V.1.8 14th · March · 2022

# Practical Exercises in Computer Networks and Distributed Systems

## Network Interfaces and Protocol Stacks (WIP)



Internet end systems, or Internet hosts, connect to Internet by means of a Networ Interface Card. The PCs in our laboratories use the Linux operating system in one of its several flavors (Debian, Ubuntu, Fedora, etc). In this laboratory practice we set out to understand the basic commands that allow our system to communicate in Internet, monitor and manage that connection. You can check the commands and processes explained hereon by using your own laptop or one of the PCs in Lab B6.

## The network interface

At the beginning of this course on Computer Networks we study that networking is a systems discipline that defines networks by using a number of concepts organized into a hierarchy of layers. Each layer offers one or several services that can be used by calling the layer interface. You simply call the interface in order to make the service work in some useful manner. When some application in our computer needs to access the network, it does so by calling the *network interface.*

The operating system offers us a series of commands to properly bootstrap the network adapter, configure it, use it, monitor it, etc. Before we can issue a command to the NIC[1] we will have to find out its name, that is, the name given to it by the operating system in the bootstrap process, for more than one NIC may be present in the system. The kernel associates a name and a device driver with each NIC and the system administrator allocates one or more IP addresses to it according to the network design specifications. If you bring your laptop with you, it will retrieve its IP address wirelessly by using a broadcast-based protocol named DHCP, thereby granting you the basic conditions for network connectivity. Virtually, no intervention on your part is required as much as the system boot process is concerned. We will study this protocol in a few lessons; for the time being, we will proceed to explain the most basic network configuration command: ifconfig.

The OS-X and Linux command ifconfig reports the network parameters of one or more NICs, also, upon system boot up it will configure the different NICs according to the contents of a series of text files written by the administrator, for now, we aim to just check ifconfig reporting capabilities.

### Exercise 1. ifconfig

Login to your system and execute the following ifconfig command which will let you know its IP address and other significant network parameters. The example system is OS-X, in Linux you will have to substitute en0 (The network device name) for eth0 or maybe enp1s4, eno1, etc. If you provide no specific device name, ifconfig will list out all the device names known by the system (In most of the command lines we assume **$** as the *shell prompt*):

[1] The acronym NIC means Network Interface Card, which is comprised of an electronic circuitry along with some software installed on it

```
$ ifconfig en0
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu
1500
     options=2b<RXCSUM,TXCSUM,VLAN_HWTAGGING,TSO4>
     ether c8:2a:14:55:1d:93
     inet6 fe80::ca2a:14ff:fe55:1d93%en0 prefixlen 64 scopeid 0x4
     inet 192.168.99.99 netmask 0xffffff00 broadcast 192.168.99.255
     media: autoselect (1000baseT <full-duplex,flow-control>)
     status: active
```

ifconfig stands for "interface configuration", i.e. "network device interface configuration", you will find that command in all the main stream operating systems, including Windows[2]. Let's review the parameter list reported by the command highlighting the most important ones. In the present case, the kernel booted up and in the process it detected an administratively established configuration for our 1000BASE-T device, the point-to-point communication medium is a set of copper wires connected to the transmitting electronics. The medium is managed in such a way that it allows simultaneous bidirectional communication (full-duplex) and the computer and the network device to which it is connected support flow-control, that is, the regulated reception of information with no possibility of overflowing its limited-size input buffer. Other significant parameters are related to the internet identification of our computer system, that is, its IP address (Internetwork Protocol), its network mask or net mask that indicates which number of upper-weight bits of the IP address constitute the network number (IP addresses are hierarchical and contain three parts: the network number, the subnetwork number and the node number).

The broadcast address is a special network-address used for communicating with all the computers belonging to a specific network, in the case of our lab, the network will be of the Ethernet type and the broadcast address of this network is used by certain applications to offer and discover network-based services, for example, DHCPv4 is a broadcast-based protocol that serves the purpose of assigning IP addresses and other network-related parameters to end nodes in an unattended, automatic manner. The ether field represents the hardware address of our Ethernet NIC, this number is assigned by the NIC's manufacturer and is normally not changed, it allows our NIC to be identified univocally within our network, these addresses in the Ethernet technology realm are named hardware or MAC (Medium Access Control) addresses. These addresses are not to be confused with the IP addresses mentioned before: the MAC addresses guarantee that traffic directed to our system is properly delivered and received by the destination system, the IP addresses identify our system in an internetwork (IP, Internetwork Protocol), that is in Internet. As you already guessed, there is a mechanism that maps IP addresses to MAC addresses within any local network, this mechanism includes a protocol that we will study soon, its name is ARP (Address Resolution Protocol).

### Exercise 2. dmesg

Depending on machine-specific details, another command that comes in handy if you need to find out the network devices configured on a boot is dmesg, normally you will have to issue it via sudo, that is, by requesting to raise your user privilege level to that of root user

[2] In Windows the command to configure your NIC is ipconfig

```
$ sudo dmesg | grep ether
BCM5701Enet: Ethernet address 3c:07:54:52:ae:2a
AirPort_Brcm4331: Ethernet address 60:c5:47:23:9e:e5
AppleBCM5701Ethernet: a 1 extract_link_settings - EEE logic error
Ethernet [AppleBCM5701Ethernet]: Link up on en0, 10-Megabit, Half-duplex, No flow-
control, Debug [794d,0101,0de1,0300,0000,0000]
Ethernet [AppleBCM5701Ethernet]: Link down on en0
Ethernet [AppleBCM5701Ethernet]: Link up on en0, 1-Gigabit, Full-duplex, Symmetric
flow-control, Debug [796d,2301,0de1,0300,c5e1,3800]
AppleBCM5701Ethernet: 0 1 BCM5701Enet::replaceOrCopyPacket worked after N tries
```

When this machine booted, it eventually configured an ethernet NIC named AppleBMC5701Ethernet, the system device name is en0 and it follows the Gigabit standard in full-duplex mode and with flow-control.

### Exercise 3. Listing all the NICs installed

Observe below the output obtained at a Linux system that has three NICs, *i.e.,* three Network Adapters and respond to these questions:

1. How many NICs are installed in the host (Internet host is a computer system that has the Internet protocols installed)?

2. Is lo0 a NIC also?

3. List the IP addresses applied to the interfaces

4. According to your observations, can a host have more than one IP address?

```
$ ifconfig

enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.99  netmask 255.255.255.0  broadcast 192.168.2.255
        inet6 fe80::e2d5:5eff:fe27:2f72  prefixlen 64  scopeid 0x20<link>
        ether e0:d5:5e:27:2f:72  txqueuelen 1000  (Ethernet)
        RX packets 12664002  bytes 8520448423 (7.9 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8751984  bytes 8692363357 (8.0 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.99  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::8616:f9ff:fe03:c504  prefixlen 64  scopeid 0x20<link>
        ether 84:16:f9:03:c5:04  txqueuelen 1000  (Ethernet)
        RX packets 3189873  bytes 249203072 (237.6 MiB)
        RX errors 0  dropped 105601  overruns 0  frame 0
        TX packets 4888951  bytes 6790010291 (6.3 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
enp4s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.100.100  netmask 255.255.255.0  broadcast 192.168.100.255
        inet6 fe80::93eb:1f8b:5d1b:a225  prefixlen 64  scopeid 0x20<link>
        ether 18:d6:c7:02:8b:59  txqueuelen 1000  (Ethernet)
        RX packets 4621272  bytes 405429294 (386.6 MiB)
        RX errors 0  dropped 19240  overruns 0  frame 0
        TX packets 17830  bytes 1474441 (1.4 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo:     flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 5005022  bytes 10502237579 (9.7 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5005022  bytes 10502237579 (9.7 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Not all operating systems produce the same information about each installed NIC. Observe the output obtained at an OS-X system restricted to interface en0 (Ethernet 0):

**$ ifconfig en0**

