Facultat Informàtica de Barcelona
FIB
Grau en Enginyeria Informàtica

Computer Networks Laboratory
Bachelor in Informatics Engineering
(XC-grau)
ENGLISH VERSION

Llorenç Cerdà Alabern, José M. Barceló Ordinas i David Carrera

September 2020

Index

Laboratory environment (“xarxes” image)

In this introductory chapter there’s a general description of the environment configuration that we will use for the laboratory sessions. When booting the PC, the “xarxes” image has to be selected. This image has been done using a Linux distribution of a reduced size called slitaz (http://www.slitaz.org).

1. Basic configuration

User and password: xc / xc

Superuser and password: root / root

The usual situation will be logging in as the “xc” user and in the terminal switching to root if needed.

The icons of the applications that we will usually use are in the bottom part of the desktop:

These are, from left to right: terminal, web browser, wireshark, calculator and editor.

To configure the PC via DHCP you need to execute the following command as a superuser. This is necessary to be able to access the pclabxc server to do the miniexams.

```
# udhcpc -i e0
```

2. PCs’ Interfaces

To do the practical sessions you will use the following communication ports of the PC (see figure 1):

- `ttyS0` (COM1 in windows): You will connect here the terminal to be able to configure the CISCO routers and commuters.
- `e0`, `e1`, `e2`: three Ethernet cards. The operative system gives the names `eth0`, `eth1`, `eth2` to these cards. There’s a problem, though, that the physical position of the card with the same name can change from a PC to another. In order to make the cards positions correspond with their physical position in all the PCs, the images use the command ifrename/iftab in the boot phase. With this command we rename the `eth0`, `eth1`, `eth2` physical interfaces with the names `e0`, `e1`, `e2`, in such a way that they are placed as the figure 1 indicates. Bear in mind that even though in this manual we could use sometimes the by default names (`eth0`, `eth1`, `eth2`), you should use the names `e0`, `e1`, `e2` depending on the card what you want to use (which we can identify by their physical position in the PC).

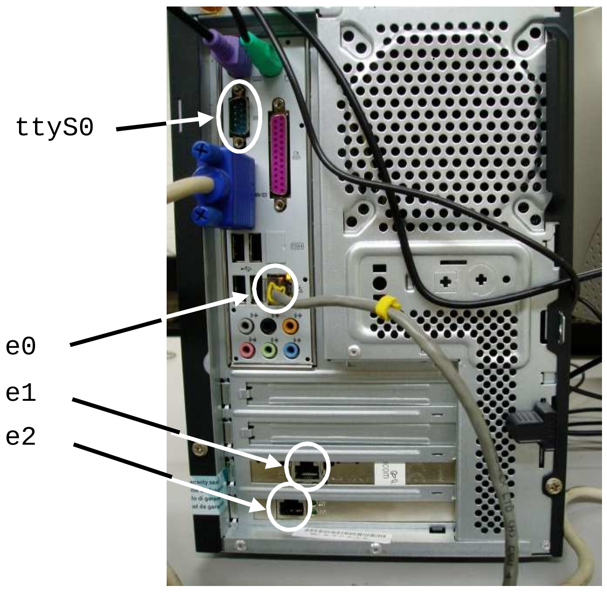


Figure 1: Communication ports of the lab PCs that we will use in the sessions.

2.1. Identification of the name of the Ethernet interfaces

In some PCs of the laboratory the name the interfaces does not correspond with e0, e1 and e2, as described above. The reason is that there are Ethernet cards that stopped working and were changed. Here's a simple method to determine the name of the interfaces, and what is their physical location on the PC.

First you need to determine the name that Linux assigned to the interfaces. This can be done running "ifconfig -a", as shown below:

```
xc# ifconfig –a
eth3        Link encap:Ethernet  HWaddr 08:00:27:4E:4C:C7
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:10 Base address:0xd020

eth1        Link encap:Ethernet  HWaddr 08:00:27:BE:7D:7F
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:9 Base address:0xd240

eth4        Link encap:Ethernet  HWaddr 08:00:27:5A:83:86
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:11 Base address:0xd260
```

From the dump we can see that, in this example, the name of the interfaces is eth3, eth1 and eth4. Now we need to determine the physical position of the interface on the PC. To do this we will take into account that in the cable that connects the PC to the network of the laboratory continuously arrive packets of the switch where it is connected. Therefore, we just have to capture packets with tcpdump. If we observe that packets arrive, it means that we are capturing on the interface where the network cable is connected. For example, to determine if the network cable is connected to eth3 we would execute:

```
xc# ifconfig eth3 up
xc# tcpdump –ni eth3
20:00:14.229940 STP 802.1d, Config, Flags [none], bridge-id
833e.00:11:5c:05:f5:40.8004, length 43
20:00:14.516673 STP 802.1d, Config, Flags [none], bridge-id
8004.00:11:5c:05:f5:40.8004, length 43
^C
```

From the previous dump we can see that the cable is actually in eth3. Then we would unplug the cable, connect it to another card, and repeat the previous commands with the name of another interface, for example eth1. If traffic arrives, it means that the card where the cable is connected is indeed eth1. Thus, the remaining one would be eth4.

Tools to review the labs

All the labs you do in the lab sessions can also be done at home with the tools explained below. It is advisable to do them if you have any doubt or have not had time to finish the lab in the laboratory.

1. For the labs using the PCs

Labs: 1 LINUX configuration, 6 TCP, 7 DNS

In the following link you have a virtual machine (VM) created using virtualbox (https://www.virtualbox.org) whit a linux image as the one installed in the lab.

http://studies.ac.upc.edu/FIB/grau/XC/slitaz50-xarxes.ova

To import from VirtualBox:

Fitxer → Import appliance

In order to have multiple VMs, clone the image as many times you need with virtualbox:

Select the image → clone (Mark the option "reinitialize the mac address of all network cards") → Linked clone

You can create a network of VMs to do the lab.

You will see that the VM is configured with 4 NICs. In order to connect several VMs: parameters → network → name and put the same name in all VMs to connect.

2. For the labs using IOS

Labs: 2 IOS Configuration, 3 RIP, 4 ACL and NAT, 5 Switches

In the following link you can download packettracer, a CISCO simulator. You only need to register to be able to free download de simulator.

https://www.netacad.com/about-networking-academy/packet-tracer/

The routers we have in the lab are 1841, the switches are 2950.

Lab 1. Basic commands for the IP level configuration with UNIX

1. The loopback interface

The first interface that we need to activate when configuring the IP level is the loopback interface. This interface is a kind of short circuit, meaning, the datagrams that are sent to this interface never leave the machine, they immediately return to the IP level that sent them. So, the loopback is used in the communication between processes that use TCP/IP inside the same machine. The network address assigned to the loopback is `127.0.0.0`. The interface is typically assigned to the address `127.0.0.1.` The name used in linux for this interface is `lo`. Moreover, the name `localhost` is typically assigned in linux to the address `127.0.0.1`

The available linux automatically configures the loopback interface.

2. The file `/etc/hosts`

In order to not having to use the IP addresses, a UNIX machines allows the assignation of names to IP addresses with the file `/etc/hosts`. For example, the content of this file could be:

```
xc# cat /etc/hosts
127.0.0.1              localhost
192.168.60.112         linux
192.168.60.101         pc1
```

Example 1: Contents of the file `/etc/hosts`.

