v 1.4

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

CN Practice on PF_PACKET Socket Programming

Study Guide

This study guide outline is not to be included in your LabBook writeup

1. A valuable resource as you undertake the practice exercises is the Practices and Questionnaires that we did the past academic year:

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

2. 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 and read/write onto the raw socket successfully. I have started a service in paloalto.unileon.es that grants the CAP_NET_RAW capability to your programs such that they successfully open a raw socket even if the running user is not the root user. Assuming that your user name is student0 and that your executable file name is magic, the following command will achieve granting magic the CAP_NET_RAW capability:

   ```
   $ 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:

   ```
   $ setcap -v 'CAP_NET_RAW=epi' magic
   magic: OK
   ```

Sundry technical information

MAC addresses of relevant Lab B6's host interfaces

`http://paloalto.unileon.es/cn/Q/mac-ip.txt`

Source code

http://paloalto.unileon.es/cn/labs/pf_packet_send.c
http://paloalto.unileon.es/cn/labs/pf_packet_receive.c

Exercises for practice

This practice aims to illustrate sending and receiving with Linux PF_PACKET sockets. The first program referenced above illustrates sending and the other illustrates receiving. The following sequence of exercises ultimately builds a pair of programs one of which sends a frame and the other one receives it.

1. Checking the send program

a. Create session #1 in paloalto.unileon.es, that is a remote session

   Log in with the student user and password 19xxdpq16

b. Create a directory for your personal work, for example directory name might be 123Z; then, change directory to it:

```
$ mkdir 123Z
$ cd 123Z
```

c. Download the base program referenced in the source code section above:

   Run wget for the download from the link above

d. Edit the program so that it uses an appropriate multiplexing key (Etehrtype). You'll be assigned a unique Ethertype for this practice.

   Choose an editor such as vi, nano or gedit in Linux and enter the ethertype value assigned personally to you by replacing the numeric constant 0x07ff assigned to constant ETHERTYPE_EXPERIMENTAL. Search for a constant declaration like this and make the change:

```
#define ETHERTYPE_EXPERIMENTAL 0x07ff
```

e. Save and compile the program

```
$ gcc -o send pf_packet_send.c
```


f. Have the CAP_NET_RAW capability set on your program by sending its full path name to /home/administrator/fifo.cn

```
$ echo /home/student/123Z/send > /home/administrator/fifo.cn
```

g. Open an ssh session (Session 2) on the host that is to receive the frame sent by the sending program. Run tcpdump with options appropriate for the check we want to do now (Consult previous practices). This instance of tcpdump will wait until the frame is sent on the next step.

h. Run the program in session 1 and check on the remote computer (Session 2) that the frame sent is correctly received.

At this point, our program sends a frame to a given destination MAC and Ethertype and its reception has been checked by using tcpdump. Now, we move on to sending the frame once again and have it received with the use of a new program that we must evolve.

2. Checking the receive program

At this point we want to check the pair sending-receiving program. We assume a local session for the receive program (your lab host, Session 2) and a remote host session for the receive program (paloalto, Session 1, S1).

a. (On session S2) Download the receive program

b. (S2) Edit the receive program so that it is only delivered the traffic having an Ethertype equal to your personal Ethertype. As before, modify accordingly the value of symbolic constant `ETHERTYPE_EXPERIMENTAL`.

c. (S2) Upgrade the execution capabilities of your program by send its name to the cn capability fifo:

```
$ echo /home/student/receive > /home/administrator/fifo.cn
```

d. (S2) Execute program receive which will wait for a packet to be received.

e. (S1) Run the send program

f. (S1) Check that the receive program did receive the data successfully.

3. **The receive program needs several adaptations that we aim to cover on the** following sections:

    a. (S1) The receive program needs that you adapt the loop where data is printed out. For example, if the send program sends text, modify the receive program print loop to print text.

    b. (S1) Have the receive program print out the source MAC contained in the received frame

    c. (S1) Have the receive program print out the length of the received data

4. **Correct the receive program errors. Use tcpdump for checking send and** receive at the sender host and/or at the receiving host.

5. **Extend the send program so that it computes the even parity of 1's to the** received data (Only 1 parity bit for the whole structure). The parity bit will be encapsulated into one byte which will be sent appended to the data bytes.

6. **Extend the receive program so that it computes the even parity of 1's to the** received data (Only 1 parity bit for the whole structure). The parity bit will be encapsulated into one byte. Then, the computed parity bit and the received parity bit will be compared and the program will print out the result of this comparison.

