

# VIMINAL (\*)

## Virtual Model for Ip Network Architecture Lab

(\*) The **Viminal Hill** (Latin *Collis Viminalis*, Italian *Viminale*) is the smallest and least important of the famous [seven hills](http://en.wikipedia.org/wiki/Seven_hills_of_Rome) of [Rome](http://en.wikipedia.org/wiki/Rome), and as such always referred to as *collis* rather than *mons*. (wikipedia : [http://en.wikipedia.org/wiki/Viminal\\_Hill](http://en.wikipedia.org/wiki/Viminal_Hill))

### Introduction

VIMINAL (Virtual Model for Ip Network Architecture Lab) plate-form is an autonomous network and system lab environment. Available on a liveCD, it offers network models on which you have extended rights. It integrates all the materials needed to securely play system and IP network labs on common computers. The main goal is to play such labs, with no installation or configuration modifications on your computer. VIMINAL liveCD is based on VNUML (Virtual Network User-Mode Linux <http://jungla.dit.upm.es/~vnuml/>), a neat project of Telematics Engineering Departement (DIT) of the Technical University of Madrid (UPM).

### What is VNUML ?

*"VNUML stands for Virtual Network User Mode Linux. It is is an open-source general purpose virtualization tool designed to quickly define and test complex network simulation scenarios based on the great User Mode Linux (UML) virtualization software. It has been initially developed in the context of Euro6IX research project to simulate IPv6 IX scenarios based on Linux and zebra/quagga routing daemons (see our recent article on IEEE Comms. Magazine). However, it is a useful tool that can be used to simulate general Linux based network scenarios. VNUML is aimed to help in testing network applications and services over complex testbeds made of several nodes (even tenths) and networks inside one Linux machine, without involving the investment and management complexity needed to create them using real equipment. VNUML tool is made of two main components: the VNUML language used for describing simulations in XML; and the interpreter of the language (vnuml command), that builds and manages the simulation, hiding all UML complex details to the user. VNUML has been developed by the Telematics Engineering Department (DIT) of the Technical University of Madrid (UPM) in Spain. This software is released under GNU Public Licence. It has been developed with the partial support from the European Commission under the Euro6IX IST research project."*

More information can be found on the [VNUML web site](#)

### Authorisations and privileges considerations

- VIMINAL liveCD boots the computer under the « viminal » user, with low level rights. So

user viminal has no privilege on the real host. Super-user account (root) having a new random password on each liveCD reboot, there's no simple way to gain super user privileges on the real computer.

- On each VNUML virtual machine you have super-user privileges. I know root with no password, it's bad ! But remember user-mode linux virtual machines are sandbox with no particular privilege on the real host.

## Environments

- VIMINAL is based on Gentoo. Why ? First because that's my usual distribution. For me there's no bad GNU/Linux distribution. The good one, is the distribution you use, you know how to configure and you are able to easily update. Second Gentoo has a good liveCD generator tool named Catalyst2. With Catalyst2 creating and updating dedicated liveCD is two conf files and some basic bash scripts.
- As a convenience VIMINAL liveCD embeds two graphical desktops. The host desktop is KDE, the virtual machine desktop is Fluxbox. We choose two different desktop window managers to easily distinguish real host desktop and virtual machine desktop.
- Two modes are available to have direct virtual machine access : Console mode with SSH and graphic desktop with VNC. VNC is used instead of a direct X-window export because VNC has a native session mode. You can leave the virtual machine desktop, and find it in the same state when you reconnect.

## Getting started

First of all, you have to boot your computer on the liveCD. Depending on your computer configuration, you may have to change the boot sequence order in the bios (see your computer notice to find how to change your boot sequence order if not booting first on cdrom). Boot step automatically starts linux, and ask your keyboard preference for the tty terminal (choose you keyboard preference by typing your two letters iso country code if you don't have an 'us' keyboard which is the default).

```

:: Scanning for sata_qstor...sata_qstor loaded.
:: Scanning for ahci...ahci loaded.
:: Scanning for ata_piix...ata_piix loaded.
:: Scanning for dm-mod...dm-mod loaded.
:: Scanning for dm-mirror...dm-mirror loaded.
>> Activating udev
>> Making tmpfs for /newroot
>> Attempting to mount CD:- /dev/hdc
>> CD medium found on /dev/hdc
>> Loading keymaps
Please select a keymap from the following list by typing in the appropriate
name or number. Hit Enter for the default "us/41" US English keymap.

 1 azerty   7 cf       13 es      19 il      25 mk      31 ru      37 trf
 2 be       8 croat    14 et      20 is      26 nl      32 se      38 trq
 3 bg       9 cz       15 fi      21 it      27 no      33 sg      39 ua
 4 br-a    10 de      16 fr      22 jp      28 pl      34 sk-y     40 uk
 5 br-l    11 dk      17 gr      23 la      29 pt      35 sk-z     41 us
 6 by      12 dvorak  18 hu      24 lt      30 ro      36 slovene  42 wangbe

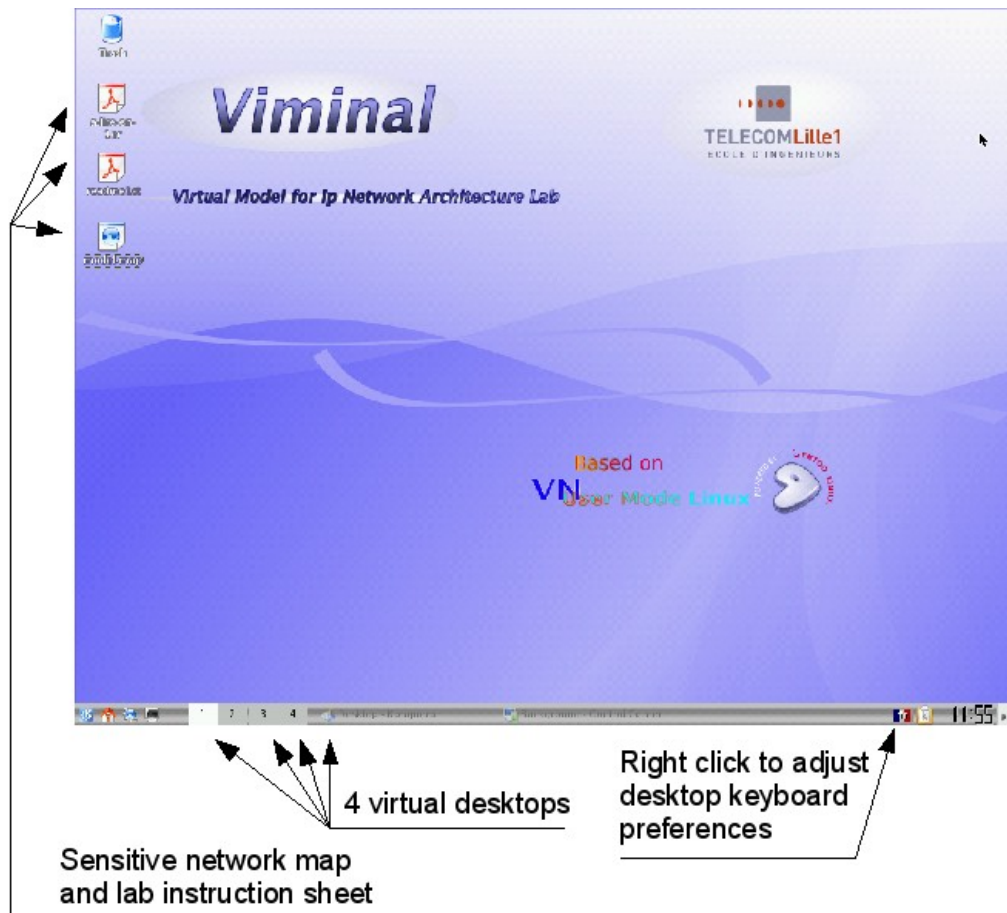
<< Load keymap (Enter for default): fr_

```

## Playing the lab

### The VIMINAL desktop

The liveCD automatically logs the viminal user and starts a graphical desktop session (KDE desktop). At the end of the boot step you must have a graphic screen looking like this one :



All what you need to play the lab is directly available on this desktop :

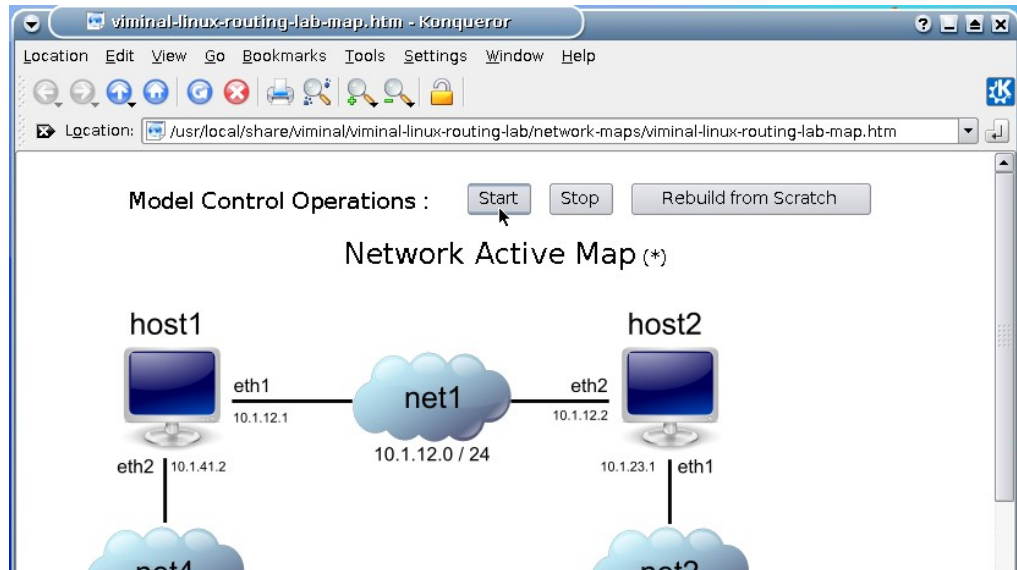
- this read me first instruction file,
- the sensitive network map to interact with the network model (a graphical representation of the network model on which you can do start, stop control operations, and have direct access on the virtual machines of the model),
- the lab instruction sheet to play the lab.