When a name is given instead of an IP address to a command, this command does a call to the system resolver. The resolver looks first the file `/etc/hosts` for the resolution. If the name isn't there, then it looks if in the file `/etc/resolv.conf` a name server address is available. In a positive case, it will use the DNS (RFC1035) protocol to ask for the name resolution to the server

3. IP *forwarding*

The mechanism of IP *forwarding* consists in the transmission of a received packet through one of the physical interfaces of a node (a host or router) to another physical interface (which could be the same). The functioning is the following: The IP module has a function that processes the packets than have to be transmitted (`ip_output`) and one that processes the received packets (`ip_input`), in the way shown in the figure. If the IP forwarding functionality is not activated, the `ip_input` function discards all the packets that don't have as a destination one of the interfaces of the node. On the contrary, if the node has the IP forwarding activated, `ip_input` sends to `ip_output` all the packets that are received and don't have as a destination the same node.

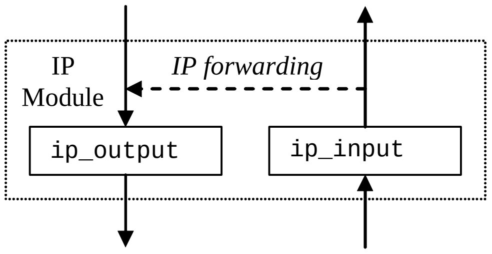


Figure 2: `ip_output` and `ip_input` functions.

A router has IP forwarding activated by default, given that its purpose is to route IP packets. A host, on the other hand, usually doesn't have this feature activated. In linux, the kernel can be compiled in a way that it has the IP forwarding feature with the following option:

```
IP forwarding/gatewaying (CONFIG_IP_FORWARD) [n] y
```

To activate it, you also need some of the kernel variables to have a value different to zero. You can see the values of these variables with the command showed in the following example:

```
xc# sysctl -a | egrep forward
net.ipv6.conf.default.forwarding = 0
net.ipv6.conf.all.forwarding = 1
net.ipv6.conf.eth2.forwarding = 1
net.ipv6.conf.eth1.forwarding = 1
net.ipv6.conf.eth0.forwarding = 1
net.ipv6.conf.lo.forwarding = 1
net.ipv4.conf.lo.mc_forwarding = 0
net.ipv4.conf.lo.forwarding = 1
net.ipv4.conf.default.mc_forwarding = 0
net.ipv4.conf.default.forwarding = 1
net.ipv4.conf.all.mc_forwarding = 0
net.ipv4.conf.all.forwarding = 1
net.ipv4.ip_forward = 1
```

Example 2: Forwarding variables of the kernel.

The image has IP forwarding activated by default. Alternatively, it can be activated executing:

```
root# sysctl –w ip_forward=1
```

4. Basic commands

In this section we will describe the basic commands to configure a UNIX machine. The description that is done here corresponds to the commands that are present in the *linux-xarxes* images prepared for the sessions. The parameters of these commands or it’s behaviour can change slightly in other UNIXs, Windows, or even in different linux distributions.

4.1. `ifconfig` Command

It allows the configuration of an interface. The typical ways of calling this command are:

```
ifconfig interface [IP_address netmask mask] [broadcast @broadcast]¹
```

Command 1: Assignation of an IP address and activation of an interface.

Where [] means optional parameters . Activates an interface and assigns an address to it. If the mask is not provided, the one corresponding to the IP address class is assigned. If the broadcast address is not provided, the OS assign the one that corresponds to the mask. To designate an Ethernet card, Linux uses the name `ethi`, where `i` is `0` for the first card, `1` for the second, etc. The names are automatically assigned by the kernel as their drivers are correctly loaded. Remember that the interfaces are renamed to ei.

For example, to assign an IP address and activate an Ethernet card:

```
xc# ifconfig e0 10.0.0.1 netmask 255.255.255.0
```

Example 3: Configuration of the `e0` interface.

If we want to deactivate an interface (f.e. e0) we have to execute:

```
ifconfig e0 down
```

Command 2: Deactivation of an interface.

And to activate it again:

```
ifconfig e0 up
```

Command 3: Activation of an interface.

To show the active interfaces we have to execute the command without parameters, as shown in the next example:

[1] We will use the following agreement: the keywords are in bold and the user parameters are not.

```
xc# ifconfig
e0      Link encap:Ethernet  HWaddr 00:40:F4:65:E6:BE
          inet addr:10.0.0.1  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::240:f4ff:fe65:e6be/64 Scope:Link
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:75 errors:0 dropped:0 overruns:0 frame:0
          TX packets:213 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:7236 (7.0 Kb)  TX bytes:86584 (84.5 Kb)
          Interrupt:10 Base address:0xbe00
...
```

Example 4: Listing of the configured interfaces.

If we want to list the interfaces known by the kernel (active or not) we have to execute:

```
ifconfig –a
```

Command 4: Listing of the interfaces known to the kernel.

4.2. `route` Command

It allows the addition or deletion of entries in the routing table and to show its content. The typical invocations are:

```
route add|del -net|-host destination [netmask mask] [gw gateway] [dev intf.]
```

Command 5: Usage of the route command.

Where | means alternative parameters and [] means optional parameters. If no mask is provided, the OS assigns the one of the class. If no interface is provided, the OS tries to deduce it from the assigned addresses. The Gateway has to be provided only if the destination network is not directly connected to one of the interfaces.

```
route [-n]
```

Command 6: Listing of the routing table.

Shows the contents of the routing table.

With the `-n` it shows the IP addresses in a numerical way. For example:

```
xc# route –n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.60.0     0.0.0.0         255.255.255.0   U     0      0        0 e0
```

Example 5: Listing of a routing table.

The default route has the address 0.0.0.0 and mask 0.0.0.0. If the gateway of the default route is, for instance, 192.168.1.1, in order to add the default route can be used one of the following commands:

```
xc# route add -net 0.0.0.0 netmask 0.0.0.0 gw 192.168.1.1
xc# route add [–net] default gw 192.168.1.1
```

Example 6: Adding a default route.

Note that the keyword “default”, is equivalent to “-net 0.0.0.0 netmask 0.0.0.0”.

4.3. `Arp` command

The `arp` command allows the visualization and manual modification of the table that maintains the ARP module (Address Resolution Protocol). In this table the correspondence between IP addresses and hardware addresses is stated. The typical invocations are:

```
arp
```

Command 7: Shows the ARP table.

```
arp –s IP_address hw_address
```

Command 8: Assigns the hardware address `hw_address` to the IP address `IP_address`.

```
arp –d IP_address
```

Command 9: Deletes the entry `IP_address` from the table.

4.4. `Ping` command

The `ping` command is a kind of sonar that lets you verify if a certain interface is in reach of the network level, and to measure the time from transmission to reception. *Ping* sends periodically a packet to the address that is given as a parameter that provokes the answering from the destination. To stop the *ping* you need to do a `CONTROL-C`. For example, to know if we can access the machine `192.168.60.200`:

```
xc# ping 192.168.60.200
PING 192.168.60.200 (192.168.60.200): 56 data bytes
64 bytes from 192.168.60.200: icmp_seq=0 ttl=255 time=0.6 ms
64 bytes from 192.168.60.200: icmp_seq=1 ttl=255 time=0.6 ms
64 bytes from 192.168.60.200: icmp_seq=2 ttl=255 time=0.6 ms
^C
--- 192.168.60.200 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.6/0.6/0.6 ms
```

Example 7: Ping to a remote machine.