```
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
 options=2b<RXCSUM,TXCSUM,VLAN_HWTAGGING,TSO4>
 ether 3c:07:54:52:ae:2a
 inet6 fe80::3e07:54ff:fe52:ae2a%en0 prefixlen 64 scopeid 0x4
 inet 192.168.2.106 netmask 0xffffff00 broadcast 192.168.2.255
 media: autoselect (1000baseT <full-duplex,flow-control>)
 status: active
```

1000BASE-T means that our NIC has successfully negotiated a fully functional Gigabit Ethernet over copper link (1000 Mbps, baseband signaling over cat 5e or cat6 structured cabling). If either the transmission medium itself or its physical surroundings are such that that speed can not be reliably attained, the NIC will fall down to a 100Mbps speed in a mode known as Fast Ethernet. As you can see, Ethernet technologies play an important role in today's campus networking and is the reference technology employed in the deployment of the networks of the University of Leon.

### Exercise 4. Basic Linux commands

The following commands usually offer a lot of help when working in Linux. Play with each and try to interpret the results.

**$ man ifconfig**

The man command uses to offer very valuable and reliable information, also, you can find a lot of information about ifconfig in internet. The ifconfig command is an example of a command that can be found in a number of systems, however, that does not mean they accept exactly the same option. Some options may or may not be available depending on the specific Linux/UNIX system considered. Using ifconfig for setting up the network interface entails root privileges, therefore, either you switch user to root via the su or sudo commands (Recall that the NIC device names of the specific system you are

working at will not be the same that appear in the examples below):

**$ su** or **$ sudo su**

Alternatively,
You can submit the ifconfig command as a sudo parameter, in which case, the super-user change will apply to the execution of the ifconfig command:

```
$ sudo ifconfig enp0s4 up
```

The latter command activates the interface enp0s4 if it were not activated at the time, or it will do nothing in case it was already activated. It's an example of ifconfig execution that requires super-user privileges since it attempts to modify a network interface configuration.

Lastly, if your shell can not find ifconfig, issue a whereis command by passing it the name of the command which file system location you wish to find out. The example right below asks the location of ifconfig and the system responds with **/sbin/ifconfig**. You can use the answer to compose the command's full path name.

```
$ whereis ifconfig
/sbin/ifconfig
```

Here, we proceed to execute the command:

```
$ /sbin/ifconfig
```

If you suspect that you'll be often using commands from the /sbin folder, maybe you should append it to your PATH environment variable in your $HOME/.profile file; add the following line to the file:

```
$ PATH=$PATH:/sbin
```

The following commands should be helpful in these practices, also:

Print the current directory:

```
$ pwd
/home/student
```

Change directory to folder practice-1 which resides in my current directory:

```
$ cd practice-1
```

Listing the files in the current directory

```
$ ls -l
-rwxr-xr-x  1 networks networks       8640 Oct 17 20:22 sizes
-rw-r--r--  1 networks networks        281 Oct 10 12:59 sizes.c
-rw-r--r--  1 networks networks       2378 Jan 11 14:58 sockoptions.c
drwxr-xr-x  4 root     root           4096 Apr 13  2018 sphinx
drwxr-xr-x  6 root     root           4096 Mar 24  2018 tutorial
```

Capture the standard output of the command ifconfig eno1 into a file

```
$ ifconfig enp4s0 > mydoc
```

Now, you can edit that file with an editor (gedit, for example) or a command line visual editor like vi:

```
$ vi mydoc
```

Add some text for separating the different sections of the documentation you're generating to ease editing later:

```
$ echo "------- Exercise 23 -------" >> mydoc
```

If you wish to *append* more text to file mydoc, you can do it with the shell append operator (Represented by the digraph >>); for example, I'm going to append the list of UDP sockets active in my system now to the foregoing file. The netstat command generates the listing and grep will filter the lines containing the string "udp"; the >> operator will append the foregoing output to the contents of file mydoc:

```
$ netstat –av | grep udp >> mydoc
```

Now, we check the file contents is what we expected:

```
$ cat mydoc
enp4s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.100.100  netmask 255.255.255.0  broadcast 192.168.100.255
        inet6 fe80::93eb:1f8b:5d1b:a225  prefixlen 64  scopeid 0x20<link>
        ether 18:d6:c7:02:8b:59  txqueuelen 1000  (Ethernet)
        RX packets 27376  bytes 2038864 (1.9 MiB)
        RX errors 0  dropped 1137  overruns 0  frame 0
        TX packets 356  bytes 28882 (28.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

------- Exercise 23 -------

udp        0      0 0.0.0.0:35429           0.0.0.0:*
udp        0      0 0.0.0.0:ipp             0.0.0.0:*
udp        0      0 0.0.0.0:mdns            0.0.0.0:*
udp        0      0 0.0.0.0:1900            0.0.0.0:*
udp        0      0 protocol:ntp            0.0.0.0:*
udp        0      0 protocol:ntp            0.0.0.0:*
udp        0      0 protocol:ntp            0.0.0.0:*
udp        0      0 localhost:ntp           0.0.0.0:*
udp        0      0 0.0.0.0:ntp             0.0.0.0:*
```

Installing new software under Debian or Ubuntu Linux entails using the following commands:

Update your distribution's package repositories:

```
$ sudo apt-get update
```

Update your distribution's packages:

```
$ sudo apt-get upgrade
```

Install a software product; in this case we are installing the ssh (Secure Shell server) by including the package name (Typically, these names can be found by searching the Internet):

```
$ sudo apt-get install ssh
```

Install the network sniffer **tcpdump,** which we use in this course for simpler network monitoring and observation needs:

```
$ sudo apt-get install tcpdump
```

## Exercise 5. Ifconfig options

Review the main command-line options of the ifconfig command in your system.

## Exercise 6. Ifconfig for setting NIC parameters

0. Listing the network devices that are available in a Linux host.

    a. Install the lshw (List Hardware) command if necessary (The installation procedure is explained above, under Exercise 4). Redirect the full listing to a file and then, edit it by deleting the information related to devices other than those of type *Network.*

    b. Of the information printed out for *network* devices, the logical name is the string that must be used for configuring the device after system boot.

    c. Check the network devices that are enabled after system boot. Execute ifconfig. Each device logical name represents one network device; its configuration follows the logical name and includes:

        - The hardware status: UP, BROADCAST, RUNNING, MULTICAST
        - Whether an IP stack is enabled on top of the device. In that case, its IP address will be printed alongside its IP Network Mask and the IP Broadcast Address.
        - The MAC address
        - Device transmission and reception statistics

    d. If some *physical device* is present that is not logically enabled, enable it and enter it into file /etc/network/interfaces . We'll assume that the device's logical name is enp1s0:
        auto enp1s0
        iface enp1s0 inet manual

    e. Reboot the system so that it loads the new configuration. After system boot, you'll be allowed to set the selected device (enp1s0) configuration manually, from the shell.

1. Can the ifconfig command be used to configure the NIC or just to report its network parameters? Compose the ifconfig command line that sets the network configuration of your system's main Ethernet interface (Typically en0 or eth0), the interface parameters to be used in this example case are the following:

    IP address 192.168.86.222; netmask 255.255.255.0 up

2. Once you submit the command, if it does not report any error, reboot your system and observe the boot process to identify the moment at which the system scans and configures each of the NICs present in it.

3. Finally, log on as administrator and check the interface configuration by using ifconfig. Did the new configuration survive the reboot? If not, what administrative steps would you take to have the system automatically set the interface configuration that the administrator wishes?

4. Search the internet for the specific steps that *your Linux distribution* entails for making ifconfig configuration changes permanent. (The distributions at use in our lab are Debian and Ubuntu).

5. Change permanently the IP address and netmask of your host by following these guidelines:

```
$ cd /etc/network
$ vim interfaces
auto enp3s0
iface …
```

**Exercise 7. Ifconfig for setting the NIC MTU (Maximum Transfer Unit)**

1. Execute the ifconfig command and take note of the MTU of your main network interface.

2. What does MTU mean? Explain your answer.

3. Can the MTU be changed? If, so, change your MTU to 1000, then reboot your system and see whether the change survived the reboot.

So far, we have reviewed the ifconfig command which is used to setup the network interfaces of the system, now, we are to move to explaining the use of several utilities that help us establish whether our system has internet connectivity with other hosts in the Internet.

## The ping command

The UNIX ping command is used to see whether an Internet host has IP connectivity to another Internet host. The ping name stands for *Packet Internet Gopher* and is an old-time utility in Internet hosts; it functions by using the ICMP (Internet Control Management Protocol) protocol which is an essential IP control-plane protocol. Depending on the options included by the user, ping can send and receive different ICMP protocol messages. The simplest form of the ping command sends the ICMP ECHO message to the specified destination IP address and the destination (receiving) IP module[3] will react by sending back the same ECHO message. The sending ICMP can measure the Rtt (Round Trip Time) upon receipt of the ICMP ECHO back message. Below is an example of ping executed in OS-X: (If you want to stop the ping command, compose the key combination ctrl-C)

