Ethernet/IP interaction emulated with NETKIT. DHCP relay, proxy ARP, Port stealing and ARP poisoning attack.

Marco Bonola, Lorenzo Bracciale

Corso di Reti di Accesso e Trasporto – Tor Vergata
Prof. Stefano Salsano
A.A. 2010-2011

Part 1

Basic Linux networking configuration, DHCP Relay and Proxy ARP with NETKIT

Outline

GOAL 1: practically understand the interaction between Ethernet and IP through 2 simple networking scenarios emulated with NETKIT

1. Introduction to NETKIT
2. tcpdump and wireshark
3. DHCP in Linux – DHCP Relay
4. Proxy ARP

Preliminaries…

Download the tarball with all the NETKIT Lab files:

```
knoppix:$ wget
   http://byron.netgroup.uniroma2.it/~marlon/RAT/
   es2011.tar
```

Extract the tarball:

```
knoppix:$ tar xvf es2011.tar
```

NETKIT

1. a system for emulating computer networks
2. based on uml (user-mode linux)

   ✓ user-mode Linux is a Linux kernel (inner part of the LinuxOS) that can be executed as a user process on a standard Linux box

   ✓ a user-mode Linux process is also called virtual machine (vm), while the Linux box that hosts a virtual machine is called host machine (host)
3. each emulated network device is a virtual Linux box

Emulator vs Simulator

- Emulation: to recreate the behavior of a system, with no regard for how the system functions internally
- Simulation: modeling of the components of a system

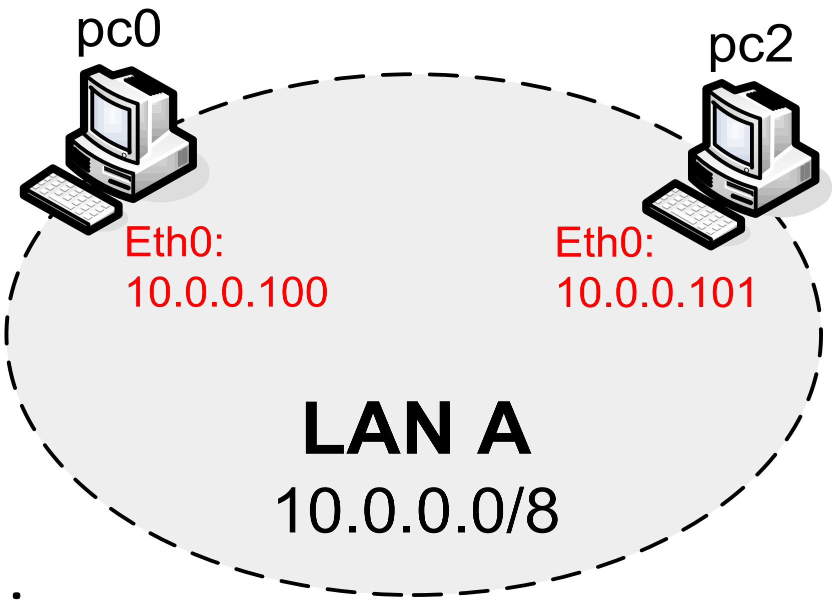
note: the LinuxOS is shipped with software supporting most of the network protocol

hence, any Linux machine can be configured to act as a bridge/switch or as a router

Virtual Machines commands

- **vstart**: starts a new virtual machine
- **vlist**: lists currently running virtual machines
- **vconfig**: attaches network interfaces to running vms
- **vhalt**: gracefully halts a virtualmachine
- **vcrash**: causes a virtual machine to crash
- **vclean**: “panic command” to clean up all netkit processes (including vms) and configuration settings on the hostmachine

Example – LAN, no routing



Start the virtualmachines:

```
knoppix:$ vstart pc1 --eth0=A
knoppix:$ vstart pc2 --eth0=A
```

On pc1:

```
pc1:# ifconfig eth0 10.0.0.101
```

On pc2:

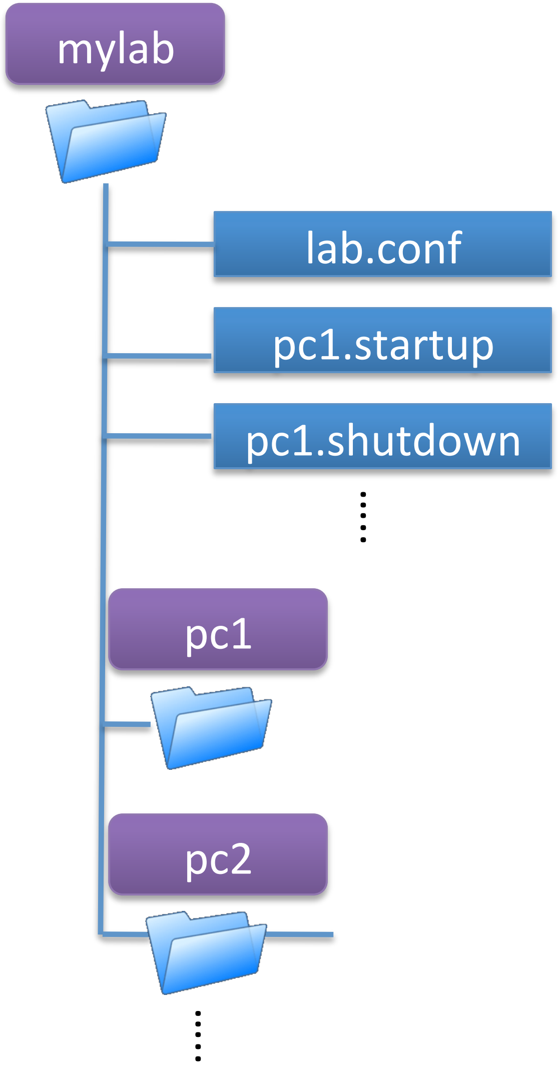
```
pc2:# ifconfig eth0 10.0.0.102
```

Let’s try to ping pc2 from pc1

NETKIT-Lab

Netkit-Lab: automates multiple virtual machine startup. To create a lab we need:

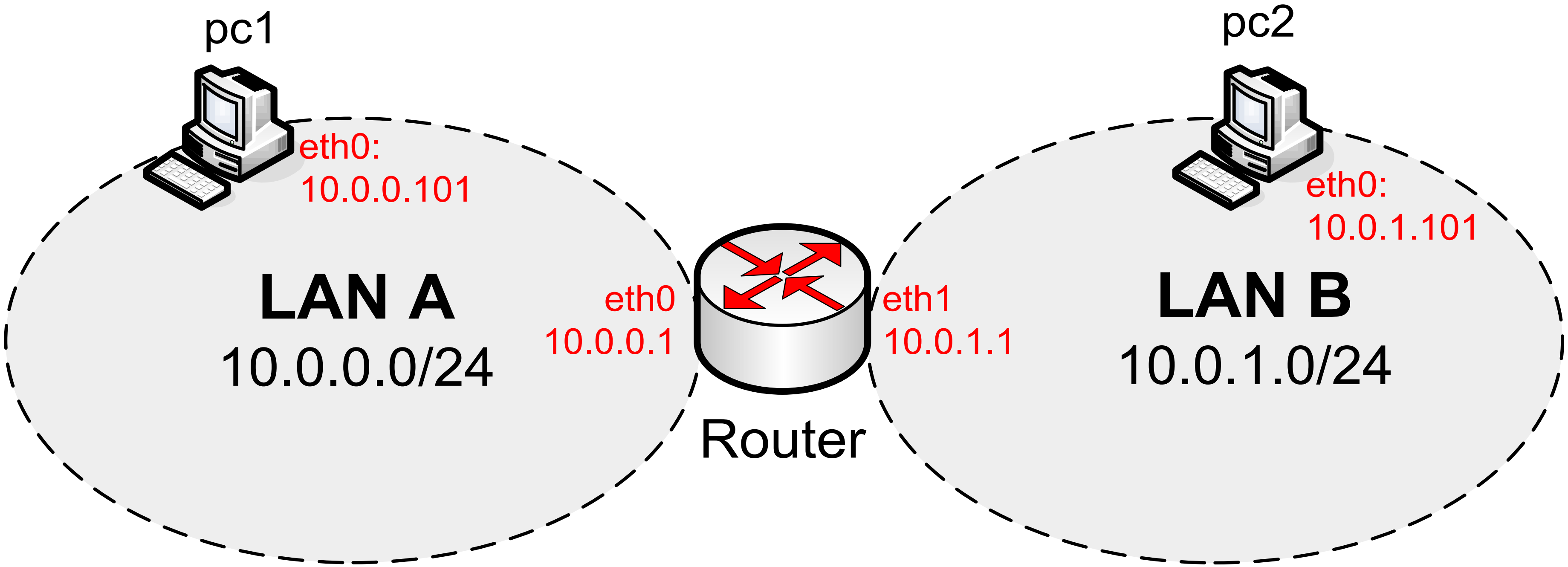
1. a lab configuration file describing the network topology (lab.conf)
2. a set of subdirectories that contain files to overwrite in the vm filesystem. Each folder points to / of the vm named as the folder.
3. [optional] .startup and .shutdown files that describe actions performed by virtual machines when they are started or halted
4. [optionally] a lab.depfile describing dependency relationships on the startup order of virtual machines



NETKIT-Lab commands

- **lstart**: start a netkitlab
- **lhalt**: gracefully halt all vms of a lab
- **lcrash**: cause all the vms of a lab to crash
- **lclean**: remove temporary files from a lab directory
- **linfo**: provide information about a lab without starting it
- **ltest**: run tests to check if the lab is working properly

Example2 – 2 LAN, 1 router



NOTE – the files for this example are in:
`esercitazione-2010/example2-lab/`

Example2 - Lab set-up

lab.conf:

```
router[0]=A
router[1]=B
pc1[0]=A
pc2[0]=B
```

pc1.startup:

```
ip link set eth0 up
ip address add 10.0.0.101/24 dev eth0
ip route add default via 10.0.0.1
```

pc2.startup:

```
ip link set eth0 up
ip address add 10.0.1.101/24 dev eth0
ip route add default via 10.0.1.1
```

Example2 - Lab set-up

router.startup:

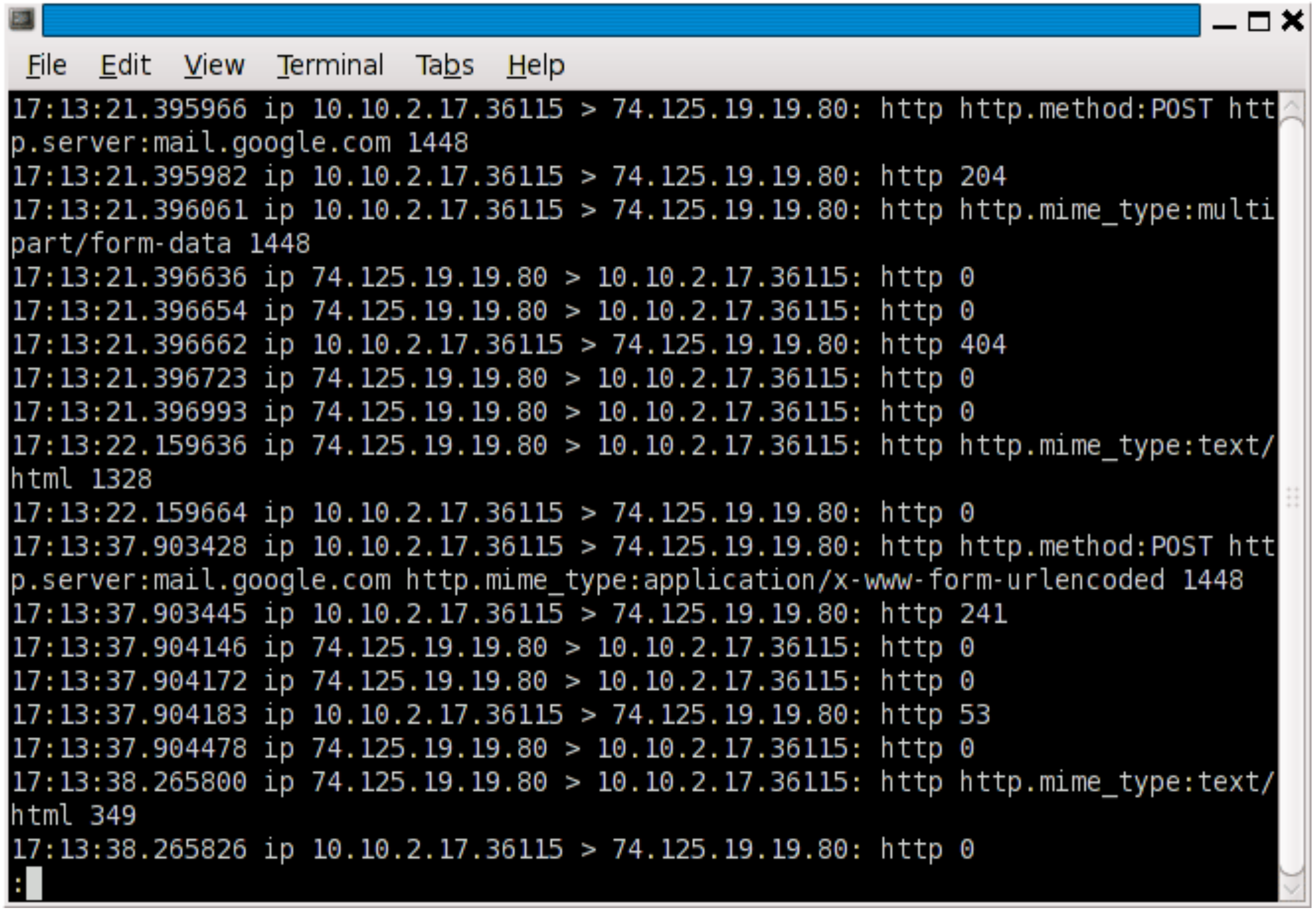
```
ip link set eth0 up
ip link set eth1 up
ip address add 10.0.0.1/24 dev eth0
ip address add 10.0.1.1/24 dev eth1

echo 1 > /proc/sys/net/ipv4/ip_forward
```

to start the LAB:

```
knoppix:$ lstart
```

tcpdump

command line network analyzer



```
17:13:21.395966 ip 10.10.2.17.36115 > 74.125.19.19.80: http http.method:POST htt
p.server:mail.google.com 1448
17:13:21.395982 ip 10.10.2.17.36115 > 74.125.19.19.80: http 204
17:13:21.396061 ip 10.10.2.17.36115 > 74.125.19.19.80: http http.mime_type:multi
part/form-data 1448
17:13:21.396636 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:21.396654 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:21.396662 ip 10.10.2.17.36115 > 74.125.19.19.80: http 404
17:13:21.396723 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:21.396993 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:22.159636 ip 74.125.19.19.80 > 10.10.2.17.36115: http http.mime_type:text/
html 1328
17:13:22.159664 ip 10.10.2.17.36115 > 74.125.19.19.80: http 0
17:13:37.903428 ip 10.10.2.17.36115 > 74.125.19.19.80: http http.method:POST htt
p.server:mail.google.com http.mime_type:application/x-www-form-urlencoded 1448
17:13:37.903445 ip 10.10.2.17.36115 > 74.125.19.19.80: http 241
17:13:37.904146 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:37.904172 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:37.904183 ip 10.10.2.17.36115 > 74.125.19.19.80: http 53
17:13:37.904478 ip 74.125.19.19.80 > 10.10.2.17.36115: http 0
17:13:38.265800 ip 74.125.19.19.80 > 10.10.2.17.36115: http http.mime_type:text/
html 349
17:13:38.265826 ip 10.10.2.17.36115 > 74.125.19.19.80: http 0
:
```

tcpdump – some usage examples

Capture all packets on all interfaces and don’t detect hostnames:

```
tcpdump –i any -n
```

Capture all packets on eth0 and save the trace on file (the whole packets…):

```
tcpdump –i eth0 –w file –s0
```

Capture 10 packets on eth0 to destination `$DEST`:

```
tcpdump –i eth0 –c 10 dst host $DEST
```

Capture all HTTP packets on eth0:

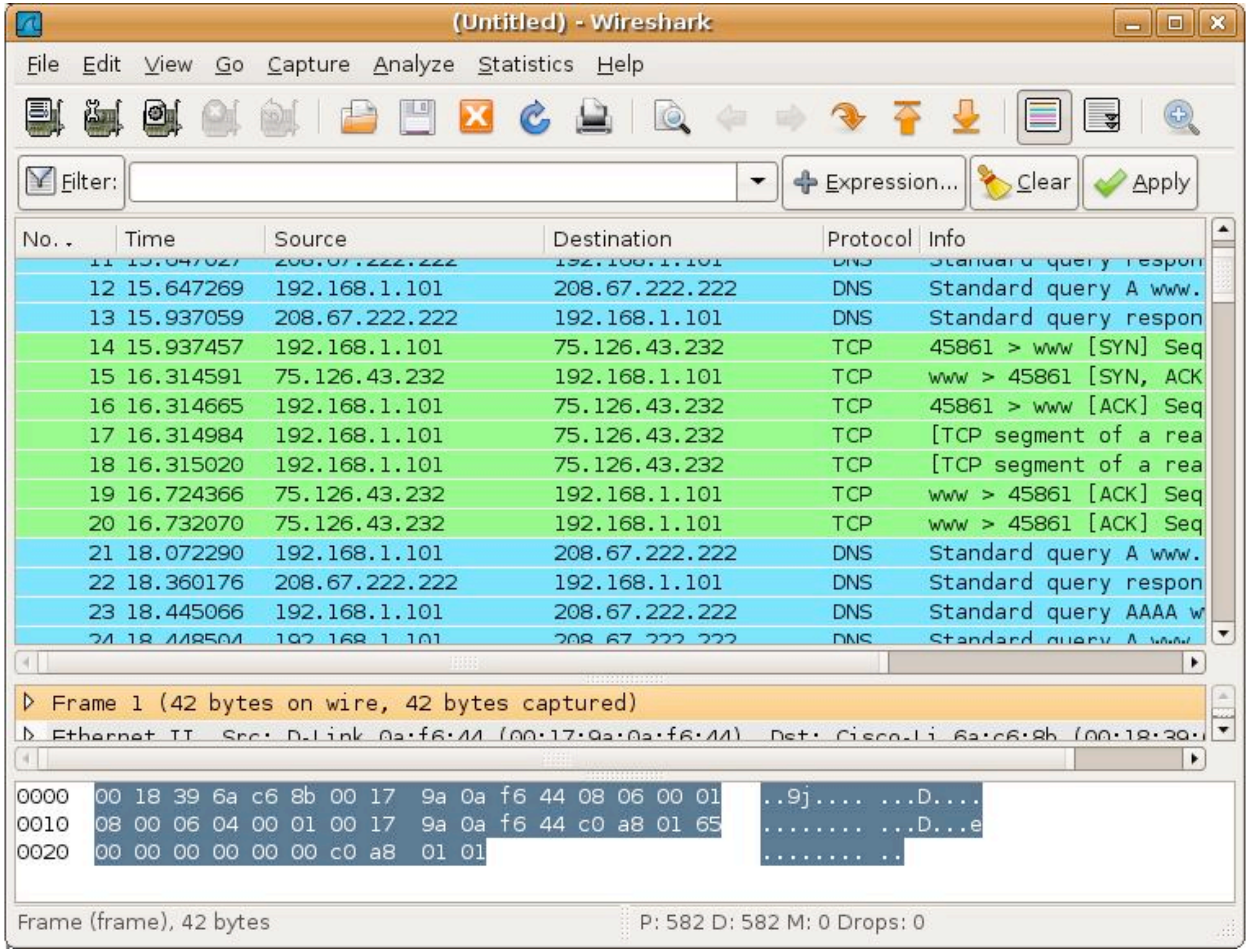
```
tcpdump –i eth0 tcp port 80
```

Capture all packets with destination or source address != $ADDR and port in the range [10000:20000]:

```
tcpdump –i eth0 host not $ADDR portrange 10000-20000
```

Wireshark

THE Network Analyzer



No.	Time	Source	Destination	Protocol	Info
11	15.647027	208.67.222.222	192.168.1.101	DNS	Standard query respon
12	15.647269	192.168.1.101	208.67.222.222	DNS	Standard query A www.
13	15.937059	208.67.222.222	192.168.1.101	DNS	Standard query respon
14	15.937457	192.168.1.101	75.126.43.232	TCP	45861 > www [SYN] Seq
15	16.314591	75.126.43.232	192.168.1.101	TCP	www > 45861 [SYN, ACK
16	16.314665	192.168.1.101	75.126.43.232	TCP	45861 > www [ACK] Seq
17	16.314984	192.168.1.101	75.126.43.232	TCP	[TCP segment of a rea
18	16.315020	192.168.1.101	75.126.43.232	TCP	[TCP segment of a rea
19	16.724366	75.126.43.232	192.168.1.101	TCP	www > 45861 [ACK] Seq
20	16.732070	75.126.43.232	192.168.1.101	TCP	www > 45861 [ACK] Seq
21	18.072290	192.168.1.101	208.67.222.222	DNS	Standard query A www.
22	18.360176	208.67.222.222	192.168.1.101	DNS	Standard query respon
23	18.445066	192.168.1.101	208.67.222.222	DNS	Standard query AAAA w
24	18.448504	192.168.1.101	208.67.222.222	DNS	Standard query A www.

```
▷ Frame 1 (42 bytes on wire, 42 bytes captured)
▷ Ethernet II, Src: D-Link_0a:f6:44 (00:17:9a:0a:f6:44), Dst: Cisco-Li_6a:c6:8b (00:18:39:
```

```
0000  00 18 39 6a c6 8b 00 17  9a 0a f6 44 08 06 00 01   ..9j.... ...D....
0010  08 00 06 04 00 01 00 17  9a 0a f6 44 c0 a8 01 65   ........ ...D...e
0020  00 00 00 00 00 00 c0 a8  01 01                     ........ ..
```

Frame (frame), 42 bytes   P: 582 D: 582 M: 0 Drops: 0

We can use wireshark to graphically display on the host machine the trace captured with tcpdump….

Let’s see some real packets..

Let’s try with a ping from PC1 to PC2.

Before sending the pings, let’s run tcpdump on “any device” and save the output on a file.