**Nota** : you can adjust your desktop keyboard preferences by right clicking on the flag in right bottom of the screen (default is french azerty keyboard).

## Starting the network model

There's two ways to start the network model :

- The simplest one : just open the sensitive network model map, and click on the “start” button.
- Manually : open a shell terminal (konsole on KDE), and manually launch the `/usr/local/share/viminal/name-of-the-lab/model-control-scripts/name-of-the-lab-start.sh` script

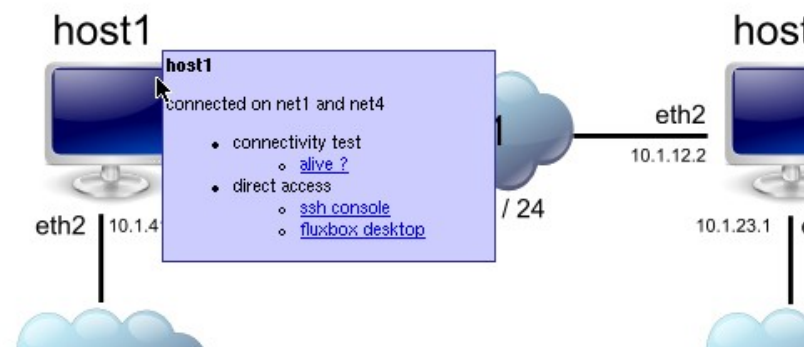


This starts all the networks and virtual machines of the model. It can be quite long (depending of your computer capacity) and open a lot of consoles. Be patient and read the lab instruction sheet during this step.

When the model is up, your graphical desktop displays a set of consoles (for each machine of the model console #0 displays log messages of the virtual machine, and console #1 is a direct login console), which can be ugly for the usability. You can switch on another virtual desktop (remember you have 4 virtual desktops on KDE, so you can switch between them by a simple mouse click in the taskbar). Or you can minimize all these consoles by right clicking on the console reference in the taskbar and then select "Minimize All".

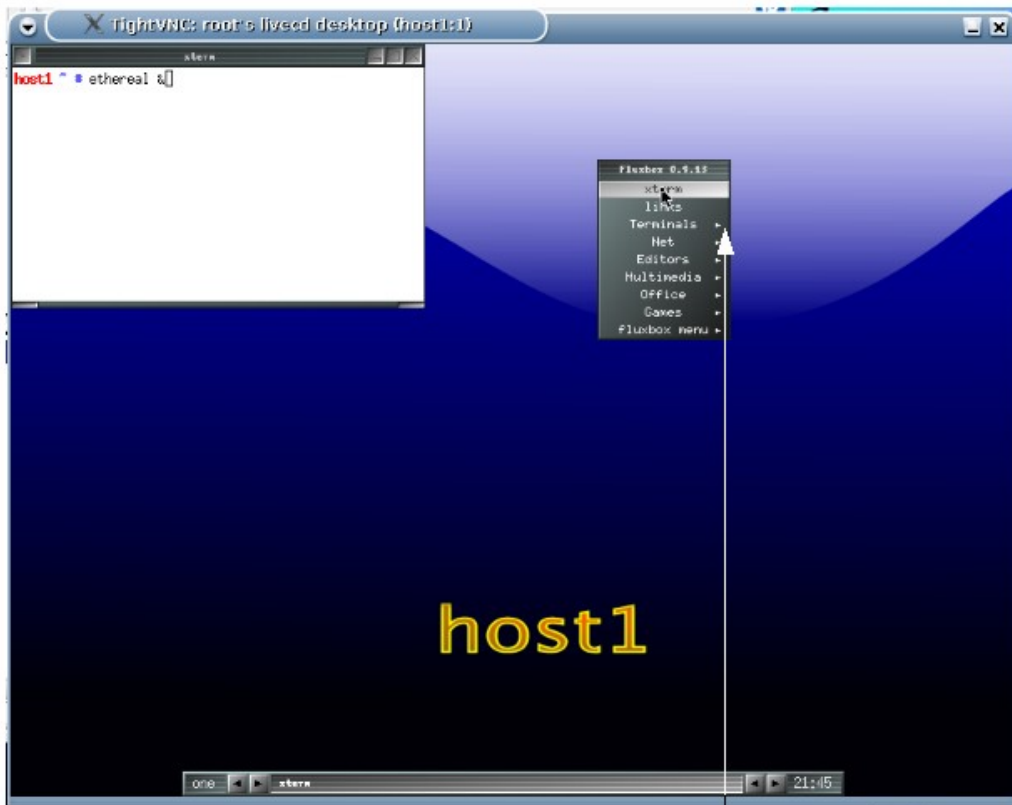
In the sensitive network map, a mouse rollover on each host of the model open a small menu which allows a ping test (alive ?), a ssh access (on first access you have to respond yes to generate ssh keys) and a graphical Fluxbox desktop access.

## Network Active Map (\*)



On a Fluxbox desktop, a right button click on the mouse gives you an access to a set of tools with super-user rights.

Host1 fluxbox desktop



Mouse right click

**Nota :** In some circumstances, on one or several virtual machines, this Fluxbox desktop is not fully operational. In that case right click access to the set of tools does not work. To restart the Fluxbox desktop just proceed as follows. Close the invalid Fluxbox desktop window. Using the sensitive map open a ssh console on that virtual machine and then restart X-Window and Fluxbox environments chaining the following commands in that order.

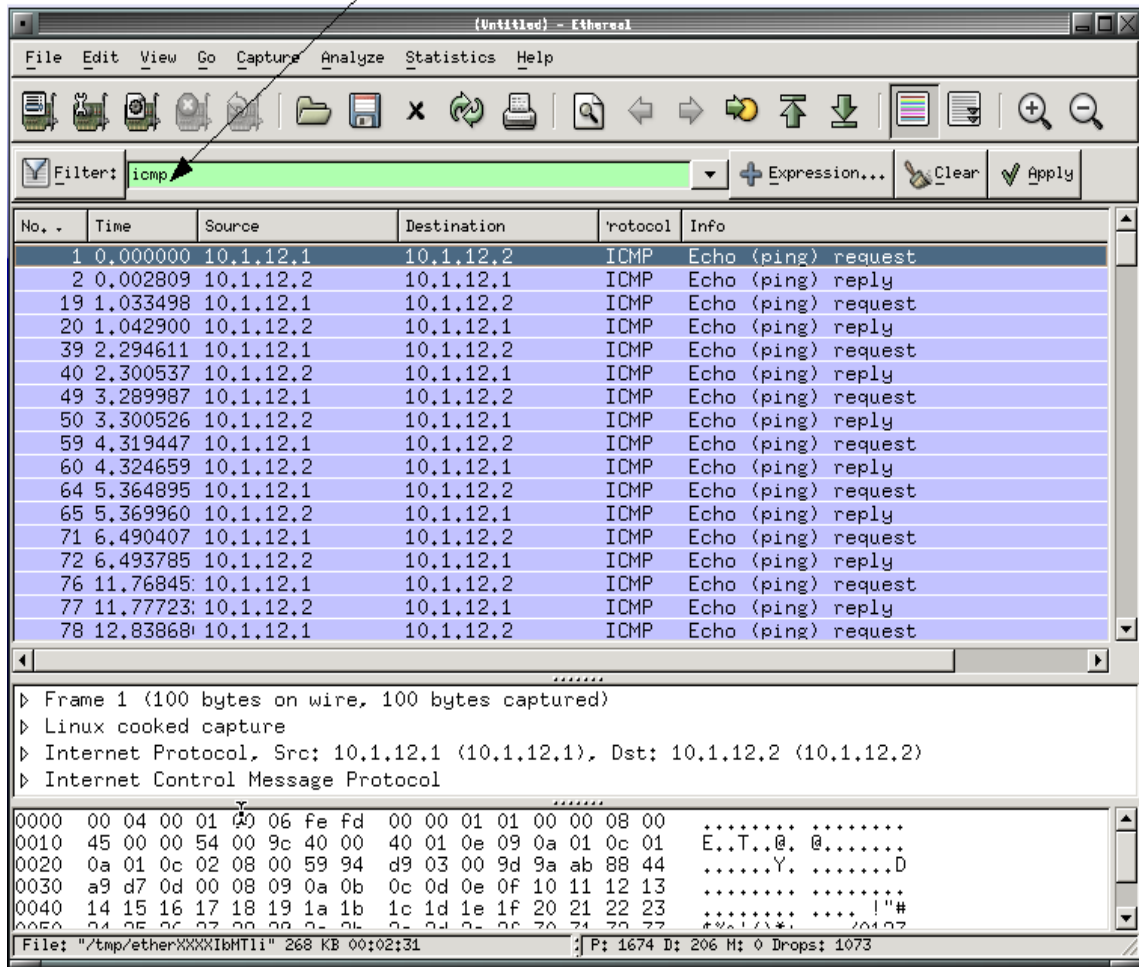
```
~# /etc/init.d/vm-fluxbox stop
~# /etc/init.d/vm-xvnc restart
~# /etc/init.d/vm-fluxbox start
```

The machine then displays a set of messages before giving back the shell prompt (~#). Then you can close the ssh terminal, and open the Fluxbox desktop using the sensitive map.

## Capturing network traffic

Each virtual machine of the model, embeds Wireshark, the great network traffic capture tool. As you have super-user rights on, you can observe traffic on all interfaces of the virtual machine. To launch Wireshark, just type 'wireshark &' at the shell prompt in an xterm. In the 'Capture' menu, launch traffic capture on all ethernet interfaces. Let capture run several seconds, then stop. Wireshark will then display all captured datagrams on each ethernet interface of the virtual machine. To make reading easier, filter the displayed trace by applying a display filter in the filter field (icmp display filter in the example bellow).