```
$ ping www.telefonica.net
PING www.telefonica.net (213.4.130.95): 56 data bytes
64 bytes from 213.4.130.95: icmp_seq=0 ttl=120 time=43.752 ms
64 bytes from 213.4.130.95: icmp_seq=1 ttl=120 time=41.791 ms
64 bytes from 213.4.130.95: icmp_seq=2 ttl=120 time=43.660 ms
64 bytes from 213.4.130.95: icmp_seq=3 ttl=120 time=43.293 ms
64 bytes from 213.4.130.95: icmp_seq=4 ttl=120 time=43.070 ms
64 bytes from 213.4.130.95: icmp_seq=5 ttl=120 time=42.248 ms
```

[3] Module refers to the software organization that implements the TCP/IP protocols in the Linux operating system, including ICMP

```
64 bytes from 213.4.130.95: icmp_seq=6 ttl=120 time=43.548 ms
64 bytes from 213.4.130.95: icmp_seq=7 ttl=120 time=43.364 ms
64 bytes from 213.4.130.95: icmp_seq=8 ttl=120 time=43.014 ms
64 bytes from 213.4.130.95: icmp_seq=9 ttl=120 time=42.474 ms
64 bytes from 213.4.130.95: icmp_seq=10 ttl=120 time=43.837 ms
^C
--- www.telefonica.net ping statistics ---
11 packets transmitted, 11 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 41.791/43.096/43.837/0.635 ms
```

Depending on the specific platform, the output may differ but essentially ping tells whether there exists IP connectivity between two Internet hosts. As we will review, the Internet architecture has a single, central protocol in the network layer, its name is IP (Internetwork Protocol), there is no other network protocol in this architecture. From an implementation standpoint, the IP module belonging to the TCP/IP stack of a typical Linux/Unix/Window must contain other two modules in turn: the ARP and the ICMP. As we mentioned above, the ARP module maps internetwork addresses (IP addresses) to level-2 MAC addresses (The hardware address of an Ethernet NIC, for example), the ICMP protocol belongs to the control plane, that is, it is not involved in effective data transfers but cares for those transfers by coordinating the network equipment in ways that we will take up later. Together, the three protocol modules: IP, ARP and ICMP constitute what is normally known as the *IP module.* The technical specifications corresponding to the protocols that govern the Internet are called RFCs (Request For Comments) by the IETF (Internet Engineering Task Force), for instance, the RFC that documents the ICMP protocol is RFC 792. It's common that some RFCs affect other future RFCs, sometimes to the point of being superseded by them. At the beginning of an RFC, a list of affected RFCs appears alongside its past history. To finish this brief foray into the realm of Internet protocols we must recall that ICMP messages are encapsulated in IP datagrams; we will delve into these important concepts in chapter 1 and further.

### Exercise 8. ICMP protocol messages

1. Which ICMP code defines an ECHO REQUEST? And, which ICMP code defines an ECHO REPLY?

2. Search the Internet for RFC 792; these documents are just a simple Internet search away and are available in several formats for improved viewing, searching, etc. Skim that RFC 792 and search for the message type and code that defines an ICMP Echo and an ICMP Echo Reply.

3. Speculate about different scenarios where the response to an ICMP echo request is not received? In that case, what is the behavior of the originating node? See the following example and try to reproduce it in your computer, maybe by trying to contact other hosts that might have not been powered up yet (Compose IP addresses that that belong to our lab's network whose network number is 192.168.1.0 and whose netmask is 255.255.255.0, for example: 192.168.1.120, 192.168.1.121, etc.).

```
$ ping 192.168.99.102
PING 192.168.99.102 (192.168.99.102): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
ping: sendto: No route to host
Request timeout for icmp_seq 4
ping: sendto: Host is down
Request timeout for icmp_seq 5
```

```
ping: sendto: Host is down
```

4. The procedure to have a host not respond to ICMP echo requests consists of setting kernel parameter to the right integer value (1) as is illustrated below. Use ping to test connectivity with a host that has disabled the echo responses and check whether other Internet services such as the Web keep functioning.

```
# List the kernel parameters that contain echo_ignore in their name
$ sysctl -a | grep echo_ignore
net.ipv4.icmp_echo_ignore_all = 0
net.ipv4.icmp_echo_ignore_broadcasts = 1

# Disable ICMP echo responses at this host
$ sudo sysctl –w net.ipv4.icmp_echo_ignore_all=1

# Enable ICMP echo responses at this host
$ sudo sysctl –w net.ipv4.icmp_echo_ignore_all=0
```

**Exercise 9. Other tasks with the ping command**

1. Find your system's current IP address with the ifconfig command, and send ping to it. Do you you receive echo response? If that is true, observe the RTTs (Round-trip times) and check that those RTTs are substantially smaller than those received when pinging www.telefonica.net, for instance or www.princeton.edu or www.cisco.com. Propose an explanation to those small RTTs.

2. In the preceding exercise you contacted your own IP in order to establish whether connectivity from your computer to itself were possible. Often a client application wants to connect with a server application that is running on the same system as the client. What would happen if effectively the network to which your system is connected were down and yet you needed to have client and server communicate, exchange protocol messages? Testing a client and a server within the same Internet host is possible even in the case your network is down, to that end, it is mandatory that the IP host have a special IP address (127.0.0.1, or host name localhost) configured to a *loopback network device*. This device plumbs the messages outgoing from the IP layer back onto itself, thereby making a physical network connection unnecessary. See the following example and execute the same test it in your system:

```
$ ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.066 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.139 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.181 ms
^C
--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.066/0.129/0.181/0.048 ms

$ ping localhost
PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.050 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.179 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.179 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.179 ms
```

```
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.174 ms
^C
--- localhost ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.050/0.152/0.179/0.051 ms
```

**Exercise 10. Other options of the ping command**

Explore other ping options (Execute man ping) and flags in your specific operating system –these options and flags are not the same across the sundry of mainstream operating systems of today. Tell us which ones seem useful.

## Checking the state of your system's TCP/IP stack: netstat

**netstat** is a versatile and powerful command that has a lot of options for checking the state of the TCP/IP stack of your system; the stack is made up of all the network software modules that take responsibility for the TCP/IP protocols and services in your system, the word stack represents the fact that these software modules are organized in a hierarchical manner that might remind us of a stack of modules.

Your system's networking software is made up not only of your NIC interfaces, obviously other essential pieces are built upon pure software as is the case with the IP module (Recall this module comprises modules for IP/ARP/ICMP). If we look up the hierarchy from the physical level of your Gigabit Ethernet (GBE) adapter we find the signaling electronics that belong to the integrated circuits that make up your GBE adapter, then, at level 2 we find two sublayers: the MAC and LLC, the former providing the correct *shared medium* access and the latter builds the frames with the IEEE 802 standardized structure. Then upwards in the hierarchy we find the IP module made in pure software and last by now, at level 4 we find two transport protocol modules, the powerhouse TCP and the more humble UDP and, last in this illustration, at level 5 (session) we find the famous Sockets that ultimately connect applications in a an end-to-end fashion over the internet: yes, the internet connects clients to servers and the clients are implemented by application threads running on an average operating system like Windows and the server, likewise, runs on threads of an average operating system like Unix. If we want to take such a overall-hierarchy look we use the netstat command., see the following glimpse at the output of the netstat command in a BSD-Unix-based operating system (Apple's MAc OS-X):

**Exercise 11. Watching the connections active in my host at this time**

Start your web browser and compose the following URL (Universal Resource Locator, a standardized form of pointer to web resources): http://examples.oreilly.com/9780596007218/. When the web page shows up, click on the zip file's link displayed, the associated file contains the Java code corresponding to a nice book on Java Network programming from O'Reilly, now, while the file is being downloaded type the following command on your terminal, this command will let you see the end-to-end TCP connection that binds your system to O'Reilly's, you can see the result from my own system here:

```
paloalto:~ Chema$ netstat  -t | grep oreilly
tcp4       0      0  192.168.99.99.53921    ftp.oreilly.com.http   ESTABLISHED
tcp4       0      0  192.168.99.99.53920    ftp.oreilly.com.http   ESTABLISHED
```

If you want to see the full listing of TCP connections to/from your system and their associated state, suppress the stdout filtering grep from the command line. In the case above, the web browser established two different connections from my system to O'Reilly, one on TCP port 53921 and the other 53920, the technical details that justify the two TCP connections will be studied in our brief exposure to the application level in chapter 9. Briefly explained, the output from the command manifest the two TCP connections, their current state (ESTABLISHED)at the time the command is executed, the originating IP address, which in both cases is 192.168.99.99 and the destination IP address which in this case is represented by its DNS (Domain Name System) equivalent "ftp.oreilly.com". The originating TCP ports in 192.168.99.99 are 53921 and 53920 respectively, as to the destination end of both connections, you can see that the port is the same (http) which number is 80, you can check that http as a port name corresponds to TCP port number 80 by grepping against your local port names database (/etc/services):

**Exercise 12. Seeing the contents of the services file in your Linux system**

Check whether your /etc/services file contains the name for TCP port number 80:

```
paloalto:~ Chema$ egrep "80/" /etc/services
http            80/udp      www www-http # World Wide Web HTTP
http            80/tcp      www www-http # World Wide Web HTTP
```

TCP port number 80 corresponds to the http service, i.e., access to the World Wide Web with its protocol, http (Hyper Text Transfer Protocol).

**Exercise 13. The options of the netstat command**

Consult the options available to the netstat command in your operating system. Check whether you have available some graphical front end to netstat, for example, in MAC OS-X, under applications/utilities you have such a GUI-based utility named "Network Utility":


Figure 1. A network monitoring utility from the OS-X system

GUI-based utilities like this come in handy when one does not want to memorize more and more commands and options, this is the result of executing it:

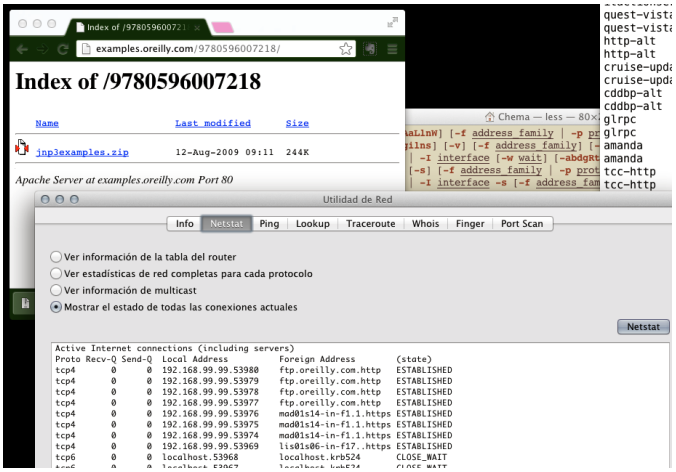

Figure 2. Output from netstat on the GUI network utility of the OS-X system

I am sure you have observed that there are other network utilities available, we will introduce some of them in this and upcoming experiments.

## Tracing the path to an end system (An Internet host): the traceroute command

The ICMP protocol messages can be used for purposes other than IP connectivity, for example, to trace the network pathway from an originating system to a destination system. The use of ICMP for this purpose is somewhat contrived and involves the use of a field of the IP packet named TTL or *Time To Live* which was conceived for the purpose of discarding an IP packet after it enters a routing loop, habitually due to a configuration error.

When an IP router receives an IP packet, it decrements its TTL field (TTL = TTL - 1) so it will eventually reach the value 0 if the number of routers visited by the packet suffices. IP prescribes that an IP packet's TTL reaching 0 should be discarded altogether (The packet is dropped). TTL was conceived as an upper bound to the number of routers an IP packet should cross on its trip to its destination, thus, in case of a network design or network management error caused a network loop, packets would have a guarantee not to loop in there forever. A looping packet would waste a variety of network and compute resources, and consequently it has to be avoided at all costs. Clever use of the TTL field of an IP packet can be used for estimating the sequence of routers that make up the *current* path from a source host to a given destination host. The traceroute utility accomplished exactly that.

The traceroute utility aims to estimate the sequence of routers that comprise the network pathway to a destination host, at the specific time the test is being carried out (Network pathways can change at any time, due to the varying degrees of Internet congestion and the availability of links. The traceroute utility, at a host, sends a probe IP packet containing a null-payload UDP message from UDP port 33434 (By default) with TTL=1. This IP packet, upon being received by the first IP router on the pathway that leads to some destination host chosen by the user, will get TTL=0 and the router will react by sending back an ICMP *Time Exceeded Message (TEM)* to the source host. The TEM message stands for the router if adequately interpreted by the source host after it receives it. Now, the source host may determine the identity of the first router since the received IP packet carries the IP address of the sending network interface of the identified router. Now, traceroute will send a probe IP packet containing TTL=2, which will elicit an ICMP *TEM* from the second router on the

pathway. This process of sending probe packets and expecting an ICMP *TEM* to each continues until it is the destination host which sends the ICMP TEM, at which time, traceroute will finish.

```
Brainstorm:~ chema$ traceroute www.princeton.edu
traceroute to www.princeton.edu (128.112.132.86), 64 hops max, 52 byte packets
 1 192.168.2.1 (192.168.2.1) 1.362 ms 1.030 ms 0.713 ms
 2 192.168.1.1 (192.168.1.1) 1.626 ms 1.726 ms 1.441 ms
 3 192.168.153.1 (192.168.153.1) 36.751 ms 36.398 ms 35.612 ms
 4 241.red-81-46-53.staticip.rima-tde.net (81.46.53.241) 35.946 ms 37.531 ms
37.617 ms
 5 so2-0-0-0-grtmadpe3.red.telefonica-wholesale.net (84.16.6.201) 42.791 ms 44.928
ms 43.290 ms
 6 xe2-1-0-0-grtpartv2.red.telefonica-wholesale.net (84.16.15.170) 66.932 ms
    xe7-1-2-0-grtparix1.red.telefonica-wholesale.net (213.140.36.134) 121.620 ms
63.980 ms
 7 xe-6-0-6-0-grtwaseq2.red.telefonica-wholesale.net (94.142.116.233) 155.683 ms
    xe7-1-2-0-grtwaseq2.red.telefonica-wholesale.net (94.142.126.81) 146.309 ms
    xe6-1-4-0-grtwaseq2.red.telefonica-wholesale.net (94.142.116.217) 146.060 ms
 8 xe5-0-1-0-grtwaseq5.red.telefonica-wholesale.net (213.140.36.61) 146.071 ms