We can also *ping* an interface of the same machine we are using, for example, to loopback:

```
xc# 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=0.1 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.0 ms
^C
--- 127.0.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.0/0.0/0.1 ms
```

Example 8: Ping to the *loopback* interface.

In example 10, a *broadcast ping* is done (the *broadcast* address is the one that has all the bits of the field *host* to 1). With this ping we can know the other machines connected to the same network.

```
xc# ping  192.168.60.255
PING 192.168.60.255 (192.168.60.255): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.1 ms
64 bytes from 192.168.60.7: icmp_seq=0 ttl=255 time=0.6 ms (DUP!)
64 bytes from 192.168.60.3: icmp_seq=0 ttl=255 time=0.7 ms (DUP!)
...
^C
--- 192.168.60.255 ping statistics ---
1 packets transmitted, 1 packets received, +12 duplicates, 0% packet loss
round-trip min/avg/max = 0.1/1.9/7.0 ms
```

Example 9: Ping *broadcast*.

The `DUPs` indicate that we have received more than one response packet to the same *ping*.

4.5. Traceroute command

Traceroute allows knowing the routers that a packet goes through until it reaches its destination. To know it, traceroute sends UDP 3-packet sequences to an arbitrary port (greater than 30.000) where it’s unlikely that some process is listening there to send back an answer. Each sequence is sent with a TTL that starts at 1 and increases until it reaches its destination. With this, the first 3 packets (sent with TTL=1) will be discarded by the first router, who will send back an ICMP error message of type “TTL=0 during transit” for each packet. The same thing will happen with the following packets until the destination is reached. In this case, the destination will discard the 3 packets (because there’s no one listening to the port that they are aimed) and will generate 3 ICMP error messages of type “port unreachable

The following example shows two executions examples of `traceroute`. On the first case a traceroute is done to a machine in the same network (named beco). On the second case it's done to a machine (rogent) placed in a different network, but it has to go through only one router (with name arenys5). The time shown in the output of traceroute is the time from the sending of the three packets to the receiving of the three respective answers. If a packet is lost, traceroute displays an asterisk.

```
xc# traceroute beco
traceroute to beco.ac.upc.es (147.83.35.81), 30 hops max, 40 byte packets
 1  beco (147.83.35.81)  1.747 ms  0.551 ms  0.531 ms

xc# traceroute rogent
traceroute to rogent.ac.upc.es (147.83.31.7), 30 hops max, 40 byte packets
 1  arenys5 (147.83.35.2)  0.918 ms  0.840 ms  0.762 ms
 2  rogent (147.83.31.7)  0.591 ms *  0.537 ms
```

Example 10: `traceroute`.

4.6. `Tcpdump` command

The tcpdump command allows the capture of the packets that arrive or leave from an interface. For example:

```
xc# tcpdump -ni e0
tcpdump: listening on e0
16:14:58.430994 arp who-has 10.0.0.2 tell 10.0.0.1
16:14:58.431080 arp reply 10.0.0.2 is-at 0:40:f4:65:e6:be
16:14:58.431150 10.0.0.1 > 10.0.0.2: icmp: echo request (DF)
16:14:59.430026 10.0.0.1 > 10.0.0.2: icmp: echo request (DF)
16:15:00.430034 10.0.0.1 > 10.0.0.2: icmp: echo request (DF)
^C
```

Example 11: `tcpdump`.

In this example, the option `–n` means that we don't want to do a name resolution (alternatively, tcpdump calls the resolver of the OS and keeps waiting some seconds until it gets the response). The `-i` option allows the specification of the interface that we want to listen. Next, tcpdump prints a line for each packet that it receives or sends. Each line starts with the packet capture time (with format: hours:minutes:seconds), followed by the source IP address and the destination IP address (if the datagram is IP), and some other relative information from the packet it has captured. In the previous example the packets captured with a ping are shown. In a following lab session we will study more deeply this command.

5. Practice realisation

5.1. First part: configuring a host

The objective of this practice is the configuration of the laboratory network such as the one showed in Figure 3.

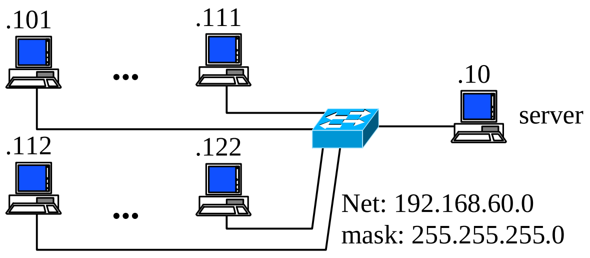


Figure 3: Laboratory network D6003.

There will be a network with IP `192.168.60.0`. The image server will be connected to the laboratory network but will be situated in another room. To assign IP addresses to the PCs we will use the following agreement: `192.168.60.<PC number>`. For example, if the number of the PC etiquette is 3, the PC will have as address `192.168.60.103`. To do this first part you have to follow these steps:

1. List the interfaces (the same as in Example 4) to check that only the `lo` interface is configured. List the routing table (Example 5) to check that the table is empty. List the ARP table (Command 7) to check that it is also empty.
2. Try to make a *ping* to `127.0.0.1`. Check that the same PC answers.

3. Try to make a *ping* to the broadcast address `192.168.60.255`. Check that the network is inaccessible.
4. Assign the IP address to the *ethernet* `e0` card using the aforementioned agreement. Check that the interface has activated listing the interfaces. Check if linux has added the entry at the routing table that allows accessing the network hanging from the *Ethernet* card. If it's not that way, use the `route` command to add it.
5. Try to make a *ping* to the *ethernet* card to make sure that it is accessible.
6. Make a *ping broadcast* to discover which machines are connected to the network. List the ARP table to see the *hardware* addresses of these machines.
7. Add the entry "`192.168.60.x pcx`" to the file `/etc/hosts`, where x corresponds to the IP address of one of the PC that replied to the broadcast ping. You can do it with the editor `vi`, `leafpad` (as root user) or simply by executing: "`echo 192.168.60.x pcx >> /etc/hosts`".
8. Try to make "`ping pcx`" to check that the machine is accessible.

5.2. Second part: configuring a linux router

The objective is configuring a PC as a *router* to be able to communicate PCs placed in different networks, as Figure 4 shows.

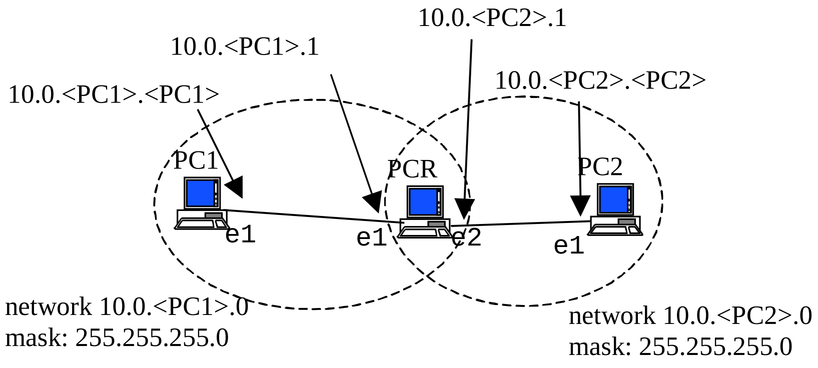


Figure 4: Topology with 1 router.

