier accès Mars 2017.

[5]https://www.ibisc.univevry.fr/~petit/Enseignement/AdminSystem/Virtualisation/Virtualisation/vgrid.html, Le dernier accès Mars 2017.

[6] **S. François**, “La Virtualisation”, Projet de recherche et communication scientifique, université libre de Bruxelles, Université d’Europe, 2010.

[7] **Popek et Goldberg**, 'Formal Requirements for Virtualizable Third Generation Architectures',ftp://ftp.cis.upenn.edu/pub/cis700-6/public_html/04f/papers/popek-goldberg-requirements.pdf, 1974.

[8] **équipes informatiques des établissements d'enseignement agricole publics, des membres du réseau des DRTIC et Eduter-Cnerta**, 'Virtualisation de l’architecture serveurs Pour le système d’information de l’EPLEFPA', http://support.eduter-cnerta.fr/fileadmin/user_upload/pdf/livre_blanc_virtualisation.pdf, Novembre 2013

[9] **E. RAMAT**, "Introduction à la simulation:principaux concepts", Laboratoire d’Informatique du Littoral – Calais, doczz.fr/doc/2652757/introduction-à-la-simulation---principaux-concepts-plan, Février 2017

[10] "Émulation", https://fr.wikipedia.org/wiki/Émulation, 2017 , Le dernier accès Février 2017

[11] **Equipe Administration Système Smile**, "Livre blanc Virtualisation Version 1.0", www.smile.fr

[12] **Alexis de Lattre, Rémy Garrigue, Tanguy Ortolo, Adrien Grand**, "Formation Debian GNU/Linux", http://formation-debian.via.ecp.fr, Le dernier accès Avril 2017

[13] **YAKETE-OUALIKETTE**, "Virtualisation d'un réseau informatique avec Netkit", Mémoire de fin de cycle Licence ASRALL, 2011 -2012

[14] **G. Di Battista, M. Patrignani, M. Pizzonia, M. Rimondini**, "the poor man's system for experimenting computer networking", www.netkit.org, Oct 2014

[15] **Massimo Rimondini**, "Netkit: Easy Emulation of Complex Networks on Inexpensive Hardware", www.netkit.org

[16] **Fabio Martignon**, "TP – Netkit", Université Paris-Sud Laboratoire de Recherche en Informatique, 2013

[17] **A. Quidelleur,** Le logiciel Netkit Installation et utilisation, Université de Paris-Est Marne-la-Vallée

[18] **Massimo Rimondini**, "Interdomain Routing Policies in the Internet: Inference and Analysis", UNIVERSITÀ DEGLI STUDI ROMA TRE, 2007

[19] **Jeff Dike**, "User Mode Linux", Prentice Hall , 2006

[20] **Jeff Dike and others**, "ubuntu manuals", http://manpages.ubuntu.com/manpages/trusty/man1/uml_switch.1.html, Le dernier accès Février 2017

[21] http://wiki.netkit.org/index.php/Labs_Official, Le dernier accès Mars 2017

[22] "Découvrir UML, ou comment mettre des Linux dans son Linux",http://docplayer.fr/5131028-Decouvrir-uml-ou-comment-mettre-des-linux-dans-son-linu.html , 24 août 2008, Le dernier accès Mars 2017

[23] **Mechraoui imane, Hegga meriem**, "Virtualisation avec UML", mémoire UNIVERSITÉ AMMAR TELIDJI LAGHOUAT, 2015/2016

[24] **BOURAS Mohamed**,"Etude des outils de simulation des réseaux (OMNeT++, Netkit)", mémoire UNIVERSITÉ AMMAR TELIDJI LAGHOUAT, 2013/2014

[25] **Maurizio Pizzonia,Massimo Rimondini**, "Netkit: Easy Emulation of Complex Networks on Inexpensive Hardware",

[26] **Erik Allais, Anatolie Golovco**, "Routage", master informatique, 2006-2007

[27] **Jean Robert hountomey**, "routage statique", AFNOG

[28] "Pare-feu (informatique)", https://fr.wikipedia.org/wiki/Pare-feu_%28informatique%29, Le dernier accès Mars 2017

[29] "Notes de configuration pour IPTABLES et NETFILTER", https://jeanbonnard.wordpress.com/notes-de-configuration-pour-iptables-et-netfilter/, Le dernier accès Mai 2017

[30] **GREGOR N. PURDY**, "Les iptables LINUX", https://books.google.fr/books?isbn=2841772586 ,2004, Le dernier accès Mai 2017

[31] **Stéphane Gill**, Résolution de nom avec Bind, Stephane.Gill@CollegeAhuntsic.qc.ca, 2004

[32] **Abdel YEZZA**, "DNS : Revue Rapide", abdel.yezza.free.fr/introduction/dns/dns_basics.pdf

[33] "[DNS] serveur DNS sous windows server – cours", Portail de la Maintenance des Supports Informatiques et Réseaux | Cours, TP, Tutoriels, Formations , http://reseauwdakchi.blogspot.com/2015/02/dns-serveur-dns-sous-windows-server.html, le dernier accès Mia 2017