Brainstorm:~ chema$ traceroute paloalto.unileon.es
traceroute to paloalto.unileon.es (193.146.101.46), 64 hops max, 52 byte packets
 1 192.168.2.1 (192.168.2.1) 3.036 ms 0.950 ms 0.689 ms
 2 192.168.1.1 (192.168.1.1) 1.426 ms 1.491 ms 1.324 ms
 3 192.168.153.1 (192.168.153.1) 36.583 ms 37.640 ms 37.073 ms
 4 130.red-81-46-37.staticip.rima-tde.net (81.46.37.130) 38.480 ms 36.203 ms
37.621 ms
 5 rediris-2.espanix.net (193.149.1.154) 49.484 ms 48.600 ms 47.699 ms
 6 ciemat.ae2.uva.rt1.cyl.red.rediris.es (130.206.245.10) 49.008 ms 48.970 ms
49.420 ms
 7 unileon-router.red.rediris.es (130.206.201.46) 56.238 ms 56.826 ms 56.040 ms
...
```

### Exercise 14. Establishing the network path to a host: the traceroute command

Repeat the traceroute to paloalto.unileon.es from the network location where you are, let the trace complete, contrast the results to the screen dump above.

### Exercise 15. Migrating your wired network connection to a WIFI AP

Move your network access point to the wireless access point and repeat the traceroute of exercise 1 and try to make sense of the results, how they change when you simply move from a wired concentrator to a Wi-FI AP, all within the campus network of the University.

### Exercise 16.1 Traceroute to a network location outside of Red Iris

Do a traceroute to a network location that does not belong to Red Iris, for instance www.hp.com, can you tell the DNS name or the IP address of the first routing interface that does not belong to Unileon?

Some of the intervening network nodes will block the outbound ICMP traffic for security reasons, that is the reason why you will see some routers represented by asterisks (* * *).

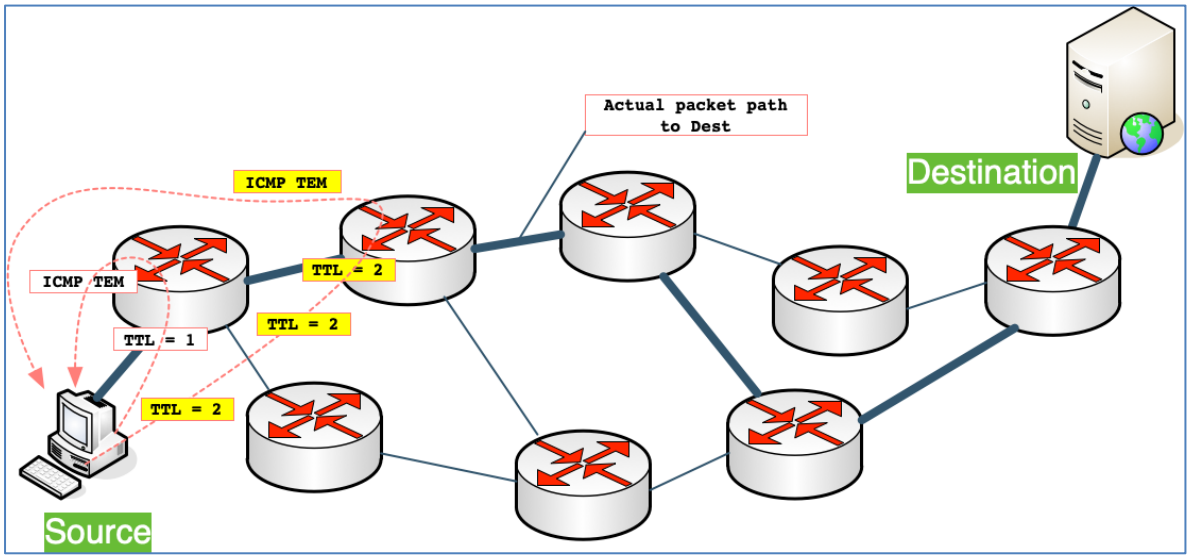


Figure 2.1. Source host sending two probe packets having TTL=1 and TTL=2 and receiving the respective TEM messages

**Exercise 16.2. The functioning of traceroute is based on the UDP and ICMP protocols**

Run tcpdump or Wireshark and develop the protocol stack resulting from a traceroute utility sending a probe IP packet.

This question is solved right below so you can follow a model in solving the ensuing two questions. Notice that the protocols in the protocol stack (Figure 3, right) are ordered into the same top-down order used in the architecture (4 through 1), however, the graphical representation displayed by Wireshark (Figure 3, left) is ordered in exactly the reverse order (The protocol appearing on the top is the physical Ethernet (Frame). Take this fact into account when deducing protocol stacks from Wireshark. In addition, observe that the Subnetwork layer as represented by Wireshark is comprised of the two uppermost lines: one for the physical layer and the other one for the datalink layer, this time, using the OSI architecture.

For technical precision, we should note that the traceroute utility doesn't call the UDP service interface (Datagram sockets); instead, traceroute manufactures each UDP datagram it sends and has each of them encapsulated into an IP packet. By convention, the port number included in the UDP datagram, must be a high one. The purpose of this artifact consists of allowing traceroute to control de TTL of each sent IP packet, which is the essence of traceroute functioning. In summary: traceroute creates a PF_INET/SOCK_RAW/protocol=1 raw socket and sets the raw socket's option IP_HDRINCL so that traceroute can set the TTL of each sent IP packet. traceroute at the source host collects the responses received from each identified router by using another PF_INET/SOCK_RAW/protocol=1 raw socket. The latter socket doesn't set option IP_HDRINCL, since it is not used for sending.

The protocol stack in Fig.3 might result misleading at first sight. Note that the UDP box means only that traceroute is complying with the specifications for the UDP protocol, that is, it is implementing that protocol's peer-to-peer interface. However, traceroute is not invoking UDP's service interface, which means that traceroute doesn't create a datagram socket for sending the necessary datagrams to the destination host.

When the IP packet makes it to the end host, the latter will complain with an ICMP message of type ICMP_DEST_UNREACH and code ICMP_PORT_ UNREACH. Figure 25-6 shows an example.

```
$ traceroute www.bbva.es (Terminal 1)
traceroute to www.bbva.es (104.83.210.61), 30 hops max, 60 byte packets
 1  DD-WRT-NetworkingLab (192.168.1.1)  0.414 ms  0.644 ms  0.750 ms
 2  n101001.unileon.es (193.146.101.1)  3.874 ms  3.849 ms  3.857 ms
 3  n111003.unileon.es (193.146.111.3)  3.879 ms  3.829 ms  3.836 ms
 4  185.179.107.225 (185.179.107.225)  5.087 ms  5.149 ms  5.228 ms
 5  REDCAYLE.ETHTRUNK0-85.ciemat.rt2.mad.red.rediris.es (130.206.201.121)  8.074 ms  9.333 ms  9.399 ms
 6  ciemat.rt2.ae1-0.telmad.rt4.mad.red.rediris.es (130.206.245.2)  24.502 ms  19.537 ms  20.246 ms
 7  rediris-ias-geant-gw.mar.fr.geant.net (83.97.88.129)  21.432 ms  21.163 ms  21.144 ms
 8  ae8.mx1.gen.ch.geant.net (62.40.98.73)  48.979 ms  49.148 ms  49.046 ms
 9  ae2.mx1.mad.es.geant.net (62.40.98.66)  46.995 ms  47.078 ms  47.303 ms^C


$ tcpdump -i eno1 -n -vvv -S  host 104.83.210.61 and udp or icmp[0]=11 (Terminal 2)