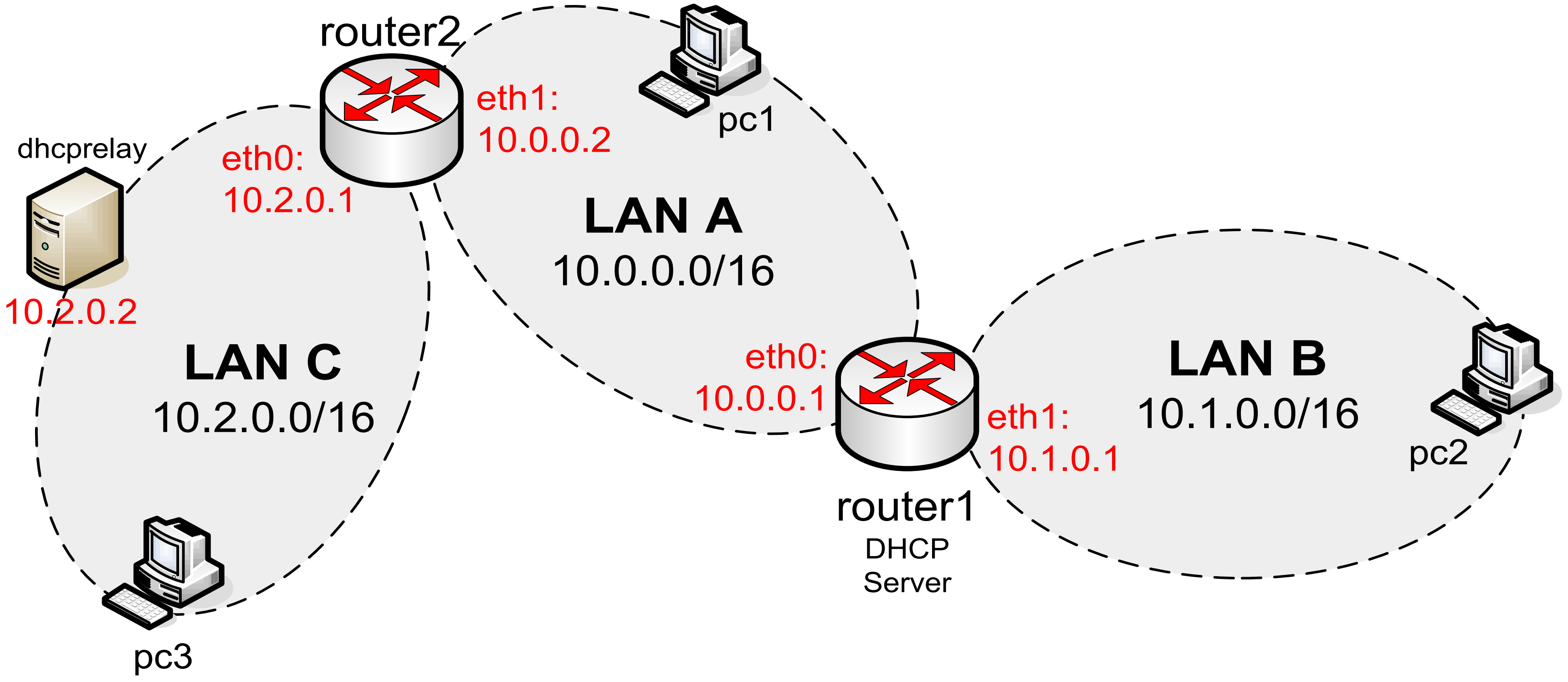
```
router:# tcpdump -i any -w /hosthome/prova.pcap
```

Now open the file with wireshark on the host machine:

```
knoppix:$ wireshark /home/knoppix/prova.pcap
```

note: hosthome in the vm is the home of the user that has launched the vm in the host machine

Example 3 – DHCP Server and Relay



NOTE – the files for this example are in:

```
esercitazione-2010/example3-lab/
To start the lab use: ./start_lab
```

At first boot the DHCP RELAY won't start...

Example 3 – Lab set-up

lab.conf:

```
router1[0]=A
router1[1]=B
router1[mem]=64

router2[0]=C
router2[1]=A
router2[mem]=64

dhcprelay[0]=C
dhcprelay[mem]=64

pc1[0]=A
pc2[0]=B
pc3[0]=C
```

pc1.startup, pc2.startup, pc3.startup:

```
dhclient eth0
```

Example 3 – Lab set-up

router1.startup:

```
ip link set eth0 up
ip link set eth1 up
ip address add 10.0.0.1/16 dev eth0
ip address add 10.1.0.1/16 dev eth1

ip route add  10.2.0.0/16 via 10.0.0.2

/etc/init.d/dhcp3-server start

echo 1 > /proc/sys/net/ipv4/ip_forward
```

Example 3 – Lab set-up

router1/etc/dhcp3/dhcpd.conf:

```
default-lease-time 3600;

subnet 10.0.0.0 netmask 255.255.0.0 {
       range 10.0.0.100 10.0.0.254;
       option routers 10.0.0.1;
}

subnet 10.1.0.0 netmask 255.255.0.0 {
       range 10.1.0.100 10.1.0.254;
       option routers 10.1.0.1;
}

subnet 10.2.0.0 netmask 255.255.0.0 {
       range 10.2.0.100 10.2.0.254;
       option routers 10.2.0.1;
}
```

Example 3 – Lab set-up

router2.startup:

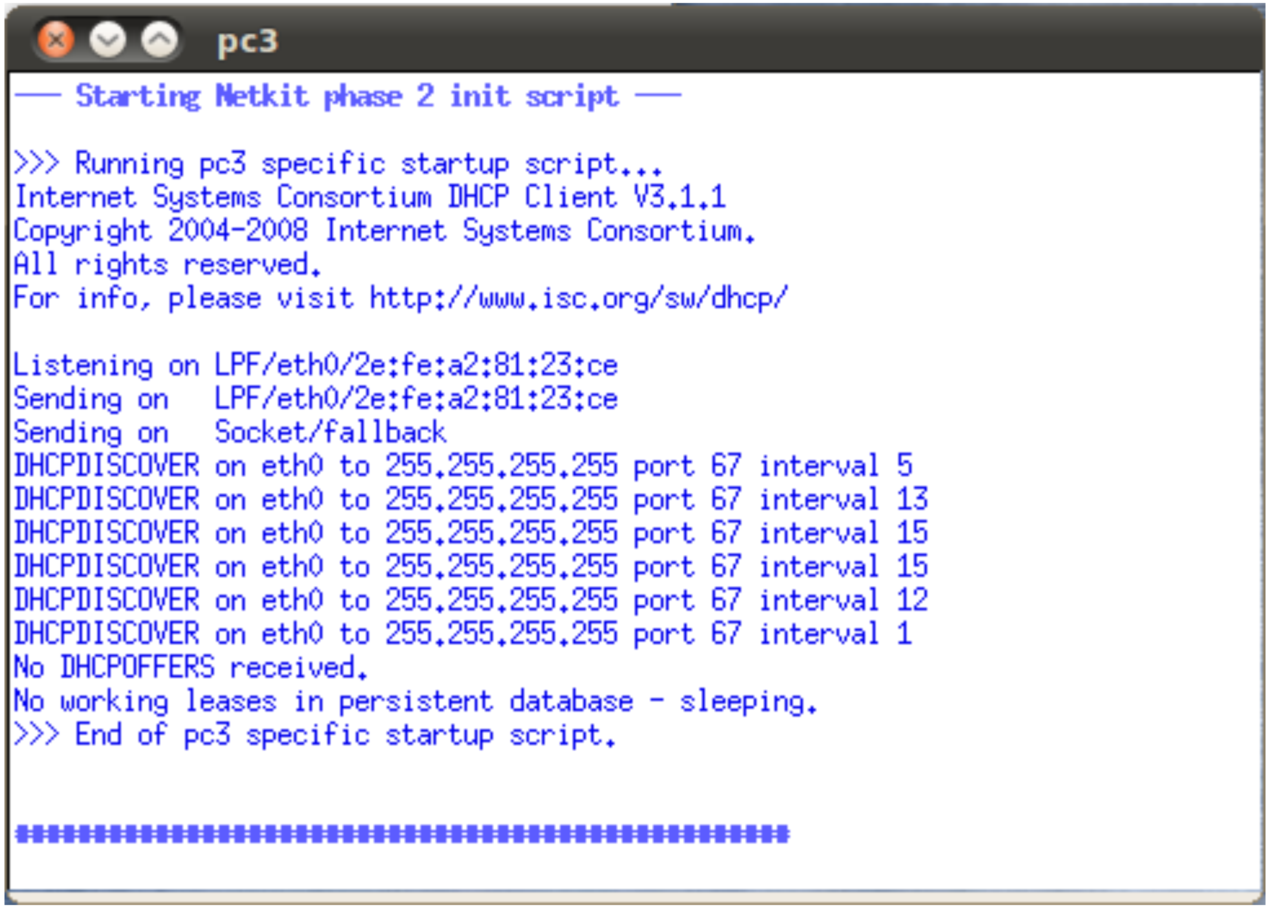
```
ip link set eth0 up
ip link set eth1 up
ip address add 10.2.0.1/16 dev eth0
ip address add 10.0.0.1/16 dev eth1
ip route add 10.1.0.0/16 via 10.0.0.1


echo 1 > /proc/sys/net/ipv4/ip_forward
```

dhcprelay.startup:

```
ip link set eth0 up
ip address add 10.2.0.2/16 dev eth0
ip route add default via 10.2.0.1

#!! No dchp relay configured and run!
```

PC3 couldn’t contact DHCP server…

SURE! The DHCP server is not in LAN C and broadcast packets doesn’t get through the router….



```
--- Starting Netkit phase 2 init script ---

>>> Running pc3 specific startup script...
Internet Systems Consortium DHCP Client V3.1.1
Copyright 2004-2008 Internet Systems Consortium.
All rights reserved.
For info, please visit http://www.isc.org/sw/dhcp/

Listening on LPF/eth0/2e:fe:a2:81:23:ce
Sending on   LPF/eth0/2e:fe:a2:81:23:ce
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 5
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 13
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 15
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 15
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 12
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 1
No DHCPOFFERS received.
No working leases in persistent database - sleeping.
>>> End of pc3 specific startup script.


##################################################
```

How to make pc3 get IP configuration from the DHCP server

Install dhcrelay package on dhcprelay VM (the files are in /root).

```
dhcprelay:~# dpkg –i *.deb
(ignore the configuration wizard..)
```

Run dhcrelay manually:

```
dhcprelay:~# dhcrelay –d –i eth0 10.0.0.1
```

usage:

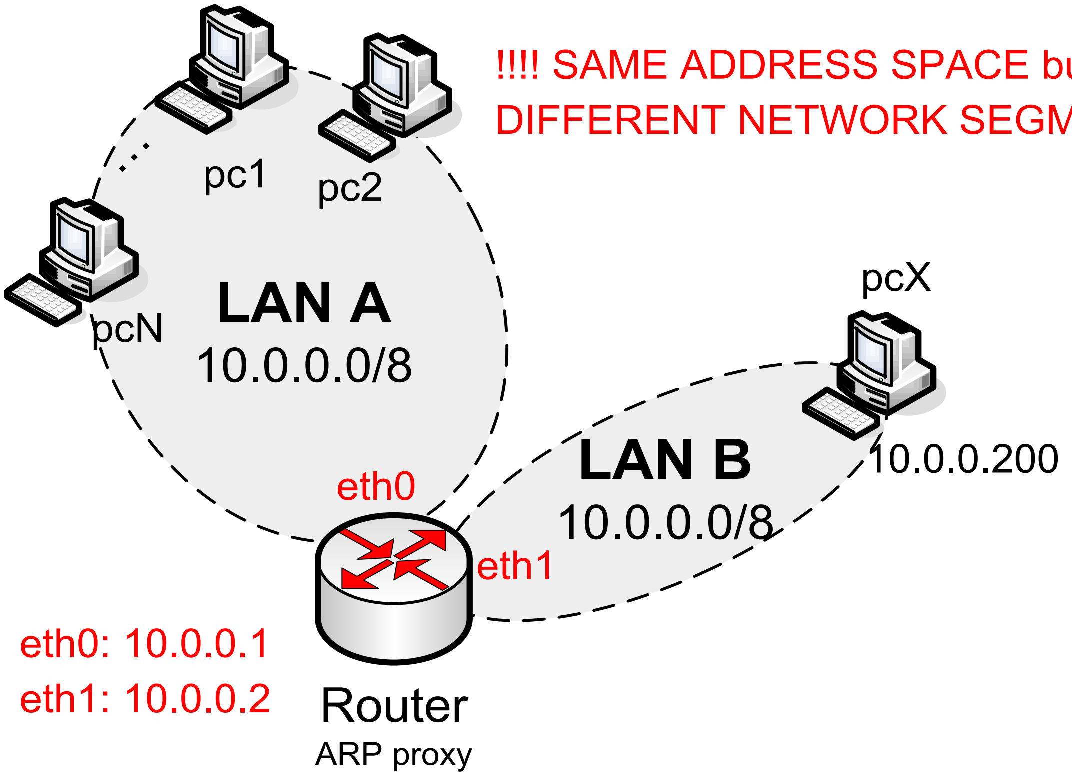
```
dhcrelay [options] DHCP_SERVER_ADDRESS
-i <ifaces>: interface to listen on
-d: don’t go in background
```

Run dhcrp client on pc3:

```
pc3:~# dhclient eth0
```

→ **OK**

Example 4 – ARP Proxy



!!!! SAME ADDRESS SPACE but DIFFERENT NETWORK SEGMENTS

lab.conf:

```
router[0]=A
router[1]=B
router[mem]=64

pc1[0]=A
pc2[0]=A

pcX[0]=B
```

NOTE – the files for this example are in:

```
esercitazione-2010/example4-lab/
To start the lab use: ./start_lab
```

Example 4 – Lab set-up

pc1.startup:

```
dhclient eth0
ip link set eth0 address 00:00:00:00:00:01
```

pc2.startup:

```
dhclient eth0
ip link set eth0 address 00:00:00:00:00:02
```

pc3.startup:

```
ip link set eth0 address 00:00:00:00:00:03
dhclient eth0
```

pcX.startup:

```
ip link set eth0 up
ip link set eth0 address 00:00:00:00:00:ff
ip address add 10.0.0.200/8 dev eth0
ip route add default via 10.0.0.2
```

Example 4 – Lab set-up

router.startup:

```
ip link set eth0 up
ip link set eth1 up
ip link set eth0 address 00:00:00:00:00:aa
ip link set eth1 address 00:00:00:00:00:bb
ip address add 10.0.0.1/8 dev eth0
ip address add 10.0.0.2/8 dev eth1
ip route flush dev eth1
ip route add 10.0.0.200 dev eth1

/etc/init.d/dhcp3-server start

echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/ipv4/conf/all/proxy_arp
```

router/etc/dhcp3/dhcpd.conf:

```
default-lease-time 3600;
subnet 10.0.0.0 netmask 255.0.0.0 {
      range 10.0.0.10 10.0.0.150;
      option routers 10.0.0.1;}
```

What’s going on…

To better understand the messages flow between pc1 and pcX let’s capture the traffic separately on eth0 and eth1