Note that to do this part you will need three PCs. So, according to the number of free PCs you can make groups of two or three. Write down the configured IP addresses in the following table:

PC1/e1	
PCR/e1	
PCR/e2	
PC2/e1	

To make the configuration we ask you to follow these steps:

9. Connect the interfaces that the figure shows with two crossover cables.
10. Deactivate the interface that you have used in the previous section (Command 2) and check with `ifconfig` that the interface is not active. Check with `route` that the entry of the routing table that used it has been deleted.
11. One of the PCs (from now on we will call it PC1) has to be configured to be in the network `10.0.<PC1>.0/24` with a *hostid* equal to `<PC1>` (where PC1 is the number of the PC).
12. Configure another PC (from now on we will call it PC2) to be in the network `10.0.<PC2>.0/24` with a *hostid* equal to `<PC2>.`
13. Configure the third PC to make it act as a router between the two networks (from now on we will call it PCR) assigning a *hosted* equal to `<PCR>` to the interfaces. The configuration of a PC as a *router* is exactly the same as it wasn't a router. The only difference is that in the case of the *router* there will be more than one interface (one for each network connected to it).
14. Make a *ping* from PC1 to PCR to check that it is accessible. Make a *ping* from PC2 to PCR to check that it is accessible. If there's no connectivity, it is possible that the cable is not connected to the appropriate interface. For example, if the card where you have connected the cable in PC1 is the one that linux identifies with `e2` but you have configured `e1`, PCR won't receive the packets sent by PC1. In this case, use `tcpdump` and `ping` to discover which physical card corresponds to each interface.
15. If you make a *ping* from PC1 to PC2 you will check that they cannot communicate. This is so because we still have to modify their routing tables to use PCR. Add the entry in the routing table of PC1 to make it

have PCR as default gateway. Make a *ping* to PC2 and check that you still can't communicate. This is so because PC2 receives the packet that PC1 sends (through PCR) but it still doesn't know how to answer it. You can check with tcpdump that PC2 effectively receives the ping of PC1. Add the entry in the routing table of PC2 to make it have PCR as default gateway. Try to do a *ping* now from PC1 to PC2 and vice-versa to check that they can communicate now.

16. Use the command `traceroute` to check that PC1 communicates with PC2 through PCR.
17. Look with tcpdump the traffic generated by the command traceroute.
18. In the configuration of the routing table of the *hosts* (PC1 and PC2) you have put an entry with the network where it is connected (linux automatically adds it when you give an IP address to the interface) and another entry with a *gateway* that allows you to reach another network. In reality, the *hosts* are usually configured with an entry to access the network where they are connected and a *by default* entry where all the *datagrams* intended to the rest of the Internet are sent. Change the configuration in PC1 and PC2 substituting the routes to other networks by a default route. Check that there is connectivity between all PCs.

5.3. Third part: interconnection of the networks of 2 groups

19. Join the networks of 2 groups to accomplish the schema of Figure 5. Use the command traceroute to check that the connection between PC1 and PC1' goes through the 4 routers. Write down the configured IP addresses on the table below. NOTE: if there aren't 2 groups available it can be configured the network of Figure 6.

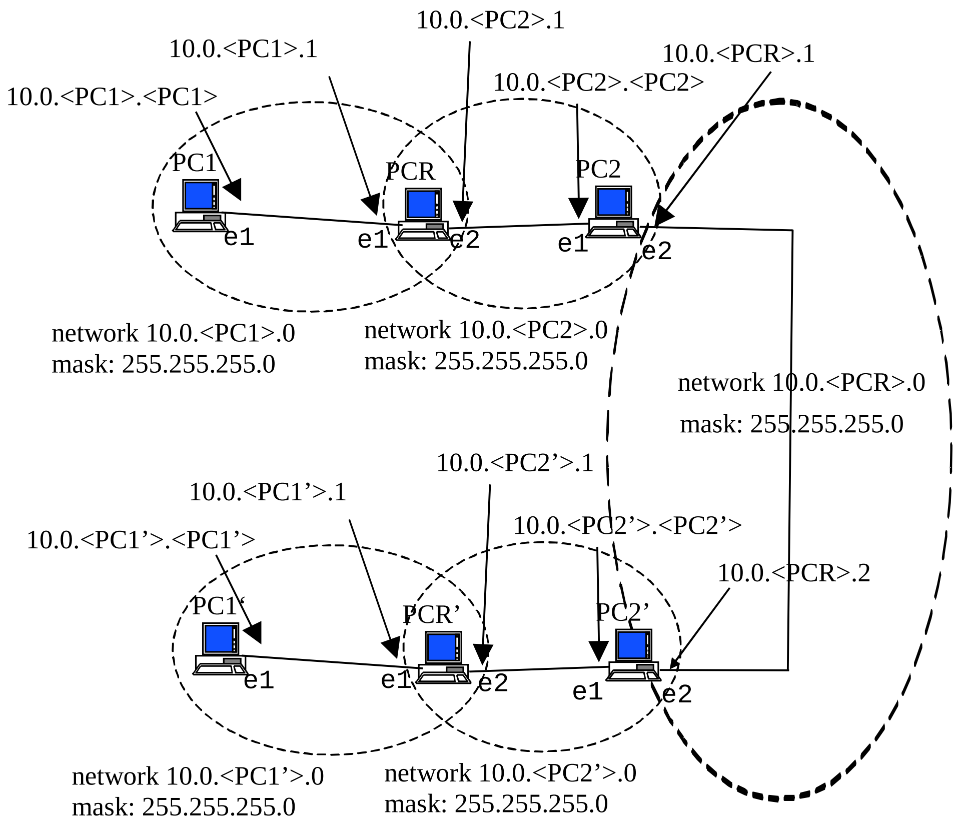


Figure 5: Topology joining the networks of 2 groups.