tcpdump: listening on eno1, link-type EN10MB (Ethernet), capture size 262144 bytes
13:32:00.488112 IP (tos 0x0, ttl 1, id 36858, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.45325 > 104.83.210.61.33434: [bad udp cksum 0xfcca -> 0xda5f!] UDP, length 32
13:32:00.488167 IP (tos 0x0, ttl 1, id 36859, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.40056 > 104.83.210.61.33435: [bad udp cksum 0xfcca -> 0xeef3!] UDP, length 32
13:32:00.488207 IP (tos 0x0, ttl 1, id 36860, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.43156 > 104.83.210.61.33436: [bad udp cksum 0xfcca -> 0xe2d6!] UDP, length 32
13:32:00.488245 IP (tos 0x0, ttl 2, id 36861, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.48830 > 104.83.210.61.33437: [bad udp cksum 0xfcca -> 0xccab!] UDP, length 32
13:32:00.488284 IP (tos 0x0, ttl 2, id 36862, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.44845 > 104.83.210.61.33438: [bad udp cksum 0xfcca -> 0xdc3b!] UDP, length 32
13:32:00.488321 IP (tos 0x0, ttl 2, id 36863, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.57335 > 104.83.210.61.33439: [bad udp cksum 0xfcca -> 0xab70!] UDP, length 32
13:32:00.488358 IP (tos 0x0, ttl 3, id 36864, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.35183 > 104.83.210.61.33440: [bad udp cksum 0xfcca -> 0x01f8!] UDP, length 32
13:32:00.488420 IP (tos 0x0, ttl 3, id 36865, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.39151 > 104.83.210.61.33441: [bad udp cksum 0xfcca -> 0xf276!] UDP, length 32
13:32:00.488459 IP (tos 0x0, ttl 3, id 36866, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.44732 > 104.83.210.61.33442: [bad udp cksum 0xfcca -> 0xdca8!] UDP, length 32
13:32:00.488504 IP (tos 0x0, ttl 4, id 36867, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.57690 > 104.83.210.61.33443: [bad udp cksum 0xfcca -> 0xaa09!] UDP, length 32
13:32:00.488512 IP (tos 0xc0, ttl 64, id 10315, offset 0, flags [none], proto ICMP (1), length 88)
    192.168.1.1 > 192.168.1.88: ICMP time exceeded in-transit, length 68
      IP (tos 0x0, ttl 1, id 36858, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.45325 > 104.83.210.61.33434: [udp sum ok] UDP, length 32
13:32:00.488542 IP (tos 0x0, ttl 4, id 36868, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.38623 > 104.83.210.61.33444: [bad udp cksum 0xfcca -> 0xf483!] UDP, length 32
13:32:00.488588 IP (tos 0x0, ttl 4, id 36869, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.55229 > 104.83.210.61.33445: [bad udp cksum 0xfcca -> 0xb3a4!] UDP, length 32
13:32:00.488626 IP (tos 0x0, ttl 5, id 36870, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.50747 > 104.83.210.61.33446: [bad udp cksum 0xfcca -> 0xc525!] UDP, length 32
13:32:00.488664 IP (tos 0x0, ttl 5, id 36871, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.55561 > 104.83.210.61.33447: [bad udp cksum 0xfcca -> 0xb256!] UDP, length 32
13:32:00.488707 IP (tos 0x0, ttl 5, id 36872, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.56919 > 104.83.210.61.33448: [bad udp cksum 0xfcca -> 0xad07!] UDP, length 32
13:32:00.488729 IP (tos 0x0, ttl 6, id 36873, offset 0, flags [none], proto UDP (17), length 60)
    192.168.1.88.42173 > 104.83.210.61.33449: [bad udp cksum 0xfcca -> 0xe6a0!] UDP, length 32
13:32:00.488803 IP (tos 0xc0, ttl 64, id 10316, offset 0, flags [none], proto ICMP (1), length 88)
    192.168.1.1 > 192.168.1.88: ICMP time exceeded in-transit, length 68
```

Figure 3.1. tcpdump trace of traceroute to host `104.83.210.61`

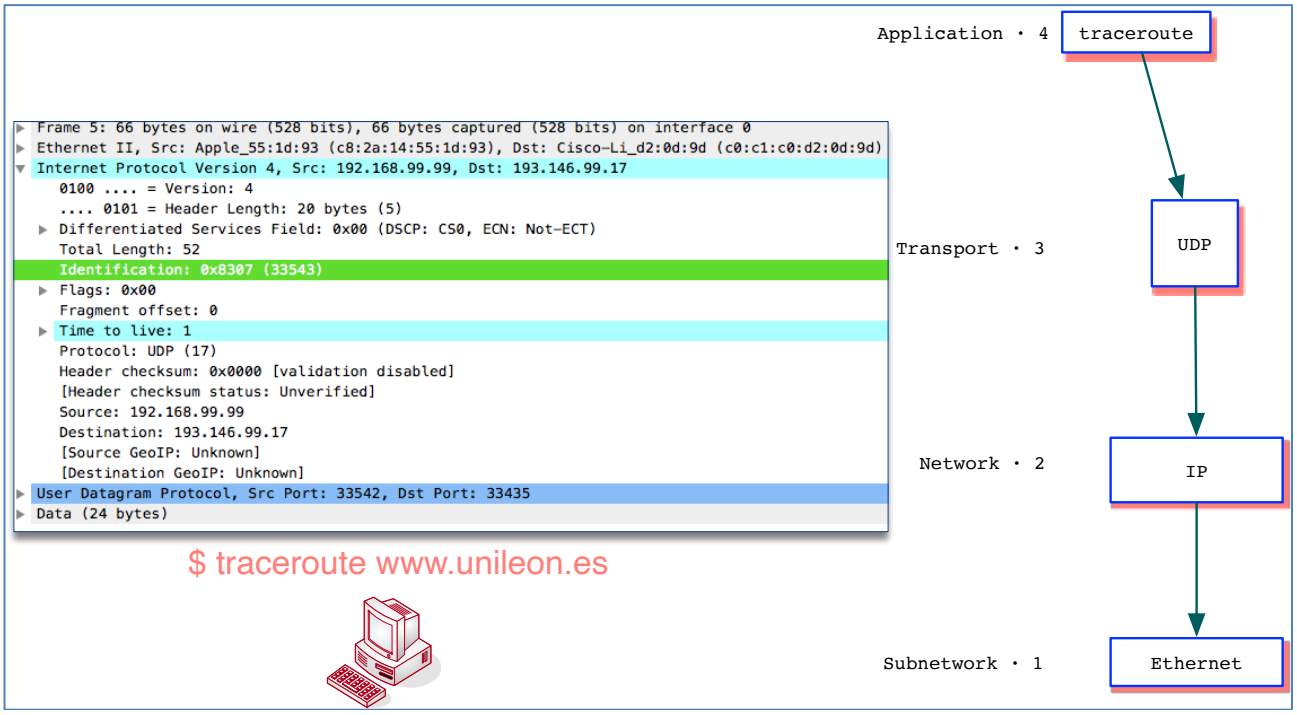


Figure 3.1. The protocol stack resulting from a traceroute probe packet (Right). The trace was obtained with Wireshark (Left)

**Exercise 17. Traceroute sending further UDP packets with increasing TTL**

Shortly after the sending traceroute receives each ICMP TEM, it sends a new probe IP packet with a TTL value one higher than the one just sent (Actually, several UDP test packets can be quickly sent in succession, with no need to wait to receive any TEM response). Carefully check whether or not you faithfully observe the foregoing behavior in the Wireshark trace. Finally, compare it to other systems' traceroute implementations for if you observe some differences (Windows traceroute counterpart utility is named tracert).

Observation:

- **Traceroute utility and TTL**
  - From RFC 791, verbatim:
    *"Since every module that processes a datagram must decrease the TTL by **at least one** even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist.". Some routers send back ICMP TTL Exceeded Message when they receive a TTL=2, not 1. The traceroute utility must take this fact into account so that it properly orders the routers comprising the path to the destination. Some ICMP TTL Exceeded messages may be generated when the receiving router receives an IP packet having TTL = 2, not 1.*

- A Wireshark filter:

Probe

(udp.port>=33000 && udp.port<34000) || icmp.code==0 && !ssdp && !dns

TEM

traceroute output

**Exercise 18. The protocol stack of a traceroute response**

Develop the protocol stack resulting corresponding to any of the ICMP TEM responses. Each of these is sent from any IP router laying amidst the pathway to an Internet destination of your choosing (For example, www.uclouvain.be) after receiving a probe IP packet from the source host executing traceroute which TTL=1.

## Virtual terminal and file transfer utilities

One the oldest command utilities in the development of Internet is telnet which allows us to connect remotely to another system, log in and then, after the prompt symbol appears on the screen you will be able to submit commands to the remote system and transparently receive its responses as though you were directly connected with a cable, serial connection.

Today, telnet is not used very much because of its lack of security, all the character strings that make up the commands typed locally and sent over the internet to the host system are sent in the clear, that is, with no protection against eavesdropping. When we want to sent those strings with guarantees of privacy we encrypt them by using a symmetric-key that will allow the receiver to decrypt them and getting the original text sent. telnet is considered so insecure that network administrators will close the telnet TCP port by default thereby filtering any traffic to or from that port. Nevertheless, the functionality offered by telnet must be important: to be able to submit commands to a remote machine, then, what is the solution? There is a wide spectrum of solutions to this problem which involve the use of different forms of cryptography, one of the most famous consists in using a secure virtual terminal program named ssh or Secure Shell. There are open software versions of ssh for Linux and Unix. The putty program for Windows supports the ssh protocol.

**Exercise 19. Ssh to paloalto**

a. Does paloalto.unileon.es have its ssh port open?

   Checking whether or not a TCP port is open in Linux/Unix host can be done by executing the netstat command

and including a few options as in the following example:

```
$ netstat -avn --tcp
```

The ssh server process is running if a line like this appears in the listing resulting from the command execution:

```
tcp        0   0       0.0.0.0:22        0.0.0.0:*       LISTEN
```

Port number 22 is open since its *endpoint* is in the LISTEN state. It can create connections using any of its assigned IP addresses; this is represented by reserved IP address 0.0.0.0. The other field that contains 0.0.0.0:* means that the local host is authorized to request connections to hosts at any IP addresses in Internet and at any port numbers (Observe the asterisk after the colon).

Include the listing of current TCP connections at your Internet host.

b. Which port number is it?

Consult the ssh service number in the /etc/services text file by searching for string 'ssh'. You may wish to use a text editor or do the search by using the grep command like here:

```
$ grep '22/tcp' /etc/services
ssh             22/tcp                          # SSH Remote Login Protocol
xmpp-client     5222/tcp        jabber-client   # Jabber Client Connection
bpjava-msvc     13722/tcp                       # BP Java MSVC Protocol
```

The /etc/services file is used in Linux, UNIX. Inlcude the results obtained from the previous grep command execution:

**Exercise 20. Observing ssh connections**

The first time that you connect with a host with ssh, the remote ssh server send you back a string that reliably identifies that server. That string is known as a fingerprint and it serves for avoiding the man-in-the-middle atack, *i.e.,* in case you do know the remote host's fingerprint the first time you connect to it. In the following example, an ssh connection request is sent to paloalto.unileon.es at ssh default tcp port, 22. Observe that the user is not specifying any tcp port in the command line. Since this is the first time that this user sends an ssh connection request to paloalto.unileon.es, the server sends its fingerprint. The user responds by accepting the server's fingerprint; we assume that that user did know the server's fingerprint and therefore it responded positively to it, so ssh adds the server paloalto.unileon.es to its list of known hosts. Henceforth, future connection requests to that server will be verified by using the saved host fingerprint, thereby not requiring any further verification involving the user. Notice that the server will have been actually verified even though no verification was requested form the user.

```
Brainstorm:~ chema$ ssh chema@paloalto.unileon.es
The authenticity of host 'paloalto.unileon.es (193.146.101.46)'
can't be established.
RSA key fingerprint is
```

```
d2:23:8d:cb:af:65:7a:91:d3:b7:1c:4a:74:0d:c1:9c.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'paloalto.unileon.es,193.146.101.46'
(RSA) to the list of known hosts.
Password:
Password:
Last login: Thu Feb 28 11:18:42 2013 from 139.red-81-35-
94.dynamicip.rima-tde.net

