v1.2

Universidad de León
Bachelor Degree on Computer Science and Engineering
Course on Computer Networks

CN Practice on Socket Programming and Wake-On-Lan

--- Study Guide ---

The following study guide outline is not to be included in your LabBook writeup:

1. Have the textbook by Peterson & Davie at hand. Most of the material that we have taught so far belongs in book chapters 1 and 2. Find the 6th edition to the book, here:

   https://github.com/SystemsApproach/book/releases/download/v6.1/book.pdf

2. A valuable resource as you undertake the practice exercises contained in **WH4-Practice** is the practices that we did the past academic year. Source code and guiding explanations can be found in the practice scripts.

   http://paloalto.unileon.es/cn/

3. **In the present WH, we use a program that accesses the Ethernet** directly, its name is ***magic*** and it has been stored for you in your remote account. Programs that access the network or datalink layers directly require a Linux capability known as *Raw Socket Capability*. This capability is usually limited to the system administrator (The root user), but you need it so that the programs that you make can successfully open the raw socket successfully. When I enable the remote access to Lab B6, I start a service for having each of your programs conveniently granted the CAP_NET_RAW capability. You simply have to redirect the file's full path name to a Linux fifo and soon the client process listening on the fifo's read side will grant your program the needed capability. Below, you will find finer details about this process.

4. **Include the solutions to the practice exercises in your LabBook writeup.**

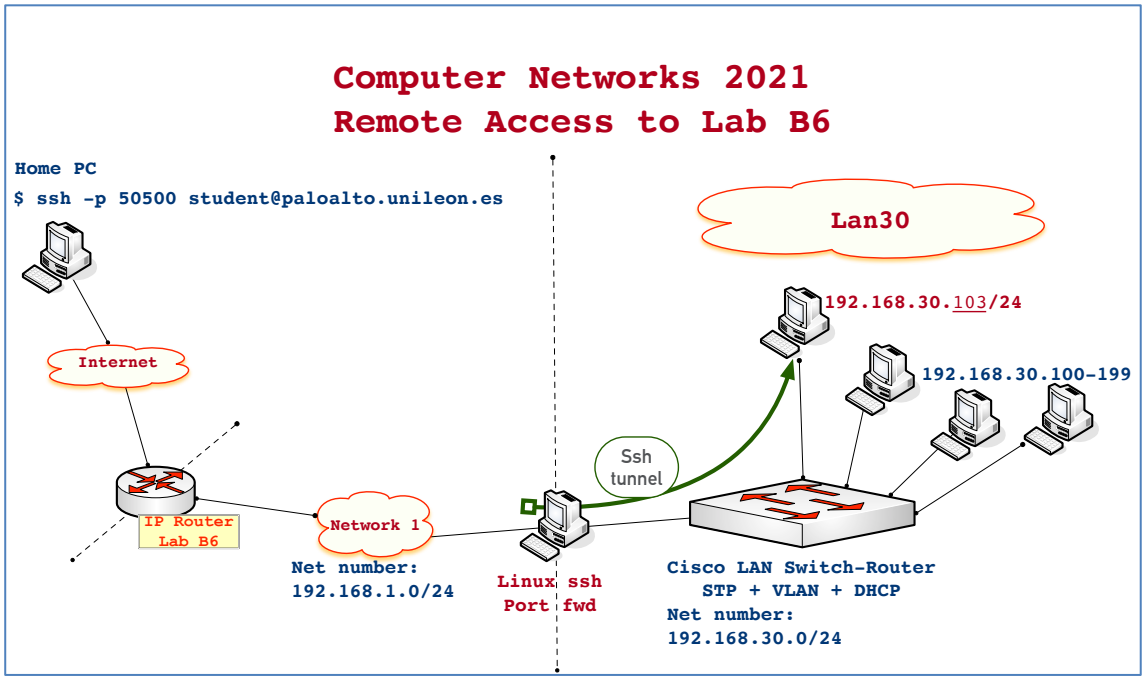

Figure1. Lan30 offers 4 hosts for practicing programming with Linux Packet Sockets

Exercises for practice

1. **Remote execution and monitoring of the "magic" program, which** accesses the raw socket PF_PACKET interface. I recommend that you review your notes from the Practice about Network Interfaces and from the Network Architecture lesson for the concept of Service Interface and Sockets; also, you can consult chapter 1 from the textbook by P&D and our lecture slides from Chapter 1.

In paloalto.unileon.es port 50500, where you log in, granting each individual student to have root privileges is not possible remotely since all the machines are being shared across different courses by numerous students. In past practices you downloaded an executable file (A program) from paloalto.unileon.es which name was **`send-magic-to-22`**. Since you were executing that program in your own personal computer, you could grant CAP_NET_RAW capability. You were able to grant your Linux user that capability because you probably belong to the administrators group, thereby authorizing you to apply a capability by simply doing **sudo**. However, this is not possible when remotely working at paloalto.unielon.es since the hosts used are shared by multiple users.

Allowing each student to apply the CAP_NET_RAW capability as you did in past practices when working in your personal Linux is precluded in this case since applying capabilities in turn requires root privileges. Consequently, I've devised a procedure for students to have their programs applied the CAP_NET_RAW which doesn't require root privileges. It consists of *sending the full path name of your executable to a certain Linux fifo*; the server listening on the

reading end of the fifo will take care of appropriately applying the CAP_NET_RAW capability without requiring root privileges. Please, use this mechanism with diligence and only for the academic purposes it was conceived for.