PC1/e1	
PCR/e1	
PCR/e2	
PC2/e1	
PC2/e2	
PC2'/e2	
PC1'/e1	
PCR'/e1	
PCR'/e2	
PC2'/e1	

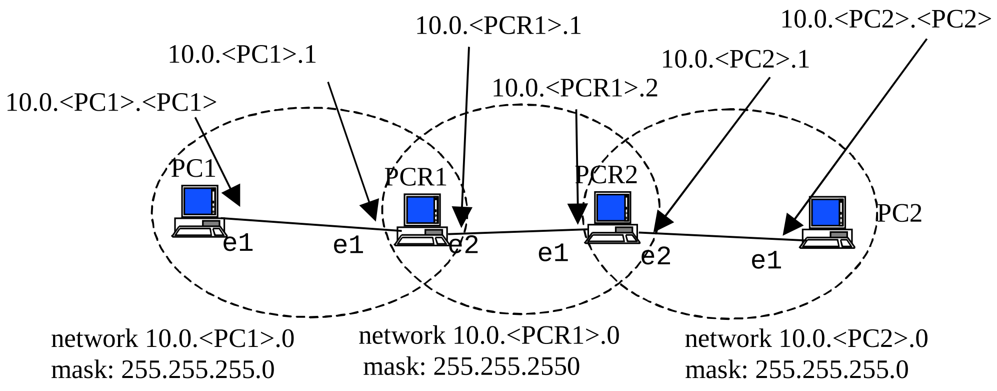
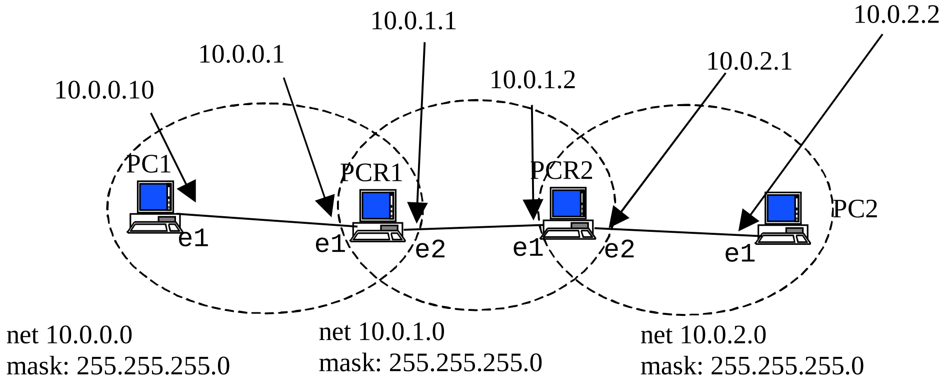


Figure 6: Topology with 2 routers.

PC1/e1	
PCR1/e1	
PCR1/e2	
PCR2/e1	
PCR2/e2	
PC2/e1	

6. Previous report



Answer to the following questions for the network of the figure:

1) Say which commands should be executed in PC1 to assign the IP address to the network interface and to use PCR1 as a by default router.
2) Say which commands should be executed in PCR1 to assign the IP addresses and to use PCR2 as a gateway to reach the 10.0.2.0 network.
3) Assume that, with the network configured, in PC1 the command “traceroute 10.0.1.2” is executed. How many UDP messages will PC1 send? How many ICMP messages will PCR1 and PCR2 send?
4) Assume that the routing table of PC1 is the one showed in the following dump. Assuming that the rest of the network is correctly configured, say which of the PCs would be accessible from PC1 (would answer a ping).

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.0.0         0.0.0.0        255.255.255.0   U     0      0        0 e1
10.0.2.0        10.0.0.1        255.255.255.0   U     0      0        0 e1
```

Lab 2. Routers CISCO: IOS

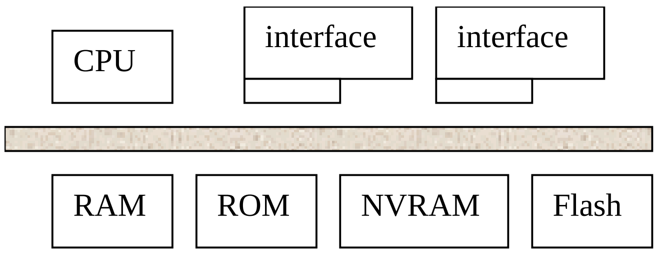
1. Objective of the practice

The objective of the practice is to know the basic concepts of the configuration of routers with the IOS operating system ("Internetworking Operating System") from the router manufacturer Cisco Systems.

2. Router structure

An IP router is a computer specialized in commuting IP datagrams. Depending on the features that it has to provide, its internal structure is more or less complex and specialized, but for the low-end models, we can think in a structure similar to a PC: CPU, memory, buses and network interfaces. For the data storage it is common to use ROM memory, flash memory and RAM and non-volatile RAM memory (NVRAM):

- RAM: code, routing tables, buffers, ARP cache, etc.
- NVRAM (non-volatile): "startup-config" configuration files.
- Flash (non-volatile): IOS image
- ROM (non-volatile): IOS image part, bootstrap code.



The operative systems of the commercial routers are specially designed to ease the packet commutation tasks, the execution of routing algorithms, interface configuration, etc. An example of this type of operative system is the IOS. The IOS has a simple architecture and usually occupies a reduced memory space. When we turn on a router, a bootstrap program, loaded in the ROM, is executed and it tests the system and loads in the RAM an IOS image, usually from a flash memory.

We will configure the router using a command line interface (CLI). It is usually done through a connection by serial line connected in the CONSOLE port of the router, using for example the HYPERTERMINAL application in Windows, MiniCOM in Linux, etc. The necessary parameters to connect are the following: Baud Rate 9600 bps, 8 bits/character, 1 Stop bit, 1 Non-Parity bit and 1 Non-Hardware-Flow-Control bit.

The active configuration of the router is found in a file called `running-config`. If we turn of the router, that configuration will be lost and will not be present when activating the router again. We can save this configuration in a configuration file (`startup-config`) that is usually stored in NVRAM memory. When booting the router, the configuration that is activated is the one stored in the `startup-config` file.

We can also configure the router accessing via telnet or using the web interface to configure the router. Also, both the IOS image and the configuration file can be obtained from a tftp server.

3. Configuration modes

The IOS routers feature a set of configuration modes that allow the visualization and configuration of the router. The configuration modes are the following:

- **BOOT or ROM monitor mode**: it is used in emergency cases (prompt typically `rmon`) as for example, the retrieval of a password, a configuration register, etc
- **SETUP mode**: allows a simple and basic router configuration via menu
- **USER EXEC mode**: it is the privileges visualization mode (prompt `R>`)
- **PRIVILEGED EXEC mode**: visualization mode with privileges (prompt `R#`)
- **Global Configuration mode or CONFIGURE**: allows the configuration of simple aspects of the router such as the configuration of the router's name, passwords, etc (prompt `R(config)#`)
- **Specific configuration modes**: allow the configuration of protocols, interfaces or, in general, more complex aspects of the router (prompt `R(config-if)#`, `R(config-route)#`, `R(config-line)#`, etc)

When booting the router we can enter the SETUP mode, which allows us to give a first configuration to the router when this lacks a pre-established configuration, or we can enter the USER EXEC mode, when the router has a pre-established configuration.

The first message that the router will emit when we connect to it will be:

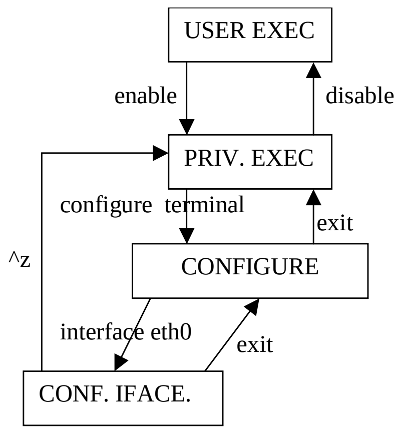
Continue with the configuration dialog [yes/no]: **no**

To which we have to answer **NO.**

In the USER EXEC mode we can consult basic aspects of the configuration of the router[2]. To consult more critical aspects of the router's configuration we have to pass to the PRIVILEGED EXEC mode. To pass from USER EXEC to PRIVILEGED EXEC mode we need to use a password (which is known as "*enable secret password*" which can be established from the CONFIGURE mode executing `enable secret <passwd>`).

From the USER EXEC and PRIVILEGED EXEC modes we cannot modify the router's configuration. To do it we have to pass from the PRIVILEGED EXEC mode to the general configuration mode (CONFIGURE). From there we can configure general aspects of the functioning of the router or pass to the specific configuration mode of each interface, routing algorithm, etc.

In the next figure the different configuration modes are shown together with the main commands to change from a mode to another.



When we are in the USER EXEC mode the prompt showed by the router is "**>**". When we are in PRIVILEGED EXEC mode the prompt is "**#**" and in the global configuration mode the prompt is **(config)#**.

For example:

```
Router> <commands in USER EXEC mode>
Router> enable
Router# <commands in PRIV. EXEC mode>
Router# config terminal
Router(config)# <commands in CONFIGURE mode>
Router(config)# exit
Router# disable
Router>
```

As we have mentioned, the changes in the configuration that are done in the global or specific configuration modes are stored in a configuration file that resides in the router's RAM called "`running-config`". This file can be seen from the privileged configuration mode using the command `''show running-config''`. If the router shuts down, these changes will be lost since they are stored in RAM memory. To maintain this changes and have them permanently stored in a NVRAM memory you have to copy the "`running-config`" file (RAM) into the "`startup-config`" file (NVRAM). This can be done from the PRIVILEGED EXEC mode with the "`copy running-config startup-config`" command.

IN THIS COURSE WE WILL NOT STORE THE ROUTER'S CONFIGURATION BETWEEN SESSIONS, SO YOU MUST NOT DO THE COPY OF THE MENTIONED CONFIGURATION

[2] With the command `?` we can obtain the list of commands that can be executed in each mode

4. Status consulting (show commands)

We can consult the status of a router using the command `show`. Depending on the type of information that we want to consult, the command can be executed from USER EXEC mode or from PRIVILEGED EXEC mode. For example:

show ip interface brief shows the status of the interfaces, their names and their configuration.

show running-config shows the active configuration file in the router.

show startup-config shows the configuration file stored in the NVRAM

show ip <parameter> shows the parameters associated to the IP protocol configuration. For example, the routing table is consulted with `show ip route`

show interfaces < interface_name> show the parameters associated to the interface

The routing table is a piece of information that is not considered privileged and that can be consulted from the USER EXEC mode. However, the contents of the configuration file are considered privileged and can only be consulted from the PRIVILEGED EXEC mode.

5. Basic configuration of the Router

Configure the name and the input messages (it is shown when connecting to the router)

```
R(config)# hostname WORD
```

To be able to access the router via telnet you have to assign a password to the vty terminals:

```
R(config)# enable password cisco → to do telnet you need to configure  password
R(config)# line vty 0 4 → configuration of 5 active terminals for telnet
R(config-line)# password cisco
R(config-line)# exit
```

6. Interface configuration

From the configuration mode we can start configuring the interfaces. For example, to configure an Ethernet interface we can do:

```
Router# configure terminal
Router(config)# interface e0
Router(config-if)# ip address @IP MASK
Router(config-if)# no shutdown
Router(config-if)# exit
Router#
```

Remember that, with the *show ip interface brief* you can consult the names of the interfaces.

The command “`no shutdown’’` is needed to activate the interface. By default, when booting the router all interfaces are deactivated. The command `‘’shutdown’’` would deactivate administratively an interface.

The serial interfaces are designed so that in the usual situation they connect to a telecom operator through a DCE (i.e. a MODEM or a Network Termination). The DCE is the one that usually sets the clock and fixes the modulation speed and consequently, the transmission speed.

If two serial ports are connected to the router (DTE-DTE) you have to use a crossover cable. Moreover, one of the two ports has to act as a DCE setting the clock. In principle, from the point of view of the router any of the two can act as a DCE, so the important thing is which connector of the cable is the one the marks the DCE port.

7. Serial interfaces

The nodes of a network can be classified in two big groups: data terminal equipment (DTE) and data communication equipment (DCE). The DTE are network devices that generate the destination of the data: PCs, routers, workstations, file servers, printing servers; they are all part of the final station group. The DCE are the intermediate network devices that receive and transmit the frames inside the network; they can be: switches, hubs, repeaters or communication interfaces.

The serial interfaces are designed so that in the usual situation they connect to a telecom operator through a DCE (i.e. a MODEM or a Network Termination). The DCE is the one that usually sets the clock and fixes the modulation speed and consequently, the transmission speed.

All the cables used to create a DTE-DCE link are serial cables, the cables used for DTE-DTE and DCE-DCE are crossover cables.

If two serial router ports are connected (DTE-DTE), you need to use a crossover cable. Moreover **one of the two ports has to act as a DCE setting the clock.** In principle, from the point of view of the router any of the two can act as a DCE, so **the important thing is which connector of the cable is the one the marks the DCE port**.

In the lab, the **RJ-45 type** serial cables will be white or grey, and the crossover cables will be red.

Once we know which port is the one acting as a DCE, it has to set the clock. We have to activate this option via IOS with the command `‘‘clockrate Bw’’,` where Bw are the bps that we want the line to work with. In the DTE port *we should not* activate this command. In the next example the Router-A is the one acting as DCE setting the transmission speed to 56 Kbps and the Router-B is the one acting as DTE.

```
Router-DCE# configure terminal
Router-DCE(config)# interface <name serial interface>
Router-DCE(config-if)# ip address <@IP> <MASK>
Router-DCE(config-if)# clockrate 56000
Router-DCE(config-if)# no shutdown
Router-DCE(config-if)# exit
Router-DCE(config)# exit
Router-DCE#

Router-DTE# configure terminal
Router-DTE(config)# interface <name serial interface>
Router-DTE(config-if)# ip address @IP MASK
Router-DTE(config-if)# no shutdown
Router-DTE(config-if)# exit
Router-DTE(config)# exit
Router-DTE#
```

The serial interfaces in the CISCO router use by default HDLC (High Data-link Level Control) encapsulation which is a standard protocol, but be aware, CISCO uses a proprietary version for their WAN links (only compatible with CISCO devices).

HDLC (Standard):	flag + address + control + data + FCS + flag
HDLC (Propietary):	flag + address + control + Propietary + data + FCS + flag

```
R(config-if)#   encapsulation hdlc          → CISCO HDLC
```

8. Name resolution

In the router you can assign IP addresses to names (the same as with the /etc/hosts file in a UNIX machine), Figure 7, and also to consult an unknown name to a DNS server (the same as with the /etc/resolv.conf file in a UNIX machine), Figure 8.

```
R(config)# no ip domain-lookup       → deactivates the DNS server lookup
R(config)# ip host NAME @IP1 @IP2    → assigns names to IP addresses
R(config)# show hosts                → lists a name and @IP cache
                                     (Configure an interface with ip host name @IP)
```

Figure 7: static DNS.

```
R(config)# ip domain-lookup
R(config)# ip name-server @IPserver1 ….. @IPserver6 → max 6 DNS servers
```

Figure 8: dynamic DNS.

Observation: By default, the router has the dynamic DNS resolution activated. If you enter in the command line a name that is not recognized as a command, the router tries to contact with a DNS server to resolve the name, and the terminal freezes for some seconds. If no name server is configured and you want to eliminate this wait, you can deactivate the dynamic DNS executing “no ip domain-lookup”.

9. Static routing

We will see next a static routing configuration example using the command `‘‘ip route’’`.