Wireshark display filter



-----oOo-----

VIMINAL  
medi6 lab  
(**M**odel to **E**xperiment and **D**iscover Ip v**6**)

Linux medi6 lab  
Model to Experiment and Discover Ip v6



**Preamble** : Content of this lab is a derived work from an initial one proposed by Olivier Barre and Matthieu Hennebo for a final project at TELECOM Lille1. Jacques Landru made a complement and an adaptation for VIMINAL (Virtual Model for Ip Network Achitecture Lab) plateforme for RIO curriculum (Reseaux et Interconnexions d'Ordinateurs). A copy of Barre and Hennebo's results is joined a this URL :

- Initial Mr. Barre and Mr. Hennebo proposition  
[//usr/local/share/viminal/viminal-mobidik-lab/instruction-sheets/tp-barre-hennebo/SujetTP\\_IPv6\\_EN.html](http://usr/local/share/viminal/viminal-mobidik-lab/instruction-sheets/tp-barre-hennebo/SujetTP_IPv6_EN.html)

I would like to thank Mr Luc Betry for his attentive rereading and his remarks

All the content of this lab instruction sheet is published under Creative-Commons license **CC-BY-NC-SA**.

**Jacques Landru**  
Ingénieur d'Etudes Institut TELECOM  
TELECOM Lille1  
Dept Informatique et Réseaux.

## Licence Creative Commons CC-BY-NC-SA



### Attribution-Noncommercial-Share Alike 2.0



You are free:

- to Share — to copy, distribute, display, and perform the work
- to Remix — to make derivative works
- ...

### Under the following conditions :



**Attribution** : You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



**Noncommercial** : You may not use this work for commercial purposes.



**Share Alike** : If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.
- Any of the above conditions can be waived if you get permission from the copyright holder.
- Nothing in this license impairs or restricts the author's moral rights.
- 

More details on these web sites :

- <http://creativecommons.org/>
- <http://creativecommons.org/about/licenses>
- [http://en.wikipedia.org/wiki/Creative\\_Commons\\_licenses](http://en.wikipedia.org/wiki/Creative_Commons_licenses)

## Content table

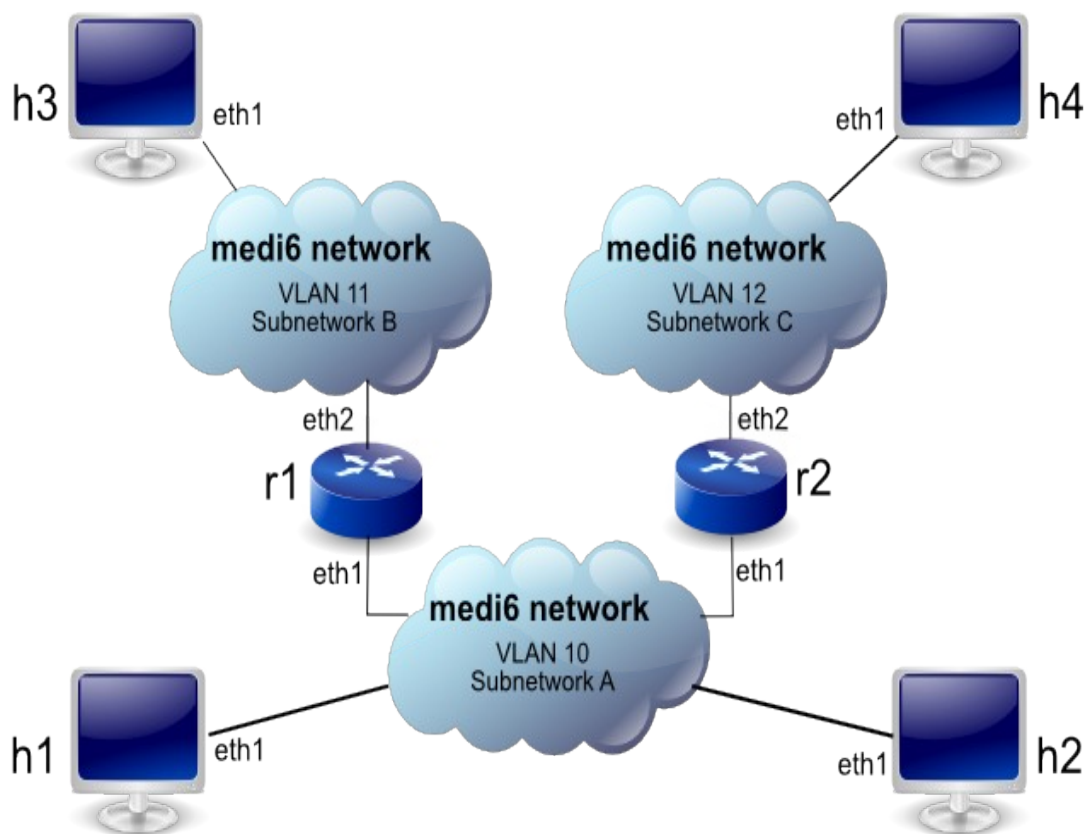
1 ) Lab context.....	7
2 ) Part I : 1st IPv6 set up.....	8
2.1 ) Model start.....	8
2.2 ) First contact with an IPv6 address.....	8
2.3 ) First IPv6 exchanges.....	8
Adding a router for stateless configuration.....	10
2.3.a ) Unicast prefix allocation.....	10
2.3.b ) "r1" Quagga router configuration.....	11
2.3.c ) Subnetworks A, B and C interconnection.....	15
2.4 ) Naming service (DNS Domain Name Service) for IPv6.....	17
2.4.a ) DNS changes for IPv6.....	17
2.4.b ) DNS server configuration.....	17
2.4.c ) Starting DNS service.....	19
2.4.d ) DNS checking.....	19
2.5 ) Before going further.....	21
3 ) Part II Transition/cohabitation IPv6 / IPv4.....	22
3.1 ) Transition and cohabitation techniques.....	22
3.2 ) Setting a 6to4 tunnel to interconnect an isolated IPv6 remote network through an IPv4 network.....	23
3.2.a ) 6to4 brief presentation.....	23
3.2.b ) 6to4 Simulation medi6 model.....	23
3.2.c ) IPv6 access tunnel end configuration.....	23
3.3 ) Setting up a transport level relay.....	27
3.3.a ) Relay technologies.....	27
Auxiliary DNS relay.....	28
3.3.b ) TOTD the GNU/Linux auxiliary DNS relay.....	28
3.3.c ) Setting up TOTD.....	28
3.3.c.1 ) Using TOTD through the 6to4 tunnel.....	32
3.3.d ) PTRTD the transport layer relay under GNU/Linux.....	32
3.3.e ) Setting up PTRTD.....	32
3.3.e.1 ) Using PTRTD through 6to4 tunnel.....	35



VIMINAL  
medi6 lab  
(**M**odel to **E**xperiment and **D**iscover **I**p v**6**)

## 1 ) Lab context

This lab is based on VIMINAL (**V**irtual **M**odel for **I**p **N**etwork **A**rchitectue **L**ab). medi6 (**M**odel to **E**xperiment and **D**iscover **I**p v**6**) model set up four virtual hosts (h1, h2, h3, and h4) and 2 virtual quagga routers (r1 and r2) interconnected on medi6 lan. The lab goal is to familiarize with basics of the new protocol as IPv6 address format, stateless auto-configuration, IPv6 address DNS integration and observe IPv6 traffic, using wireshark network analyzer. The picture below shows the medi6.LAN topology.



Linux medi6 lab  
Model to Experiment and Discover Ip v6

*Viminal*

*Virtual Model for Ip Network Architecture Lab*

**Note** : All the virtual machines have an eth0 network interface with an IPv4 address. These IPv4 addresses are used for the link between your physical machine and the virtual machines. During the lab don't use this eth0 interface, especially when using wireshark for traffic captures.

## 2 ) Part I : 1<sup>st</sup> IPv6 set up

### 2.1 ) Model start

To learn how to use VIMINAL, see “readme-1st” document on the computer desktop. Open the sensitive model map, and click on the start button. While model is starting, which can be long depending on your computer power, you can read the “readme-1st” document.

### 2.2 ) First contact with an IPv6 address

On the sensitive map open a command xterm on the “h1” machine (left button on the fluxbox desktop), or a ssh terminal.

The `ifconfig` command can be used to display the network interface configuration.

**Note :** To discover details about the `ifconfig` command, you can consult its manual pages using this command `# man ifconfig`

Type the following command to display the initial configuration of eth1 network interface.

```
h1~# ifconfig eth1
```

You also have the `ip` command which tends to replace the `ifconfig` and `route` commands for network interface configuration on most of the popular GNU/Linux distributions.

```
h1~# ip addr show eth1
```

**What is the IPv6 address of that machine ?**

.....

**What is its prefix value ? Which IPv6 type ? Is it a “routed able” address ?**

.....

**What is MAC address value of that machine ?**

.....

**Identify each part of the IPv6 address : network part, interface identifier part.**

**Look for modified EUI-64 interface identifier derived from the MAC address. See FFFE hexadecimal sequence insertion, and U/L reverse bit.**

Do the same observation on “h2” machine.

### 2.3 ) First IPv6 exchanges

On Ipv4, “ping” command is using ICMP protocol to test connectivity with a remote machine. On IPv6 we'll use ping6 command.

**Note** : More information about ping6, just type : # man ping6

Before pinging, launch wireshark traffic analyzer tool on h1 machine to start a traffic capture on eth1 network interface.

**Attention** : Don't launch capture on the pseudo “any” interface. Too many ethernet frames will be caught making capture readability difficult.

```
h1~# wireshark &
h1~#
```

On wireshark interface, to start a capture chain the following menus “capture → interfaces → eth1 → start”.

Open a new xterm and start the ping.

Verify that h1 and h2 can ping each others on IPv6 protocol. « -c 5 » option is used to limit the ping to five sending messages. When using ping6 command with link local address, you have to specify the interface with « -I » option.

```
h1~# ping6 -I eth1 -c 5 fe80::www:fdxx:yyyy::zzzz
```

**Note** : replace fe80::www:xxx:yyyy::zzzz with is the link local address of h2 machine.

**What is the RTD (Round Trip Delay) between these machines ?**

**Why the first RTD is higher ?**

Stop the wireshark capture (menu “Capture → interfaces → stop”, or click on red icon with white cross).