paloalto:~ Chema$ who
Chema console Feb 20 13:22
Chema ttys000 Feb 27 11:02
Chema ttys001 Feb 27 11:02
Chema ttys002 Feb 28 11:22 (139.red-81-35-94.dynamicip.rima-tde.net)
```

Observe that when we have effectively logged in, the command prompt is not "Brainstorm" anymore but "paloalto". I typed the who command, you can see its output reporting four sessions, the first on the host graphical console, the other three on network consoles (ttys*) the last one corresponding to my session (ttys002) which manifests the public IP address that was granted by my ISP (Internet Service Provider).

Connect with paloalto.unileon.es at tcp port 50500. After successfully logging into it, obtain a listing of all the ssh connections active at the time. Try to identify the concrete connection that the server is using for communicating with your ssh client process. Observe the following example:

- User connects and logs into paloalto.unileon.es. netstat command prints out the connection information. The source port number searched with grep was discovered on the local, client ssh window (In white, below)

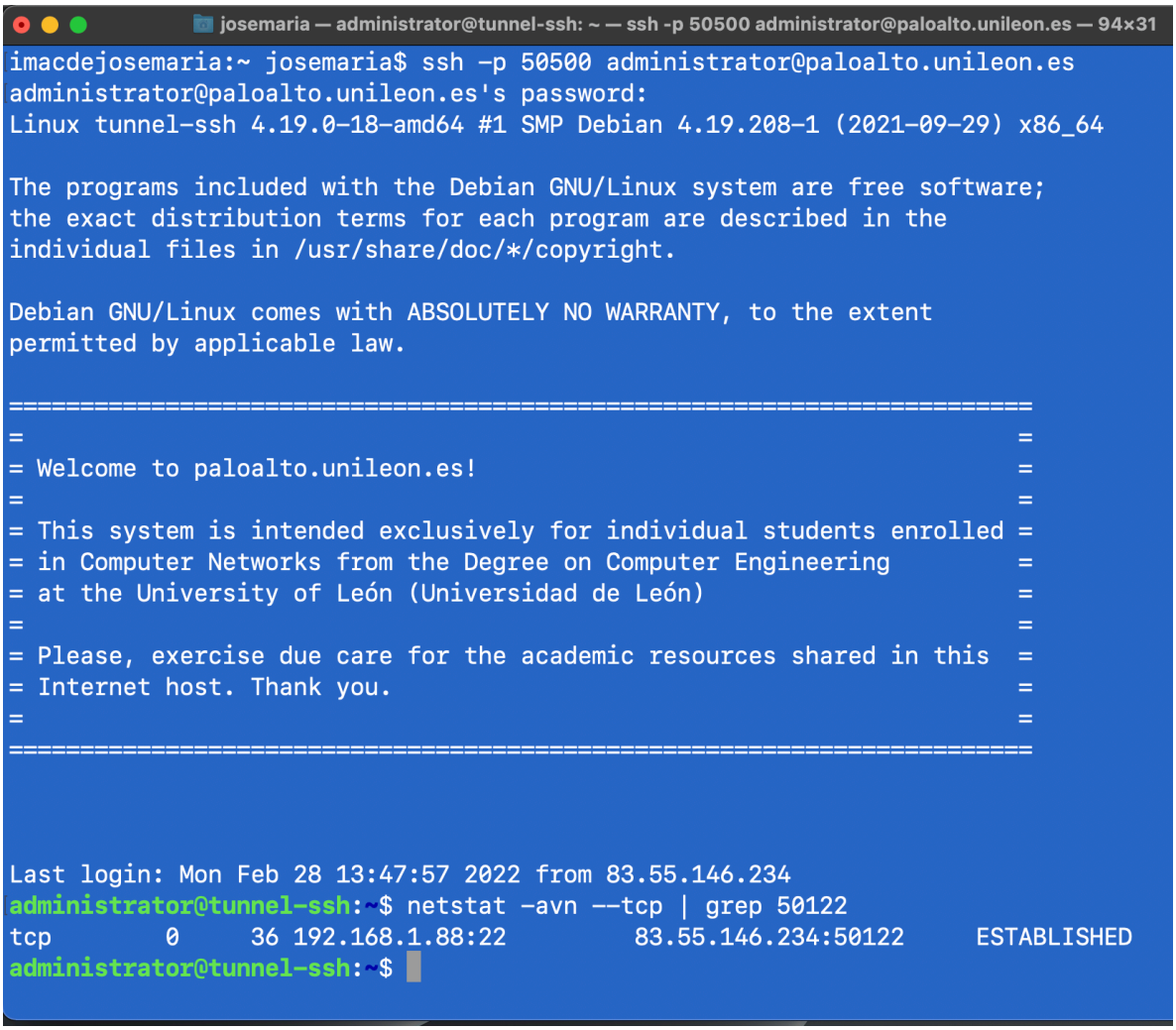


Figure 4. ssh connection information obtained at the ssh server side

- Listing of current ssh connections at the originating ssh client:


Figure 5. ssh connection information obtained at the ssh client side (The originating host)

Connect with your student account in paloalto.unileon.es at port 50500 and correlate the two netstat listings that you will obtain. Correlate both listings and try to discover the concrete TCP connection that is connecting your ssh client to the server.

**Exercise 21. man ssh**

Skim-read the ssh man page in your system, and get an overall idea of the information usually available in Unix/Linux man pages. Tell an ssh feature that you consider that is very powerful and which you had not heard yet from.

### Exercise 22. Checking that an ssh *server* process is running in a host

Connect with your student account in paloalto.unileon.es, log in and then obtain a listing of all the TCP servers available. Issue the usual netstat command ($ netstat -avn -- tcp) and observe the lines that include an asterisk as the port number and their state is 'LISTEN': those lines represent the available servers.

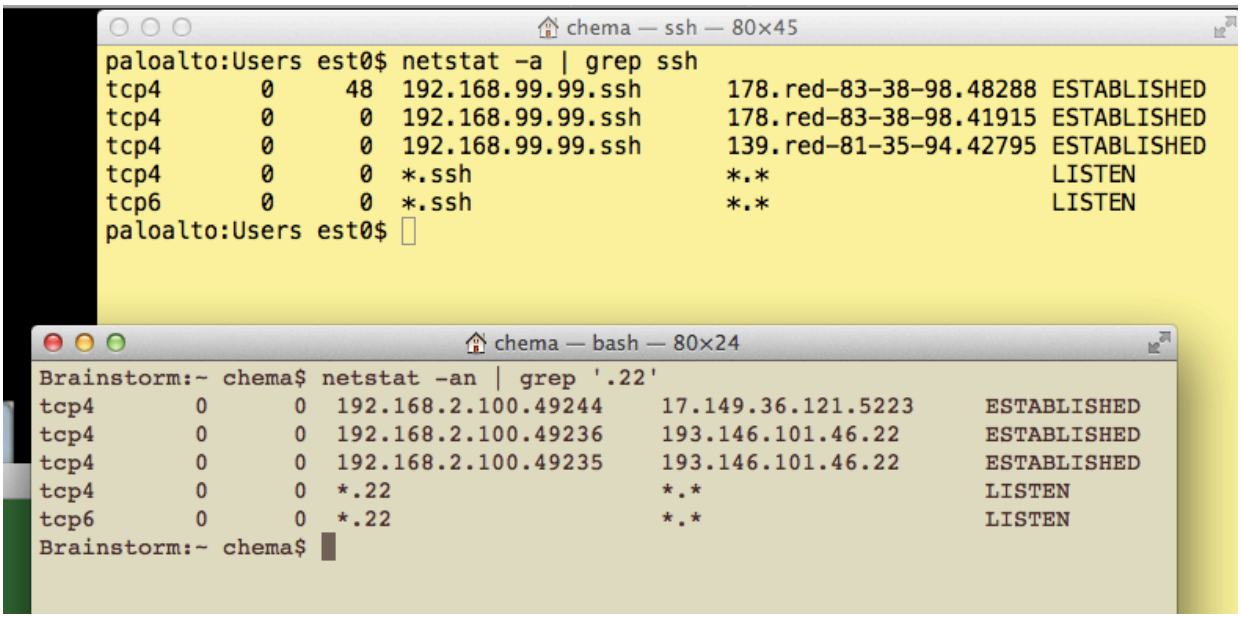


Figure 6. ssh servers represent ports as asterisks and their state is "LISTEN".

The output above reports active connections between a client and a server in state "ESTABLISHED"; lines that report state as "LISTEN", represent server processes.

19. Let's transfer a file named testfile.txt from our current directory to the home directory of user student at host paloalto.unileon.es at port 50500. Study the following example, then try to upload a file of your choosing to your student at that host. Check that the file was successfully uploaded (A small file suffices).

Another important utility in the initial development of Internet was the *file transfer utilities.* At the time there was ftp or File Transport Protocol which is still in use today, though it has the same lack of security problems that we mentioned above regarding telnet. Both utilities, telnet and ftp have the same origin in the US DoD (United States Department of Defense), for that reason, both utilities are known as DARPA services (Defense Advanced Research Projects Agency). Other similar utilities originated in a different set of software project led by the University of California, Berkeley, the names of those utilities were rlogin and rcp, these utilities are known as Berkeley services. Having the same security problems already mentioned, another secure file transfer utility was developed: scp, actually, ssh and scp both use the same underlying secure-transport protocol, though for different purposes.

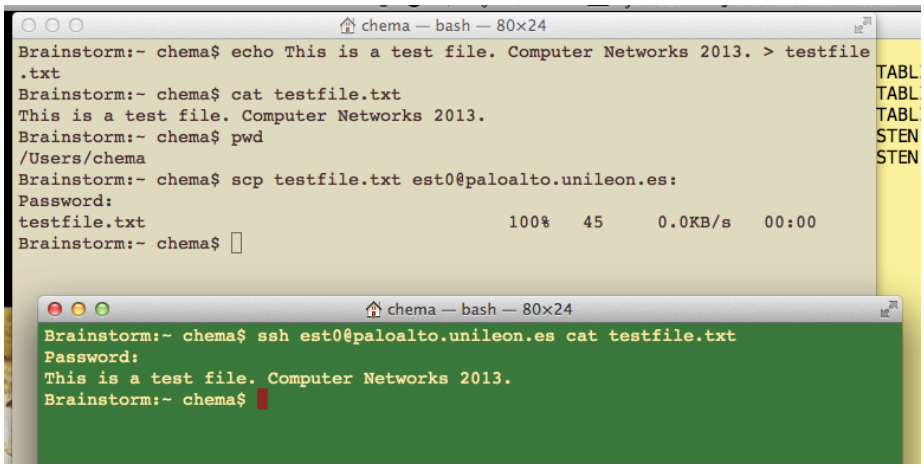


Figure 7. Downloading a file with scp

On the terminal with green background I requested the remote execution of the command "cat test file.txt" as a check that the file was actually uploaded, I did not establish a full session as in the above examples, I requested the remote execution of a single command. This capability of ssh comes in handy in many situations.

**Exercise 23. scp for uploading and downloading files**

Execute a secure file download by using scp of any file that you know that exists on the student account in paloalto.unileon.es at TCP port 50500, you will have to use the correct syntax by specifying the remote file and the the local resulting file name; please, skim the scp manual page if necessary ($ man scp).

## Check of Java Compiler and run-time

In this laboratory session we will want to check that your Java infrastructure is operational, to that end, we are going to practice with a Java Network API named NetworkInterface which allows us to determine the network interfaces installed in our computer and obtain some of its properties.

In the initial sections of this laboratory experiment we presented a series of commands related to your system network configuration, specifically ifconfig, but, ifconfig is a finished, operational command that offers us a lot of functionality, nevertheless in some situations one needs to have programmatic access to the network interfaces from Java. The following NifEnumerator.java obtains a list of the network interfaces installed in your system and prints them out on the console.

**Exercise 24. Installation and check of C and Java compilers**

Download the source code of the java example program illustrating the NetworkInterface API, use the following URL:

http://paloalto.unileon.es/cn/labs/NIfEnumerator.java

Observe the package name used if any and compile and run the program. Contrast the results obtained to those reported by the ifconfig program that we explained above.

**Exercise 25.** Look up the Java Docs for the Java NetworkInterface API, scan the methods and the most outstanding fields. Then access the Oracle Java Tutorial and scan the section devoted to NetworkInterface in it.

**Exercise 26.** Study the NifEnumerator.java source code, particularly those sections that might result least familiar to you, make sure you recall the Java Collections used.

## Check of C Compiler

In order to check your C compiler installation (GNU C compiler), carry out the following exercise, making sure that you understand each of its parts, since the involved facilities will often prove handy in other, upcoming CN Labs:

**Exercise 27.** Installing and checking the Gnu C compiler (gcc)

a. Install gcc, login to your Linux PC and issue the following command to start the installation:

```
$ sudo apt-get install gcc
(Provide your Linux user's password and respond Y to the ensuing prompts)
```

b. Download a short, test C program from our CN web server (paloalto), you may use the wget program from the command line or copy-paste the URL into your Firefox address box and, then, pressing the return key:

```
$ wget http://paloalto.unileon.es/cn/labs/check-gcc.c

$ ls –l check*
-rw-r--r--  1 Student  admin   171 22 feb 22:55 check-gcc.c

$ gcc –o check-gcc check-gcc.c
(If the compiler issued no errors, execute the program and observe the
message printed out on your terminal window)

$ ./check-gcc
Hello world, you compiled and ran this program successfully!
```

On next practice, we will write a simple networking program in C using one of the Sockets API.