```
Router(config)# ip route @IPnet MASK @IPgw
```

The first address is the destination network address. Next we write the netmask associated with that network. The third address corresponds to the router's interface where the path is set (by default gateway).

10. Practice realisation

Each group must take a pair of routers connected by the serial port.

10.1. First part

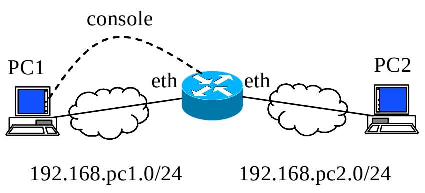


Figure 9: First part.

1. Configure the interfaces of the routers of the network in Figure 9. NOTE: The lower IP addresses of the subnetwork are usually used for the router's interfaces (because they are easier to remember), and the higher for hosts interfaces. Write down the configured IP addresses in the following table:

PC1/e1	
R1/e1	
R1/e2	
PC2/e1	

2. Configure the interfaces of the hosts connected with Ethernet and configure as router by default the interface of the corresponding router. Check the host has connectivity with the router using the ping command and see the format of the routing table of the host. Check that is possible to connect to the router using telnet.
3. Check that the hosts have connectivity amongst them.
4. See and interpret the format of the routing table of the hosts (command `''route -n'')` and of the router (command `''show ip route'').`
5. Configure telnet assigning the password cisco. Check that it is possible to connect to the router using telnet from the 2 PCs.

10.2. Second part

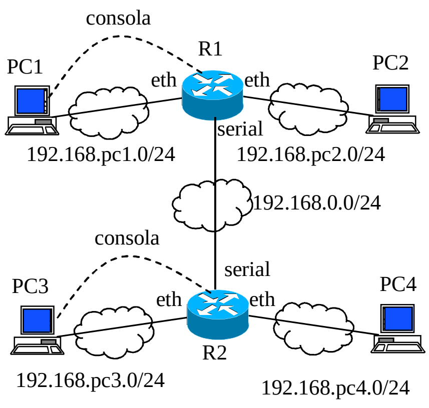


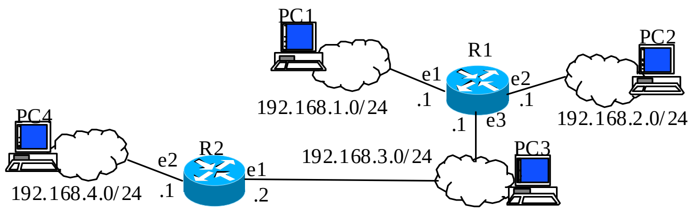
Figure 10: Second part.

6. Configure the network of Figure 10 using static routing in the routers (command “ip route”). Write down the configured IP addresses in the following table:

PC1/e1	
PC2/e1	
R1/e1	
R1/e2	
R1/s1	
PC3/e1	
PC4/e1	
R2/e1	
R2/e2	
R2/s1	

7. Check with the command “show interfaces” the type of encapsulation if the serial interfaces. Configure the serial link using HDLC encapsulation. Assign IP addresses to the ends of the serial link and check that there’s connectivity between the two routers.
8. Observe that from a host you only have connectivity with the interfaces directly connected to the router to which it is directly connected, and with other hosts situated in networks directly connected to the same router. Explain why.
9. Use the command `‘‘ip route’’` to add static entries in each router to gain connectivity with any subnetwork in the established network.
10. See the format of the routing table of the router with the command `‘‘show ip route’’` and check that you have connectivity with all the subnetworks of the network.
11. Use `traceroute` to check that PC1 reaches PC4 through the routers.
12. Remove the networks non directly connected from the routing table in R1 (command “no ip route”) and use a default route to R2. Check that the connectivity is maintained.

11. Previous report



Answer the following questions regarding the network of the figure:

1. Say which commands you should execute in R2 to assign the IP address to the network interfaces.
2. Say which commands you should execute in R1 and R2 to make all the networks accessible from all the PCs.
3. Assume that there is some error in the configuration. Say which router command allows you to see the current configuration.

Lab 3. Dynamic routing: RIPv1 y RIPv2

1. Introduction to RIP (RFC-2453)

The basic characteristics are:

- The hop count metric is the number of jumps until the destination: 1 if the destination is a network directly connected, 2 if it has to go through a router, etc.
- The routers send periodically (each 30 seconds) a broadcast RIP message in each interface with the known destinations and metrics. They are sent with UDP, source and destination port: 520.
- If we stop receiving RIP messages from a neighbour (180 seconds), we assume that it is down.
- The metric's value of infinity is 16.
- RIP version 2 introduces these changes: The netmask is added to the destinations sent in the messages. The messages are sent to the multicast address: 224.0.0.9 (*all RIPv2 routers*)

1.1. Count to infinity

The main problem of RIP is the convergence time: Meaning, the time spent from a change in the net topology until the routing tables are stabilized. This time can be especially long when there's the error called *count to infinity*. This happens when there's a change on topology and the sent sequence of RIP messages causes that a router A believes that it is able to reach a destination D, which is now inaccessible, passing through another router B which at the same time depends on A to reach D.

To solve the *count to infinity* problem the *Split horizon* modification is usually used. This modification consists in when sending a RIP message in an interface, the routing table entries that have a gateway in the same interface will be deleted.

Another modification used in the CISCO routers consists in the so called *holddown timer.* When a RIP message is received from a neighbour indicating that a network that has become inaccessible is now accessible through that router, then it marks the route and initiates a holddown timer. If when the timer expires the route is still perceived as accessible through that router, then the route is updated to include that router.

Another modification to accelerate the convergence consists in not waiting the 30 seconds to send a RIP message when a change in the routing table is produced. This technique is known as *triggered updates.*

2. Configuration of the RIP

To activate the RIP routing algorithm, the steps you need to follow are the following:

```
Router# configure terminal
Router(config)# ip routing
Router(config)# router rip
Router(config-router)# network @IPnet1
Router(config-router)# network @IPnet2
Router(config-router)# ^Z
Router#
```

Figure 11: Configuration of the RIP

The "`network`" command indicates the interfaces that have to send or process RIP messages. The network addresses have to be indicated without using subnetting (the mask of the class is assumed). For example, if the interface uses the IP address IP 10.5.4.2/24 we only need to announce the A class 10/8 with the format `''network 10.0.0.0''`. Note that the network command doesn't use a netmask, only the network address.

RIPv1 doesn't support subnetting. If we want a subnetted network we have to use RIPv2. The usage of version 2 is indicated after the "`router rip`" command, executing "`version 2`".

We can capture the packets that are sent and received with the command "`debug IP RIP`" from the PRIVILEGED EXEC mode. This option consumes a lot of resources from the system, that's why in normal operation it should be deactivated.

With the command "`show ip route`" we can observe the routing table of the router. In the information listed by the router, it appears indicated as if the route has been fixed in a static way or it has been learned with RIP.

The command `''show ip protocol''` allows you to see the RIP configuration. The command shows the RIP version activated in each interface both in input ("receive") and in output ("send"). Note that we can send

RIPv1 and we can receive both RIPv1 and RIPv2. The *hold down* time is the time that the router waits before accepting a new route for an entry that has been invalidated, to avoid the *counting to infinity*.