**Observe exchanges between h1 et h2 machines**

**A) Observe the 1<sup>st</sup> ICMP echo request message**

- **Ethernet header: find destination and source h1 and h2 MAC addresses. What is the hexadecimal value of the protocol code field ?**
- **Level 3 header, IPv6 datagram : notice bits "0110" pointing version 6 of IP, concerning the nature of higher protocol, notice the “next header” field with “0x3a” value for ICMP6 and IPv6 source and destination for h1 and h2.**
- **ICMP header note 128/0 type/code meaning ICMP6 “echo request”**

**B) Observe the two first captured frames**

- **What is this IPv6 operation ? What is the IPv4 corresponding operation ?**

**B1) Détails about the first captured frame**

- **Ethernet header : find the destination and source MAC addresses. What is the destination MAC address type ?**
- **Level 3 header, IPv6 datagram : IP version 6 note "0110" bits, higher level protocol : ICMP6 with next header value "0x3a". What are source and destination addresses. What is the type of the destination address. What is its context (prefix, lower order octets) ?**
- **ICMP header : 135/0 type /code meaning Neighbor solicitation. Which options does it content ?**

**B2) Details about the second frame**

- **Ethernet header : : find the destination and source MAC addresses. What is the destination MAC address type ?**
- **Level 3 header, IPv6 datagram: IP version 6 note "0110" bits, higher level protocol : ICMPv6 with next header value "0x3a". What are source and destination addresses compared with the first frame.**
- **ICMP header : 136/0 type /code meaning " Neighbor advertisement ". Which options does it content ?**

## Adding a router for stateless configuration

Until now, Quagga routers are inactive. That's why h1 and h2 machines have only one link local address, automatically built from the MAC address of the network interface. This type of address (link local) can't be routed, it can just be used to have traffic with the directly accessible machines, on the same broadcast domain (VLAN).

In this step, we're going to attribute unicast addresses which can be routed on the medi6 model interconnection.

### 2.3.a ) Unicast prefix allocation

IPv6 (as IPv4) makes the distinction between public addresses, usable on the public interconnection (the Internet), and private addresses, which can be routed only on a private scope (corporate intranet, campus network,...). This "unroutable" address space is governed by RFC1918 for IPv4 and RFC4193 for IPv6.

The latter define a unique local IPv6 unicast address. Like in RFC1918, the goal is to allow a private address space which can't be routed on the public internet. Unlike the RFC1918, where address space collision is problematic in case of joint venture between enterprises using the same RFC1918 addresses, RFC4193 local unicast IPv6 prefixes are practically unique. An organization, wishing a private prefix, has just has to choose a 48 bit long random prefix hashing the local time, the MAC address of one interface as described in the RFC4193 algorithm. Collision probability is very low as IPv6 address space is astronomically large.

These addresses have a local scope and shouldn't be routed on the global public Internet. Their

“routability” scope is limited to a site, or a small set of private sites inside the same IGP area like OSPF, or through point to point tunnels between sites.

Their characteristics are :

- globally unique prefix (with high probability of uniqueness) ;
- a well known prefix FC00::/7, which can easily be filtered on public border routers ;
- limitation of address conflicts or readdressing operations in case of site fusions or private interconnections ;
- Internet service provider independence of the globally unique unicast prefixes ;
- application independence, as their usage is the same as any other address ;
- in case of fortuitous routing overflow (mistaken border router configuration, erroneous DNS publication,...), uniqueness is granted, avoiding address conflicts.

For medi6 model, we don't have public global prefix which can be routed on the Internet. These prefixes are allocated by Internet service providers (ISP) when IPv6 Internet access is activated. We're going to allocate our own RFC4193 compliant 48 bits prefix using Hartmut Goebel's GPL script found at URL <http://forschung.goebel-consult.de/ipv6/createLULA.py>. This type of prefix has to be generated once, (at the newtork building). Using the script several time contradicts the spirit of the RFC.

Execute the script launching the command

```
h1~# /opt/rfc4193/createLULA.py
```

**Choose one, among the displayed globally unique unicast prefixes and note it for your medi6 network :**

...../48

It's a 48 bits long prefix, we have then 16 bits for our subnets identification on our interconnection to obtain 64 bits long unicast prefixes.

The three medi6 model subnets will be identified with 10, 11 and 12 (decimal) references (A, B and C in hexadecimal).

### 2.3.b ) “r1” Quagga router configuration

As previously seen, r1 Quagga router is inactive. That's why h1 and h2 machine has only a link local address. We're going to observe now the stateless auto-configuration mechanism, using the router advertisement information sent by the router. To configure the router, we'll directly edit the Zebra configuration files (Zebra is the router process of the Quagga software package).

**Attention :** Make a backup of your config files before modifying

```
# cp /etc/quagga/zebra.conf /etc/quagga/zebra.conf.orig
```

Before launching routing services, we're going to configure the IPv6 addresses of the router network interfaces.

“r1” router interconnects the fdxx:yyyy:zzzz:a::/64 and fdxx:yyyy:zzzz:b::/64 subnetworks (where fdxx:yyyy:zzzz:: is your unique local unicast prefix generated during the previous step).

Before configuring the eth1 interface of r1 router, restart the wireshark traffic capture on the eth1 interface of h1 to observe what happen on the broadcast domain of the A medi6 subnetwork.

Manual IPv6 configuration of the interface can be done using the `ip` command. Interface identifiers of eth1 and eth2 interfaces on r1 will be assigned to value "1" (those of r2 will be assigned the value "2").

```
r1~# ip -6 addr add fdxx:yyyy:zzzz:a::1/64 dev eth1
```

**Note 1** : replace fdxx:yyyy:zzzz with your unique local unicast prefix.

**Note 2** : You can alternatively use the `ifconfig` command which is deprecated on most of the GNU/Linux distributions.

```
r1~# ifconfig eth1 inet6 add fdxx:yyyy:zzzz:a::1/64
```

Then, verify eth1 interface addresses :

```
r1~# ip addr show eth1.
```

Observe traffic capture on h1. Note that the router spontaneously send MLDv2 (Multicast Listener Discovery) datagrams to unsubscribe its solicited multicast address (??) and one neighbor discovery datagram for the Duplicate Address Detection (DAD) algorithm. As the interface identifier is manually set to 1, and not derived from the MAC address (like for link local address), the machine replays the DAD algorithm.

Do to the eth2 interface configuration on the "b" subnetwork of medi6 interconnection.

```
r1~# ip -6 addr add fdxx:yyyy:zzzz:b::1/64 dev eth2
```

```
r1~# ip addr show eth2
```

For GNU/Linux to be used as a router and routes datagrams between its network interfaces, you have to activate the "forwarding" kernel option setting to value "1" the `/proc/sys/net/ipvx/conf/interface/forwarding` parameter, where `ipvx` is the protocol version ipv4 or ipv6, `interface` is the interface (all, eth0 eth1, eth2,...). We activate the ipv6 routing capabilities for « all », « eth1 » et « eth2 » interfaces of the router.

```
r1~#echo 1 >/proc/sys/net/ipv6/conf/all/forwarding
r1~#echo 1 >/proc/sys/net/ipv6/conf/eth1/forwarding
r1~#echo 1 >/proc/sys/net/ipv6/conf/eth2/forwarding
```

Zebra/quagga routers can be configured in direct command mode, using a Command Line Interface (CLI), as for the leading router manufacturer and its famous IOS, or by directly the configuration file `/etc/quagga/zebra.conf`. That's the latter mode we'll use for this lab.

Edit the `/etc/quagga/zebra.conf` file (You have to insert your prefix at the specified point).

```
r1~#nano -w /etc/quagga/zebra.conf
```

Configure eth1 ethernet interface of r1 to announce the "10" subnetwork (A in hexadecimal) of your unique local unicast prefix (information like fdxx:yyyy:zzzz:A::/64) and eth2 interface for the "11" subnetwork (B inn hexadecimal).

**Note** : For router configuration you can refer to IPv6 part of Quaaga documentation

(/usr/local/share/viminal/viminal-medi6-lab/instruction-sheets/docs/quagga.pdf ).

On h1, start a new traffic capture on eth1 interface of h1.

**Attention** : Don't launch capture on the pseudo "any" interface. Too many ethernet frames will be caught making capture readability difficult.

```
h1~# wireshark &
```

On wireshark interface, to start a capture chain the following menus « capture → interfaces → eth1 → start ».

On r1 machine, routing service daemon is called "zebra". "quagga" system user, which has daemon control, must have "write" permission on /var/run/quagga and /var/log/quagga directories. Before starting the service, give the ownership to the quagga user, using "chown" (change owner) command.

```
r1~# chown quagga /var/run/quagga
r1~# chown quagga /var/log/quagga
```

A startup script, in /etc/init.d directory, makes some controls before starting the service. Launch the service with " /etc/init.d/zebra start " command.

```
r1~# /etc/init.d/zebra start
```

Virtual machine then displays.

```
* Cleaning up stale zebra routes... ... [ ok ]
* starting Named [ ok ]
```

Verify all is OK, displaying the tail of the zebra log, using "tail /var/log/quagga/zebra.log" command.

Vérifiez que tout s'est bien passé, en consultant la fin du journal zebra à l'aide de la commande

```
r1~# tail /var/log/quagga/zebra.log
```

An error message means a misconfiguration, you have to correct before going further.

On h1, stop the wireshark capture (menu "capture → interfaces → stop", ou click on red icon with white cross).

In a second xterm, observe the eth1 interface IPv6 configuration.

```
h1~# ip -6 addr show eth1
```

**How many IPv6 addresses are set on eth1 interface of h1 ? Note the value of these addresses.**

**What are the interface identifiers of those addresses ? Are they identical ?**

**What are the address types ? How do you identify those types ? What are the values**

**of these type identifiers (the leading bits of the address) ?**

Do the same observations on h2 machine.

Observe the advertisement datagrams sent by r1 and captured by wireshark.