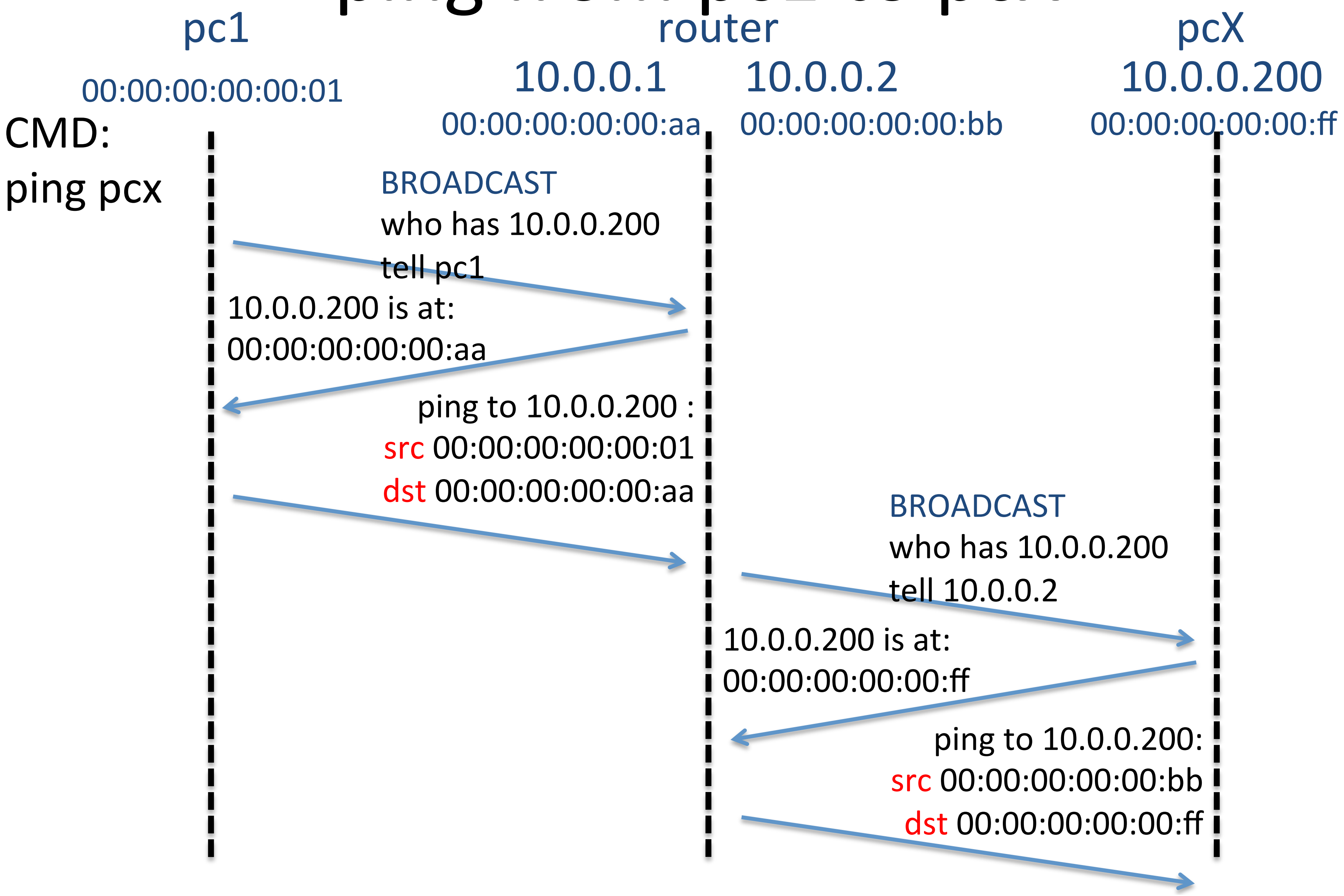
```
router:# tcpdump –i eth0 –w /hosthome/eth0.pcap&
router:# tcpdump –i eth1 –w /hosthome/eth1.pcap&
```

```
knoppix:$ wireshark /home/knoppix/eth0.pcap&
knoppix:$ wireshark /home/knoppix/eth1.pcap&
```

To close all tcpdump processes:

```
router:# killall tcpdump
```

ping from pc1 to pcX



pc1

00:00:00:00:00:01

router

10.0.0.1

00:00:00:00:00:aa

10.0.0.2

00:00:00:00:00:bb

pcX

10.0.0.200

00:00:00:00:00:ff

CMD:
ping pcx

BROADCAST
who has 10.0.0.200
tell pc1

10.0.0.200 is at:
00:00:00:00:00:aa

ping to 10.0.0.200 :
src 00:00:00:00:00:01
dst 00:00:00:00:00:aa

BROADCAST
who has 10.0.0.200
tell 10.0.0.2

10.0.0.200 is at:
00:00:00:00:00:ff

ping to 10.0.0.200:
src 00:00:00:00:00:bb
dst 00:00:00:00:00:ff

Some questions…

Q1: What happens for ping from pcX to pc1?

Q2: Why is this called “transparent routing”?

Q3: So, can I make it work without ARP proxy? (remember to flush arp cache, otherwise it will works even without Proxy ARP)

Q4: If so, how?

Q5: what is the difference with respect to the proxy ARP way?

Answers

A1: it’s just symmetric…

A2: because (i) the ping packet is actually routed (the src mac address is changed by router), but (ii) pc1 is not aware (in fact, it would works even without the default gw route)

A3: with “standard” routing.

A4: since the network prefix of pcX is the same as pc1, we need “per host” routes to pcX via 10.0.0.1 in pc1. In pcX we needs per host routes to all hosts in lanA via 10.0.0.2 (or even better, a per host route to 10.0.0.2 and a route to net 10.0.0.1/8 via 10.0.0.2).
Note: remember “longest prefix matches first”…

A5: that pc1 is aware that to reach pcX is using router to forward the packet (see the trace).

If you wanna try…

Disable proxy ARP on router:

```
echo 0 > /proc/sys/net/ipv4/conf/all/proxy_arp
```

Try to ping….. but it still works!! LIAR!!!! Why? See the ARP cache:

```
ip n
```

Flush the ARP cache on pc1 and pcX and retry.

```
ip n flush dev eth0
```

OK, now it doesn't work…

Change routes on pcX

```
ip route del 10.0.0.0/8
ip route add 10.0.0.2 dev eth0
ip route add 10.0.0.0/8 via 10.0.0.2
```

Change routes on pc1 and ping pcX

```
ip route add 10.0.0.200 via 10.0.0.1
ping 10.0.0.200
```

→ **OK**

One more thing…

I can reuse the same address for eth1 and eth0 on router.

Try it! Restart the lab

```
knoppix:$ lhalt && ./start_lab
```

Delete 10.0.0.2 on eth1 at router and add route to pcX

```
router:#  ip address del 10.0.0.2 dev eth1
router:#  ip route add 10.0.0.200 dev eth1
router:#  ip address add 10.0.0.1 dev eth1
```

Ping pcX from pc1

```
pc1:# ping 10.0.0.200
```

Uses of Proxy ARP

- Joining a broadcast LAN with serial links (e.g., dialup or VPN connections)
- Taking multiple addresses from a LAN
- Placing a server behind a firewall without changing the network configuration
- Mobile-IP
- Transparent subnet gatewaying

From: http://en.wikipedia.org/wiki/Proxy_ARP