Assume that you have compiled a program that creates a PF_PACKET raw socket which name is "**magic".** Assume that your user name is **student0**. Actually, that "**magic"** example executable file was already compiled and stored in your paloalto.unileon.es account home directory: /home/student0. To have the CAP_NET_RAW set on file **magic** without having root privileges, proceed as in the outline below (The shell prompt is `[internal 11] $`).

a. **Access your account** at paloalto.unileon.es (Consult WH_3 if necessary) using the login name and password that I sent you to your Unileon e-mail address. **RemLabB6** must be open.

b. Once you are logged in, execute the commands in the following outline - make sure you replace the student0 user name by yours.

```
[internal 11] $ whoami
student0

[internal 11] $ pwd
/home/student0

[internal 11] $ echo /home/student0/magic > /home/administrator/fifo.cn
```

Be attentive to send the full path name of your file to fifo.cn, otherwise, your program will not be located by the capability-granting process. A few seconds afterwards, if you check whether *magic* has the CAP_NET_RAW capability, you'll observe that it does have it:

```
[internal 11] $ setcap -v 'CAP_NET_RAW=epi' magic
magic: OK
```

This is the mechanism enabled by me for you to obtain the raw socket capability for your programs without having root privileges. Incidentally, the example program, **magic**, offers a convenient functionality for us when accessing Lab B6: It allows us to wake up any of the PCs available in Lan30. That is necessary to save electrical energy. When we need a PC to be powered up, we simply execute **magic** and pass it the correct command line arguments.

c. Program **magic** sends a *standard* Ethernet frame that is capable of waking up specific hosts in our LAN (The name of this frame is **magic packet**). We'll use **magic** in this practice to wake up hosts 192.168.30.100, 192.168.30.101, 192.168.30.102 or 192.168.30.103, when necessary. We'll delve into the technical details about **magic** in WH_5-Practice. For the time being, it suffices that you execute **magic** in **in Lan30** to power up a PC and that you observe the Ethernet frame sent by magic from the PC you

are logged in. Find the 192.168.30.* IP address of the host you logged in by executing **ifconfig**.

d. Note that, in this WH, you are executing magic remotely and that you need to observe the sent traffic remotely, also. The utility for observing traffic remotely is **tcpdump**, a command-line program that captures traffic like Wireshark but without a window-based GUI, which makes it very convenient at this time. Find one of the 4 PCs that belong to Lan30 which is not powered-up. Send it ping and check that is does not respond. Capture the ping responses that you have obtained, if any. Usually, the granted addresses are the lowest ones in the range (See Fig.1). If ping indicates that we are receiving ICMP Echo Replies from some machine, then login into it, record its MAC address (Ethernet HW Address) and shut it down. You'll need to pass that MAC address to the ***magic*** program.

e. We want to capture the traffic generated by ***magic*** (The magic packet), so you will need to create a new ssh session with port 50500 at paloalto.unileon.es with your user name. Create that connection now and type the following command at the shell prompt which will capture the desired traffic (Replace enp1s0 below for the correct NIC name, if necessary):

```
[internal 11] $ tcpdump -i enp1s0 ether proto 0x0942
```

The `tcpdump` command captures traffic on NIC `enp1s0`; that captured frames will be restricted to those having an Ethertype value of 0x0942. This is the *standard* multiplexing key used for Magic Packets. (Recall from CN Practice 1 that you can obtain a full listing of available network interfaces by issuing an ifconfig command). We will study the **magic packet** more deeply on an upcoming practice.

f. Now, we can proceed to wake up the intended PC by sending it the magic packet. On your other session (On another terminal window) you'll be able to observe the frame sent by program **magic**:

```
[internal 11] $ ./magic enp1s0 e0:d5:5e:dd:ec:67
```

As you suspect, `e0:d5:5e:dd:ec:67` is the MAC address used by NIC `enp1s0.`

Capture the message printed out by **magic**

g. Wait about 3 minutes for the PC to boot-up and, send it ping to check whether or not it has fully booted-up along with the full TCP/IP protocol stack. Capture the ping responses.

In case you receive no response from the host, try resending the magic packet to it and, again, waiting until the PC boots up.

In case that you have received "ping" responses (ICMP Echo Reply) from the intended host you can remotely login in that computer by using this user/password combination:

```
User = administrator

Password = 19xxdpq16
```

Exercise due care for the PC that just booted up since it is shared among all the students enrolled in this course and others. Whenever you finish your work, log out but **don't shutdown the *shared* PC.** Capture the result of executing ifconfig and highlight the host NIC's MAC address.

h. Switch to the other of your remote ssh sessions (The one on which tcpdump is running) and make a screenshot of the captured magic packet. If all ran well, you can kill **tcpdump** by composing the ctrl-c key combination.

i. Visually observe the structure of the magic packet; try to identify its structure. Which of its fields contains the destination MAC? The destination MAC is comprised of the first 6 bytes from the frame. Include the destination MAC here:

j. Next, after the destination MAC, comes the source MAC. Include it, here, also:

k. The next field in the Ethernet frame is the multiplexing key (Technically known as Ethertype). The Ethertype is comprised of 2 bytes. Include here.

l. The next bytes in the frame, those that come after the Ethertype, all of them comprise the payload. Observe it and try to explain if you see some regularity.

m. After the DMAC (Destination MAC), comes the SMAC (Source MAC) and the Ethertype (The multiplexing key used in Ethernet), finally comes 12 hexdecimal f constants (0xffffffffffff), can you explain its purpose?

n. How many times is the MAC address of the host that is intended to be powered-up repeated?

o. If you wish to repeat the experiment you can execute tcpdump with the options that *normally* print out richer information about the captured traffic, as in the fowling tcpdump command line which will display the ethernet headers along with the payload, all in hexadecimal base:


```
$ tcpdump -vvv -XX -e -i enp1s0 ether proto 0x0842
```

Note that you will have to execute **tcpdump** with **sudo** or by first switching to superuser with the **su** command.