**Find the prefix in the captured datagrams**

**IPv6 datagram header**

- **What is the IPv6 de destination address ? Which type ? What are the destination machines ?**
- **The "hop limit" value for autoconfiguration advertisement sent by routers is signifcative : What is its value ? Why such a value ?**
- **Find the 8 bits coding the next header for the higher protocol (ICMP).**

**ICMPv6 header**

- **Type : What is the type value ? What does it mean for ICMPv6 ?**
- **Code : What is the code value ?**

**ICMPv6 data :**

- **M field ("Managed address configuration" flags) : What is its value ? What does this mean for the host network interface ?**
- **What is the router time to live ?**
- **What are the ICMPv6 options ?**
- **Prefix field : Find the prefix previously set. How long is it ? what are the L (on-Link flag) and A (Autonomous) bits ?**

**Can we ping h1 from h2 with these new IPv6 addresses ?**

**Restart the ping6 command with a traffic capture. Observe the IPv6 datagram content. What are the source and destination addresses ?**

**What is the auto-configuration unicast address of h3 ? Can we ping h3 form h1 ?**

Restart the ping to h3 while capturing the traffic with wireshark. Observe IPv6 datagram content.

**What are source and destination level 2 (ethernet) addresses ? Same question about level 3 IPv6**

On h1 machine, observe IPv6 routing table.

```
h1~# ip -6 route show
```

### 2.3.c ) Subnetworks A, B and C interconnection

We're adding now the third subnetwork, activating the r2 configuration.

Make a backup `/etc/quagga/zebra.conf` copy on `/etc/quagga/zebra.conf.orig` file before editing `/etc/quagga/zebra.conf` file of r2 router to advertise the `fdxx:yyyy:zzzz:a::/64` préfix on **eth1** interface and `fdxx:yyyy:zzzz:c::/64` on **eth2**.

Manually configure IPv6 unicast addresses for eth1 and eth2 setting interface identifier to "2". Then activate IPv6 forwarding on network interfaces, edit the zebra conf file and give the write permission for quagga user on `/var/log/quagga` and `/var/run/quagga`.

```
r2~# ip -6 addr add fdxx:yyyy:zzzz:a::2/64 dev eth1
r2~# ip addr show eth1
.
.
.
r2~# ip -6 addr add fdxx:yyyy:zzzz:c::2/64 dev eth2
r2~# ip addr show eth2
.
.
.
r2~# echo 1 >/proc/sys/net/ipv6/conf/all/forwarding
r2~# echo 1 >/proc/sys/net/ipv6/conf/eth1/forwarding
r2~# echo 1 >/proc/sys/net/ipv6/conf/eth2/forwarding
r2~# nano -w /etc/quagga/zebra.conf
.
.
.
r2~# chown quagga /var/run/quagga
r2~# chown quagga /var/log/quagga
```

**Note** : replace `fdxx:yyyy:zzzz` with your unique local unicast prefix.

You can now start routing daemon on r2 router.

```
r2~# /etc/init.d/zebra start
```

Verify h4 has an unicast address on C subnetwork. Note this address. On h1, have a look at the routing table.

```
h1~# ip -6 route show
```

**What are the modification compared to the previous version ?**

**Can we ping h4 from h1 ?**

**Why ?**

The interconnection is not operational, because r1 and r2 routing tables are not updated. We

have to activate a routing protocol to allow routing table exchanges between routers. In this lab, we'll use RIPng which is an IPv6 able version of RIP.

On r1 and r2 have a look to the `/etc/quagga/ripngd.conf` file. It contains the RIPng commands for eth1 and eth2 interfaces to advertise connected networks.

Before starting the routing daemon, start a new wireshark traffic capture on h1.

On r1 and r2 start ripngd service using `/etc/init.d/ripngd start` command.

```
r1~# /etc/init.d/ripngd start
```

```
r2~# /etc/init.d/ripngd start
```

### **Display routing table contents for r1 and r2 routers**

- **Are new routes added ?**
- **Is there a route for each subnetwork ?**

Stop wireshark capture and observe RIPng datagrams.

**What is the IPv6 destination address for these datagrams ?**

**What is the type of that address ?**

**What is its scope ?**

**What machines are concerned ? (see `/usr/local/share/viminal/viminal-medi6-lab/instruction-sheets/docs/ipv6-multicast-addresses.txt`).**

Restart a new wirewhark capture on eth1 interface of h1. Start ping6 command to h4.

**Why the first ping is longer ?**

**Which is the first default router on h1 ? (Display again the routing table using command `ip -6 route show`)**

**Is h4 machine behind this default router ?**

**Stop the h1 wireshark capture.**

**What is the destination MAC address of the first " echo request " ?**

**Observe the ICMP redirect message sent by the default router**

**- What are its options ?**



**In which files are configured these zones ?**

Edit the `/etc/bind/named.conf` file to replace “z.z.z.z.y.y.y.x.x.f.d” with the reverse hexadecimal doted notation of your 48 bit unicast local prefix `fdxxx.yyyy.zzzz::/48`). Attention, DNS reverse lookup needs to write hexadecimal digit in reverse position with dot separator.

**Attention :** write the alphanumeric hexadecimal digit (from “a” to “f”) in lower case.

```
h1~# nano -w /etc/bind/named.conf
```

DNS table data files are located in `/etc/bind/pri` Directory.

```
h1~# cd /etc/bind/pri
```

Take note of the IPv6 addresses of all medi6 machines (h1, h2, h3, and h4).

```
h1~# ip -6 addr show

h2~# ip -6 addr show

h3~# ip -6 addr show

h4~# ip -6 addr show
```

On h1 machine, keep a backup copy of the original files

```
h1~# cp medi6.lab.zone.orig medi6.lab.zone

h2~# cp
z.z.z.z.y.y.y.x.x.f.d.ip6.arpa.zone.orig your.reverse.hexadecimal.p
refixe.ip6.arpa.zone
```

**Note :** On the latter command replace “your.reverse.haxadecimal.prefix” with your unique local prefix in reverse doted hexadecimal notation.

Edit the `medi6.lab` zone file to write, where the indicated, IPv6 addresses of each medi6 lab machine. Attention h1 machine has two addresses with one having interface identifier value set to “10”. Verify your settings and save the configuration file.

```
h1~# nano -w medi6.lab.zone
```

Edit the reverse zone file to set “\$origin” parameter with your 48 bits unique local prefix in reverse doted hexadecimal notation, and for each machine of the domain the address rest (subnet and interface identifier) also in reverse doted hexadecimal notation.

**Attention** : Reverse dotted hexadecimal notation needs all digit to be written, also leading zeros or not displayed zeros between the “.”. So “\$origin” parameter has 12 hexadecimal digit and the rest (subnet and interface identifier) 20 hexadecimal digits. **Tedious, isn't it !!**

```
h1~# nano -w your.reverse.hexadecimal.prefixe.ip6.arpa.zone
```

Check your settings and save the configuration files.

### 2.4.c ) Starting DNS service

DNS service daemon is called “ named “. “ named “ system user, which has daemon control, must have the “ write ” permissions on the `/var/run/named` directory. Before starting the service, give the directory ownership to the named user, using the “ `chown` “ (change owner) command.

```
h1~# chown named /var/run/named
```

A startup script, in `/etc/init.d` directoy, makes some controls before starting the service. Launch the service with “ `/etc/init.d/named start` ” command.

```
h1~# /etc/init.d/named start
```

The machine then displays.

```
starting Named [ok]
```

Verify all is OK, displaying the tail of the system log.

```
h1~# tail /var/log/messages
```

An error message means a misconfiguration, you have to correct before going further. In that case do the modifications and restart the service using “ `/etc/init.d/named restart` ” command.

Edit the DNS client configuration file (`/etc/resolv.conf`) to set the local attached DNS server address (h1 address with interface identifier value '10').

```
h1~# nano -w /etc/resolv.conf
```

### 2.4.d ) DNS checking

We're going to test the DNS server using `nslookup` command

**Note** : `nslookup` command tends to be replaced by `dig` tool, which is more advanced. You can display manual page for `dig` with “ `man dig` ” command.

“nslookup” command works in interactive mode. A new prompt is displayed, as a “>”. Set your request type to value “ANY” to query DNS database about all RR (Resource Record) types. Then launch a query to about h1 machine, h2.medi6.lab and then test the reverse lookup querying the address with the interface identifier “10” of the h1 machine.

```
h1~# nslookup
> set type=ANY
>h1.
.
.
.
>h2.medi6.lab
.
.
.
>fdxx:yyyy:zzzz:a::10
.
.
.
>exit
```

Edit the DNS client configuration file (/etc/resolv.conf) of all the other machines of the medi6 model to set attached DNS server address (h1 address with interface identifier “10”).

```
h2~# nano -w /etc/resolv.conf
.
.
.
h3~# nano -w /etc/resolv.conf
.
.
.
h4~# nano -w /etc/resolv.conf
.
.
.
```

You can now make ping between medi6 machines using their name instead of their address.

```
h1~# ping6 -c 5 h4
.
.
.
h3~# ping6 -c 5 h2.medi6.lab
.
.
.
```

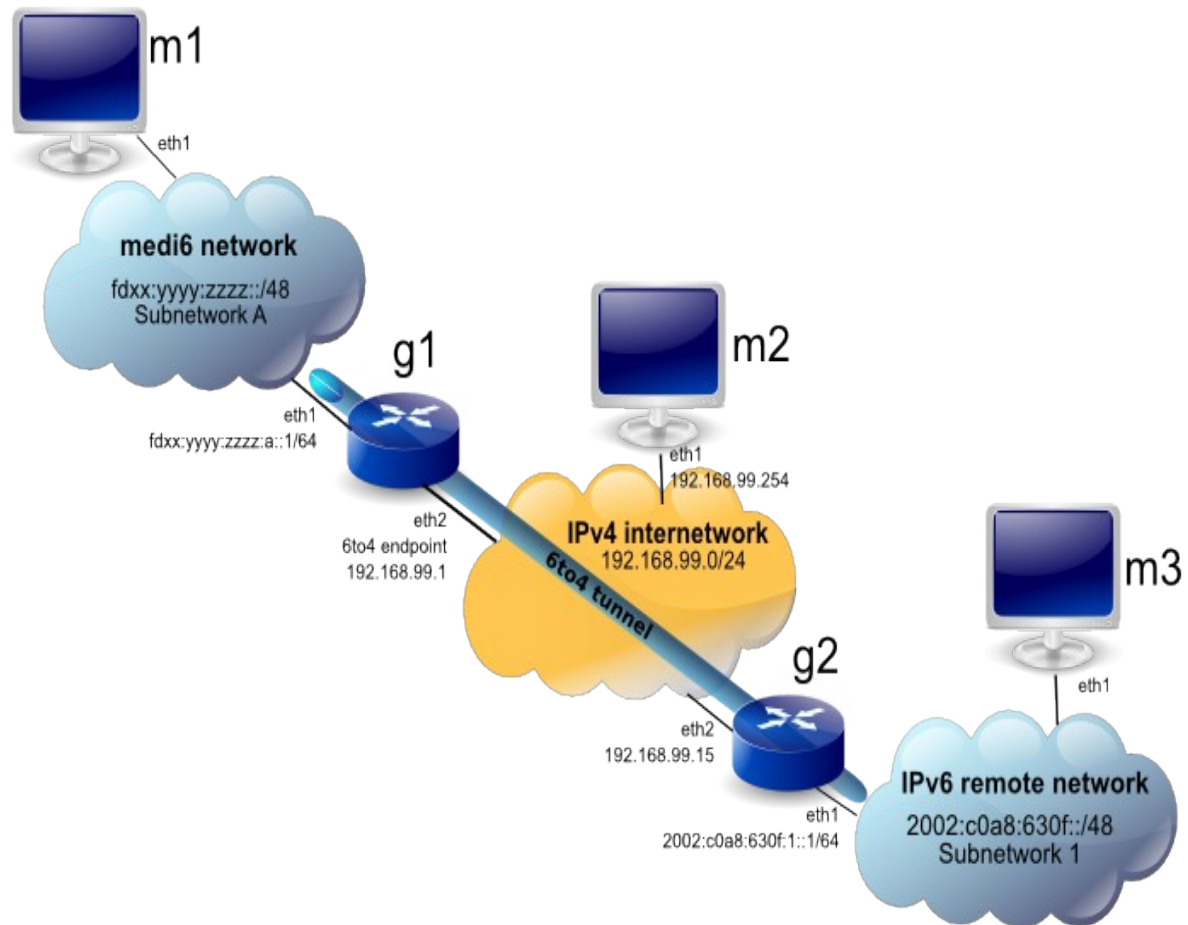
On h1, start a new wireshark traffic capture. On h2, make DNS lookup with “nslookup” command. Stop wireshark capture and observe DNS exchanges in detail.

```
h2~# nslookup
> set type=ANY
>h1.
.
.
.
>h2.medi6.lab
.
.
.
>fdxx:yyyy:zzzz:a::10
.
.
.
>exit
```

## 2.5 ) Before going further...

Stop the model, clicking on the “Stop/purge” button. After the complete model shutdown, display the second part model map of the lab (URL : Medi6 Part II) on the top right part of the map. The picture below shows the network topology for the second part of the lab.

### 3 ) Part II Transition/cohabitation IPv6 / IPv4



Linux medi6 lab  
Model to Experiment and Discover Ip v6

Viminal

Virtual Model for Ip Network Architecture Lab

**Note :** All the virtual machines have an **eth0** network interface with an IPv4 address. These IPv4 addresses are used for the link between your physical machine and the virtual machines. During the lab don't use this **eth0** interface, especially when using wireshark for traffic captures.

Launch the second medi6 model clicking on the start button. While model is starting, which can be long depending on your computer power, read the following topics of the lab.

#### 3.1 ) Transition and cohabitation techniques

In this second part, we're going to set several tools for cohabitation between the two versions of the protocol. There's no universal tool which can simply and transparently handle communications between an IPv6 only machine and an IPv4 only one. Several mechanisms can be deployed to cover the v4/v6 cohabitation and transition goals. Each mechanism has some limitations. It's the association of several of these mechanisms which can help to cover all the functionalities needed for a given architecture. They derived from a limited set of basic mechanisms :

- Dual stack,
- Encapsulation (tunnel, encapsulation IPv6 in IPv4, IPv4 in IPv6,...),
- Translation,
  - Application level (« proxy » and « reverse-proxy » http, mail relay, DNS relay,...),
  - Transport level (tptr (Transport Relay Translator) RFC3142; SOCKS relay RFC3089)
  - Network level (NAT-PT, SIIT,...)

Medi6 lab doesn't pretend to be exhaustive on this topic. In this part we'll cover 6to4 tunnel and transport relay using ptrtd.

## 3.2 ) Setting a 6to4 tunnel to interconnect an isolated IPv6 remote network through an IPv4 network

### 3.2.a ) 6to4 brief presentation

6to4 mechanism allows connection of an IPv6 island, with no direct access to an IPv6 provider but having at least one IPv4 public address to accede IPv6 service through IPv4 network (RFC3056, RFC3068). 6To4 is a neat technique to automatically build tunnel and allocate global prefix. The 2002::/16 prefix has been reserved by IANA for this usage. 48 bits prefix can easily be built appending the IPv4 border router address to the prefix. The border router has a dual connection, one on the isolated IPv6 network and one on the IPv4 network for which the ISP has given a route able address given by the ISP. The router use this public IPv4 address to build the 48 bits prefix. This 2002::/16 address plan is in accordance with the global IPv6 adress plan (RFC3513), there's 16 remainder bits to address subnetworks and 64 bits for interface identifiers.

The other side of the tunnel is handle by 6to4 public gateway set by Internet Service Providers. 192.88.99.1 IPv4 address, reserved by IANA (RFC3068) is dedicated to this purpose. It is called the "6to4 Relay anycast prefix". Every 6to4 public gateway of Internet V6 can be reached with this address. The 192.88.99.0/24 and 2002::/16 routes are BGP announced by each ISP network providing 6to4 service. An isolated IPv6 site can then automatically send its packets to the nearest (routing meaning) 6to4 public server to have Internet V6 access. 192.88.99.1 address behaves like an anycast address.

### 3.2.b ) 6to4 Simulation medi6 model

On medi6 model, we don't have a public Internet (v4 and v6) connectivity. We are going to build a 6to4 tunnel to connect an isolated IPv6 remote network to our IPv6 unique unicast local addressed (fdxx:yyyy:zzzz:a::/64) network through a private IPv4 network (192.168.99.0/24). This latter network simulates Internet v4 for our model. G2 gateway, on the isolated remote network, is given 192.168.99.15 by the access provider. Isolated remote network will then have 2002:c0a8:630f::/48 prefix (2002::/16 prefix followed by 32 bits IPv4 address of the gateway e.g. c0a8:630f in hexadecimal). The 6to4 relay anycast prefix on G1 gateway (see picture above) is simulated by 192.168.99.1 address.

### 3.2.c ) IPv6 access tunnel end configuration

For the IPv6 medi6 network, we'll reuse unique local unicast prefix (fdxx:yyyy:zzzz::/48) defined during the first part of the lab.

For the first step configure r1 router :

- activating “IP forwarding” for protocols v4 et v6,
- assigning an IPv6 address on eth1 interface,
- completing config file `/etc/quagga/zebra.conf`
- starting routing service,
- verifying addresses of eth1 and eth2 interfaces.

```
g1~# echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
g1~# echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
g1~# ip -6 addr add fdxx:yyyy:zzzz:a::1/64 dev eth1
g1~# nano -w /etc/quagga/zebra.conf

g1~# chown quagga /var/log/quagga
g1~# chown quagga /var/run/quagga
g1~# /etc/init.d/zebra start

g1~#
g1~# ip addr show eth1

g1~# ip addr show eth2
```

Do the same operations on IPv6 remote network g2 router. Be careful : g2 eth1 IPv6 address is `2002:c0a8:630f:1::1/64` and router prefix advertisement is `2002:c0a8:630f:1::/64`

```
g2~# echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
g2~# echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
g2~# ip -6 addr add 2002:c0a8:630f:1::1/64 dev eth1
g2~# nano -w /etc/quagga/zebra.conf

g2~# chown quagga /var/log/quagga
g2~# chown quagga /var/run/quagga
g2~# /etc/init.d/zebra start

g2~#
g2~# ip addr show eth1

g2~# ip addr show eth2
```

We're now going to create and activate each tunnel end using “ip” command

**Nota** : see « man ip » command for more details about ip command parameters.

On g1 router the tunnel end will be created and configured chaining the following commands :

1. “ ip tunnel add tunnel624 mode sit ttl 64 remote any local 192.168.99.1 “ command creates a 6to4 tunnel interface named tunnel624 in “sit” mode (modes are : ipip, gre , sit, isatap, ip6ip6, ipip6, for a 6to4 tunnel use “ sit “ mode). Remote interface is “any”, local interface is 192.168.99.1.
2. “ ip link set dev tunnel624 up “ command activates tunnel624 virtual interface.
3. “ ip -6 addr add fdxx:yyyy:zzzz:a::2/16 dev tunnel624 “ command sets a local address to the tunnel interface we have just created and activated. **Be careful** : It seems that the 16 bits prefix length is important for this command while it doesn't correspond to the real network prefix which is /64 in our case. We choose “2” value for the interface identifier (as “1” is already used for eth1 interface of the router).
4. “ ip -6 route add ::/96 dev tunnel624 metric 1 “ command adds a new entry in the routing table for IPv4 mapped addresses (addresses ::a.b.c.d/96). The tunnel interface we have just created has a such address ::192.168.99.1. It's on that interface we are going to route the datagrams for the IPv6 remote network. IPv4 mapped addresses will also be routed on the tunnel interface.
5. “ ip -6 route add 2002:c0a8:630f::/48 via ::192.168.99.15 dev tunnel624 metric 1 “ command adds a new routing entry to direct 2002:c0a8:630f::/48 destination datagrams to tunnel624 interface via the other tunnel end ipv4 mapped address (::192.168.99.15).
6. “ ip -6 route show “ and “ ip -6 addr show tunnel624 “ commands display tunnel624 interface address and the routing table for verification.

```
g1~# ip tunnel add tunnel624 mode sit ttl 64 remote any local
192.168.99.1
g1~# ip link set dev tunnel624 up
g1~# ip -6 addr add fdxx:yyyy:zzzz:a::2/16 dev tunnel624
g1~# ip -6 route add ::/96 dev tunnel624 metric 1
g1~# ip -6 route add 2002:c0a8:630f::/48 via ::192.168.99.15 dev
tunnel624 metric 1

g1~#ip -6 addr show tunnel624
g1~#ip -6 route show
g1~#
```

**Observe the two last commands display :**

**What are IPv6 addresses for tunnel624 interface, what is the address type ?**

**Find entries for destination prefixes ::/96 et 2002:c0a8:630f::/48 in the routing table**

On g2, the other tunnel end will be created and configured chaining the following commands :

7. “ ip tunnel add tunnel624 mode sit ttl 64 remote any local 192.168.99.15 “ command create a 6to4 tunnel interface named tunnel624 in “sit” mode. Remote interface is “any”, local interface is 192.168.99.15.
8. “ ip link set dev tunnel624 up “ command activates tunnel624 virtual interface.

9. “ ip -6 addr add **2002:c0a8:630f:1::2/16** dev tunnel624 “command sets a local address to the tunnel interface we have just created and activated. **Be careful** : It seems that the 16 bits prefix length is important for this command while it doesn't correspond to the real network prefix which is /64 in our case. We choose “2” value for the interface identifier (as “1” is already used for eth1 interface of the router).
- 10.“ ip -6 route add **::/96** dev tunnel624 metric 1 ” command adds a new entry in the routing table for IPv4 mapped addresses (addresses ::a.b.c.d/96). The tunnel interface we have just created has a such address ::192.168.99.15. It's on that interface we are going to route the datagrams for the IPv6 remote network. IPv4 mapped addresses will also be routed on the tunnel interface.
- 11.“ ip -6 route add fd00/8 via **::192.168.99.1** dev tunnel624 metric 1 “ command adds a new routing entry to direct fd00::- 12.“ ip -6 route show “ and “ ip -6 addr show tunnel624 “ commands display tunnel624 interface address and the routing table for verification.

```

g2~# ip tunnel add tunnel624 mode sit ttl 64 remote any local
192.168.99.15
g2~# ip link set dev tunnel624 up
g2~# ip -6 addr add 2002:c0a8:630f:1::2/16 dev tunnel624
g2~# ip -6 route add ::/96 dev tunnel624 metric 1
g2~# ip -6 route add fd00::

```

The tunnel is now created and configured. Observe what happens in usage.

- Note eth1 IPv6 address on m1,
- on g2 launch a wireshark capture on eth2 interface;
- on g1 launch a wireshark capture on tunnel624 interface,
- on m1 launch a wireshark capture on eth1 interface,
- on m3 machine, display IPv6 addresses to verify eth1 is well addressed with a préfixe2002:c0a8:630f:1::/64 prefix and start a “ping” to m1 machine.

```

m3~# ping6 -c 5 addr:of::m1

```

Stop wireshark captures. Observe datagrams on each check point.

### “echo request” on eth2 interface of g2

**ethernet frame : source MAC address, destination MAC address, type of payload ?**

**Internet protocol : version field, source and destination addresses, “ don't fragment flag “, upper protocol field : IPv6 (0x29).**

**Internet protocol Version 6 : source and destination addresses, « next header » field.**

**“echo request” on tunnel624 interface of g1**

**Is there a level 2 ethernet ? What means “ Raw packet data “, Why ?**

**Has the IPv6 datagram a specific form ?**

**“echo request” on eth1 interface of m1**

**ethernet frame : source MAC address, destination MAC address, type of payload ?**

**Internet protocole Version 6 : source and destination addresses, “next header” field.**

From m3 machine launch a “tracepath6” to m1 and observe results.

```
m3~# tracepath6 addr:of::m1
```

## 3.3 ) Setting up a transport level relay

### 3.3.a ) Relay technologies.

Translation or relay mechanisms allow single protocol machines to transparently communicate. Relays can be installed in cutting mode between IPv6 and IPv4 networks. They differ depending of their network level position in the OSI model.

- Application layer gateway (ALG) : A simple method to transfer data between IPv6 and IPv4 realms. Based on dual stacks machines they group together web caches and proxies, mail transfer agents (MTA, DNS servers, printing spoolers,... Clients need to be configured to delegate requests to the relays (c.f. Proxy configuration in web browsers).
- Transport layer gateway : Used by firewalls to control and relay TCP or UDP flows. They basically works like classical HTTP proxies but for other protocols. They can be seen as a generic proxies. In IPv4 world they are used to translate from private to public address spaces. SOCKS relays (RFC1928) and TRT relays (RFC3142) are in this category. In this lab we're going to set up a TRT relay using “ptrtd” package.
- Network layer relay : Well known in IPv4 world to translate flows from a private addressing area to a public one, NAT-PT network address translation port translation (NAT-PT RFC2766) can be used between IPv6 only network to IPv4 network. NAT-PT functions integrated on border router maintain mapping address table. The NAT-PT is a state mechanism, which does not survive after a reboot of the router. Meanwhile a second IPv6 stateless mechanism was defined known as SIIT (Stateless IP Icmp Translation RFC2765). It requires the use of specific IPv6 addresses ( "translated IPv4 addresses" and "mapped IPv4 addresses, which causes a dual addresses seating on stations IPv6, network address management is then harder

## Auxiliary DNS relay

To ensure transparency to applicative protocols, transport layer relays or network layer relays may rely on an auxiliary DNS relay (DNS-ALG DNS Application Layer Gateway). It will translate IPv4 addresses into IPv6 addresses with a special prefix to be routed to the transport or network layer relay. From the client perspective, it behaves like any attached DNS server. It accepts requests and transfers them to the local attached DNS server, if it already has no information in its local cache. When the client makes an IPv6 "AAAA" request, DNS relay transfers it to the attached DNS server. If the answer is of type "A" only. It concatenates a special 96 bits prefix to 32 bits of the IPv4 address. Datagrams with a destination address based on that prefix will be routed through the network to the transport layer or network layer relay (SIIT and NAT). The prefix usually reserved for this purpose is a site local prefix (fec0: 0:0: ffff:: / 64). Thus the machine with address 193.99.144.71 will have its DNS record type "A" converted by the auxiliary relay DNS in a "AAAA" record with the value fec0: 0:0: ffff: 0:0: C163: 9047.

Note that the site local prefix fec0:: / 10 has been obsoleted by RFC3879. It would therefore be preferable in the implementation of a DNS Relay to use a different non Internet route able prefix incorporating an IPv4 address.

### 3.3.b ) TOTD the GNU/Linux auxiliary DNS relay

In this lab, we'll set up TOTD (Trick Or Treat Daemon, the DNS proxy for IPv4/IPv6 translation). TOTD is a free implementation for GNU / Linux or \* BSD for that type of service. By default it is configured to use the prefix 3ffe: abcd: 1234:9876:: /64. Prefix 3ffe:: /16 was the prefix for 6bone experimental network, whose use has symbolically died June 6, 2006 (060606) (see RFC3701). Since then these addresses are not routed by operators. Address 3ffe:: /16 can be used on an experimental basis.

### 3.3.c ) Setting up TOTD.

On medi6 model, it's g1 machine which will host the auxiliary DNS relay service and m2 will be the medi6 DNS server.

In a first step we need to configure DNS service on m2 in conformance with the medi6 model addressing scheme.

First edit the file `/etc/bind/named.conf` to complete missing information about your unique local unicast prefix (be careful, you have to replace the `z.z.z.z.y.y.y.x.x.d.f` string with your unique local prefix in reverse dotted hexadecimal notation for the zone name and its associated file name.

```
m2~# nano -w /etc/bind/named.conf
```

Keep a backup copy of your original file (.orig extension) before updating.

```
m2~#cd /etc/bind/pri
m2 pri # cp medi6.lab.zone.orig medi6.lab.zone

m2 pri # cp
z.z.z.z.y.y.y.x.x.d.f.ip6.arpa.zone.orig your.reverse.hexadecimal.p
```

```
refixe.ip6.arpa.zone.  
  
m2 pri # cp  
f.0.3.6.8.a.0.c.2.0.0.2.ip6.arpa.zone.orig f.0.3.6.8.a.0.c.2.0.0.2.ip  
6.arpa.zone
```

**Nota :** On the second command replace “your.reverse.haxadecimal.prefix” with your unique local prefix in reverse doted hexadecimal notation.

Edit and complete files :

- medi6.lab.zone,
- **your.reverse.hexadecimal.prefix.ip6.arpa.zone**
- f.0.3.6.8.a.0.c.2.0.0.2.ip6.arpa.zone

to set them in accordance with the IPv6 addresses of the model machines.

```
m2 pri # nano -w medi6.lab.zone  
  
m2 pri # nano -w your.reverse.hexadecimal.prefix.ip6.arpa.zone.  
  
m2 pri# nano -w f.0.3.6.8.a.0.c.2.0.0.2.ip6.arpa.zone  
  
m2 pri# cd  
m2~#
```

Check your typing before saving your files.

DNS service daemon is called “named “. “named “ system user, which has daemon control, must have the “write” permissions on the /var/run/named directory. Before starting the service, give the directory ownership to the named user, using the “chown “ (change owner) command.

```
m2~# chown named /var/run/named
```

A startup script, in /etc/init.d directory, makes some control before starting the service. Launch the service with “/etc/init.d/named start” command.

```
h1~# /etc/init.d/named start
```

The machine then displays.

```
starting Named [ok]
```

Verify all is OK, displaying the tail of the system log.

```
h1~# tail /var/log/messages
```

An error message means a misconfiguration, you have to correct before going further. In that case do the modifications and restart the service using `/etc/init.d.named restart` command.

On m2 machine edit DNS client config file (`/etc/resolv.conf`) to set attached DNS server IPv4 address to 192.168.99.254.

```
m2~# nano -w /etc/resolv.conf
```

Check DNS service requesting the server using the `nslookup` command.

**Note :** `nslookup` command tend to be replaced by `dig` tool, which is more advanced. You can display manual page for `dig` with "`man dig`" command.

"`nslookup`" command works in interactive mode. A new prompt is displayed, as a ">". Set your request type to value "ANY" to query DNS database about all RR (Resource Record) types. Then launch a query to about m2 machine, `g1.medi6.lab` and then test the reverse lookup querying the address with the interface identifier "1" of the g1 machine.

```
m2~# nslookup
> set type=ANY
>m2.
.
.
.
>g1.medi6.lab
.
.
.
>fdxx:yyyy:zzzz:a::1
.
.
.
g2

>exit
```

Medi6 model DNS service is now up.

We have now to configure `totd` service on g1 machine. Before starting the service observe `totd` config file content (`/etc/totd.conf`).

```
g1~# less /etc/totd.conf
```

**Note :** To quit "`less`" command display just type "q" key (quit).

***What is the machine address to which DNS requests will be relayed ? (forwarder parameter). Which machine of the model does it correspond ?***

***What is the prefix value which will be added to machine with only a type-A DNSrecord ?***

**On which port the service is listening ?**

Now launch the service

```
g1~# /etc/init.d/totd start
```

The machine then displays.

```
starting totd... [ok]
```

Check all is OK, displaying the end of the log file using “ tail ” command

```
g1~# tail /var/log/messages
```

On g1 and m1 machine set your DNS client pointing to the auxiliary DNS relay address fdxx:yyyy:zzzz:a::1, replacing string fdxx:yyyy:zzzz with your unique local prefix.

```
g1~# nano -w /etc/resolv.conf
```

```
m1~# nano -w /etc/resolv.conf
```

Before checking auxiliary DNS relay, start a wireshark capture on **eth1** interface of **m1** machine and another one on **eth2** interface of **g1** machine.

From m1 machine test DNS relay using nslookup command.

```
m1~# nslookup
> set type=ANY
>m2.medi6.lab
.
.
.
>m3.medi6.lab

>exit
```

**On nslookup result display :**

- what is the IPv6 address for m2.medi6.lab machine ?
- What is its prefix value ? Why ? Find IPv4 address of m2 in the lower part of the IPv6 address.
- Why m3 machine has not a such prefixed address ?

Stop the two wireshark captures.

Observe these captures and especially “ DNS standard query “ and “ DNS standard response “ datagrams for the two requests. For m2 machine observe the transformation operated by the relay from “A” type response to “AAAA” type response.

### 3.3.c.1 ) Using TOTD through the 6to4 tunnel

On g2 and m3 machine set your DNS client pointing to the auxiliary DNS relay on **::192.168.99.1** and **fdxx:yyyy:zzzz:a::1** replacing fdxx:yyyy:zzzz string with you unique local prefix.

**Note :** On remote IPv6 network machines we must specify both addresses **::192.168.99.1** and **fdxx:yyyy:zzzz: a:: 1** in that order. In fact, GNU / Linux TOTD version does not allow to select the listening interface service. By default the service is listening on all interfaces. It's the 6to4 tunnel interface (address: 192.168.99.1) which is responding to DNS requests. If you do not mention the latter as the first DNS server connection, the DNS client displays an alert, suspecting a DNS spoofing

```
g2~# nano -w /etc/resolv.conf
```

```
m3~# nano -w /etc/resolv.conf
```

From m3 machine, check DNS relay using `nslookup` command.

```
m3~# nslookup
> set type=ANY
>m2.medi6.lab
.
.
.
>m3.medi6.lab

>exit
```

### 3.3.d ) PTRTD the transport layer relay under GNU/Linux

TRT (Transport Relay Translator RFC3142) is a translation mechanism of transport level for IPv6-only machines. It is implemented on a dual stacks machine on the network border and requires a DNS translation service such as TOTD. The datagrams, to relay, have a destination address whose prefix matches the prefix added by the DNS relay (in our case **3ffe:abcd:1234:9876:: / 64** given our totd configuration). The role of the TRT relay is to accept the TCP or UDP streams from these addresses to impersonate the IPv4 client in IPv4 world. In a TCP connection case, the TRT service establishes two separate connections for which it will ensure the abutment. A TCP/IPv6 connection between the IPv6 client and the TRT relay and a TCP/IPv4 connection between the TRT relay and the IPv4 server.

### 3.3.e ) Setting up PTRTD

On medi6 model, it's g1 machine again that will support the service. Before starting the service,

display the content of the IPv6 routing table

```
g1~# ip -6 route show
```

Check, in “ /etc/conf.d/ptrtd “ conf file, that the prefix is in accordance with the one added by tottd service (3ffe:abcd:1234:9876:: in our case).

```
g1~# less /etc/conf.d/ptrtd
```

**Note :** To quit “ less ” command display just type “q” key (quit).

Start the service using “ /etc/init.d/ptrtd start “ command..

```
g1~# /etc/init.d/ptrtd start
* Starting ptrtd ... [ ok ]
```

Display again the routing table content to observe a new entry for 3ffe:abcd:1234:9876::/64 prefix destination to internal interface named “ tap0 “.

The service is now operational. To test we can't rely on the usual commands “ ping6 “ or “ tracepath6 “. These latter are in fact based on the analysis of ICMPv6 message return. As ICMP is a protocol below the transport layer, it is not relayed through our transport relay gateway. We'll conduct our tests using the Netcat tool ( “ nc “ command). Netcat is presented as the TCP/IP Swiss Army knife. It allows standard I/O redirection on a TCP or UDP flow to a remote machine just like shell unix pipes symbolized by the character “|”. The standard input is redirected to a TCP or UDP socket and everything that is received back in that socket is redirected the standard output. As part of this lab medi6, we will use the IPv6 version of Netcat tool, i.e. the “ nc6 “ command.

**Note :** for more details on “ nc6 “ command you can display its manual page using “ man nc6 “ command or see these web pages :

« <http://www.jfranken.de/homepages/johannes/vortraege/netcat.en.html> » ou encore la page suivante  
« <http://www.jfranken.de/homepages/johannes/vortraege/netcat/README.txt> »

The test is from the m1 machine to display the output of “ /bin/hostname ” run on the m2 machine. We'll conduct this test on a TCP connection on port 6666, then the same test will be conducted through a UDP stream on port 6667.

Before running the tests, start wireshark capture on the **eth1** interface of the **m1** machine and another capture on the **eth2** interface of the **g1** machine.

On the m2 machine, open a socket listening on TCP port 6666 to execute the command “ /bin/hostname ” by running the command “ nc6 -l -p 6666 -e /bin/hostname”.

```
m2~# nc6 -l -p 6666 -e /bin/hostname
```

On m1 machine launch the client part of the test using the “ nc6 m2.medi6.lab 6666 “ command and hit “enter key” twice. The m1 screen will then display “m2” string.

```
m1~# nc6 m2.medi6.lab 6666
m2
```

```
m1~#
```

Stop the captures and observe datagrams.

**On eth1 interface of m1 :**

**- Look at the DNS request and response message. Who is destination machine for the request? What record type is requested. What is the answer ?**

**- What is the address/port source and destination of the TCP connection?**

**Same questions on eth2 interface of g1.**

**On m1 and g1 captures rebuild the TCP stream. Click on a frame of the a TCP stream and then right click the menu "Analyze → Follow TCP stream". Are the application flows exchanged identical ?**

Clear wireshark display filters and restart captures on the same interfaces.

Launch the UDP flow test chaining the following command.

On m2

```
m2~# nc6 -u -l -p 6667 -e /bin/hostname
```

On m1 use the following command and then hit "return key" twice". After "m2" string displayed hit simultaneously the "ctrl-C" keys to leave the client.

```
m1~# nc6 -u m2.medi6.lab 6667

m2
```

Stop captures and observe datagrams.

**On eth1 interface of m1 what address / port source and destination of the TCP connection?**

**Same question for eth2 interface of g1.**

Clear wireshark display filters and restart captures on the same interfaces.

We'll now proceed to an ssh session through the gateway. From m1 machine launch an ssh session to the m2 machine, accept the new session key and then chain the commands "whoami", "ls -al" and "exit".

**Note:** From a security standpoint, this configuration is not totally satisfactory. Although it uses an ssh session, which is, a priori, encrypted end to end, the ptrtd relay is an entry point for a possible "Man in the middle attack". A modified version of the ptrtd relay could, instead of simply relaying seamlessly TCP flow, negotiate a intermediary key in the initialization process of the session and decrypt ssh

stream received before re-encrypt with its own key on the output connection. The attacker would then have ample opportunity to observe the flow in the clear. This is one of the major limitations of this type of transport gateway. What level of confidence may be given to the gateway ? There would therefore be cautious in activating the parades on both SSH client and SSH server sides as managing a rigorous dual client and server authentication.

```
m1~# ssh m2.medi6.lab
m2~# whoami
m2~# ls -al
m2~# exit
m1~#
```

Observer wireshark captures on m1 and g1.

### 3.3.e.1 ) Using PTRTD through 6to4 tunnel

On g2 router add a routing table entry to direct 3ffe:abcd:1234:9876::/64 destination datagrams into the 6to4 tunnel.

```
g2~# ip -6 route add 3ffe:abcd:1234:9876::/64 via ::192.168.99.1 dev
tunnel624 metric 1
```

Restart wireshark captures on different check points : m3 eth1 interface, g1 eth2 interface and g2 eth2 interface

From m3 machine, start an ssh session on m2, accept new session keys and then chain "whoami", "ls -al" and "exit" commands.

```
m3~# ssh m2.medi6.lab
m2~# whoami
m2~# ls -al
m2~# exit
m3~#
```

Stop captures and observe source and destination addresses of the datagrams, see also encapsulation levels on the different check points.

The lab is now finished. Stop the model clicking on "Stop/purge" button on the model map. Then you can stop the host clicking the "K" button at the bottom left corner of the desktop, then "Turn Off Computer". Once the DVD is ejected you can switch off the computer.

----- oOo -----