```
router# show ip prot
 Routing Protocol is "rip"
   Sending updates every 30 seconds, next due in 8 seconds
   Invalid after 180 seconds, hold down 180, flushed after 240
   Outgoing update filter list for all interfaces is
   Incoming update filter list for all interfaces is
   Redistributing: rip
   Default version control: send version 1, receive version 1,2
     Interface        Send  Recv  Triggered RIP  Key-chain
     Ethernet2        1     1, 2
     Ethernet3        1     1, 2
```

Figure 12: Dump of the command show ip protocol.

RIPv2 can be activated globally in all the interfaces with the command ‘‘`version [1 2]`’’:

```
Router# configure terminal
Router(config)# router rip
Router(config-router)# version 2
Router(config-if)# exit
Router(config)# exit
Router#
```

Figure 13: Activation of RIPv2.

If one of the routers would still use RIPv1 and would send RIPv1 messages, the interface would reject them. It is better to change the version per interface using the commands: ‘‘`ip rip receive version [1 2]`’’ and ‘‘`ip rip send version [1 2]`’’. We activate sending only with version 2 and receiving with both versions 1 and 2.

```
Router# configure terminal
Router(config)# interface e0/0
Router(config-if)# ip rip receive version 1 2
Router(config-if)# ip rip send version 2
Router(config-if)# exit
Router(config)# exit
```

Figure 14: Configuration to send RIPv2, but receive RIPv1 y RIPv2.

NOTES:

By default, the router does “route summarization”. The summarization is done in the class, and only when the messages are sent to a network with a different base direction. For example, is in the routing tables we have the subnets 10.0.1.0/24 and 10.0.2.0/24, when sending a RIP message to the net 192.168.0.0/24 it will be informed of the net 10.0.0.0/8. To deactivate the summarization you have to execute the command:

```
Router(config-router)# no auto-sum
```

Figure 15: RIP configuration to not do route summarization.

In order for the router to be aware of the static entries (this includes the by default entry), we have to execute the command:

```
Router(config-router)# redistribute static
```

Figure 16: RIP configuration to add the static entries in the men.

The router uses two metrics: the administrative metric and the routing algorithm metric. If several routes to a same destination exist, the route with the lower administrative metric is chosen. For example, RIP has administrative metric 120 and OSPF 110. If they both add an entry on the table to the same destination, the route added by the OSPF will be chosen first (the router considers it more trustworthy). When showing the routing table we can see the metrics between brackets:

```
R1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

R    192.168.3.0/24 [120/1] via 192.168.0.2, 00:00:08, Serial0
```

In the dump, the R indicates that it has been added by RIP. 120 is the administrative metric and 1 is the metric used by the protocol. The RIP metric showed by CISCO is the number of routers to reach the destination. When seeing the metric, the RFC says that the number of jumps until the destination has to be noted (meaning, if there's a router until the destination, we will do two jumps). For this reason, the CISCO router will increment in 1 the RIP metrics that are showed in the table when the RIP messages are sent.

3. Subnets with and without class

When we do subnetting, the first and last subnets are rendered useless. This happens because the subnet address of the first subnet coincides with the subnet address of the greater net (or subnetted) and the broadcast address of the last subnet coincides with the broadcast address of the greater net (or subnetted). In order to make the routers be able to work with the first and last subnet the IOS activates by default the command `''ip subnet zero''` (`''no ip subnet zero''` to deactivate the option).

A network can work with classes (A, B or C) or can use the non-class concept (CIDR). To be able to create subnets independently of the class, the IOS activates by default the command "`ip classless`". In fact, the command works in the following way: if it is active, the router sends the packets to the supernetted interface that better adjusts to the routing table (or to the by default route). In the case it is deactivated ("`no ip classless`") the router only re-sends the packet if the route is in the routing table (or there's a by default route). If there's not in the routing table, then the router discards the packet. For example, if the net 10.0.0.0/8 is subnetted and in the table we have the networks 10.0.1.0/24, 10.0.2.0/24 and a by default entry, when receiving a datagram directed to the network 10.0.3.0/24 with "`no ip classless`", the router will discard the datagram. With "`ip classless`", on the other hand, the router will send the datagram via the by default route.

4. Practice realisation

4.1. IP Network with subnetting. RIPv2 and summarization

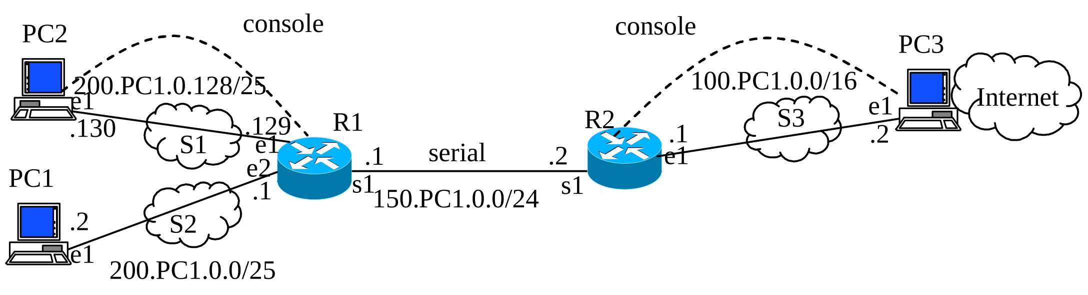


Figure 17

1) Configure the network of Figure 17. PC3 represents the router of the ISP that gives access to the Internet. Thus, the R2 router needs to have PC3 as default route. Furthermore, PC3 needs to have R2 as a router to reach the networks 200.PC1.0.0/24 and 150.PC1.0.0/24. Write down the configured IP addresses in the following table:

PC1/e1	
PC2/e1	
R1/e1	
R1/e2	
R1/s1	
PC3/e1	
R2/e1	
R2/s1	

2) Configure the interfaces of each router to send RIPv2 and receive RIPv1 and RIPv2. Configure RIP to be informed of the by default route.
3) Observe the activation of the RIP protocol with the command `''show ip protocol''` and interpret the output of this command
4) Observe the routing table with the command `''show ip route''` and check if there's connectivity between all the PCs.
5) Debug RIPv2 with the command `''debug ip rip''` (`''no debug ip rip''` to deactivate it). Interpret the messages.

6) Execute the command “no auto-sum” in the RIP configuration of the routers. How do the RIP messages and routing tables change?
7) Observe the convergence of the RIP protocol if we disconnect PC1, using the command `‘‘debug ip rip’’`. Interpret the messages. Observe how, when disconnecting, a certain time is spent until the tables converge and how a triggered update with infinite metric (16 jumps) is immediately sent. Connect again and observe the changes.
8) If we deactivate *split-horizon* in one of the interfaces, which networks will be announced in a RIPv1 routing message? To deactivate split-horizon you have to execute the command `‘‘no ip split-horizon’’` from the interface’s submode. For example, to deactivate split-horizon in the interface e0/0:

```
Router# configure terminal
Router(config)# interface e0/0
Router(config-if)# no ip split-horizon
Router(config-if)# exit
Router(config)# exit
```

4.2. CASE II: IP Network with subnetting. RIPv2 (2)

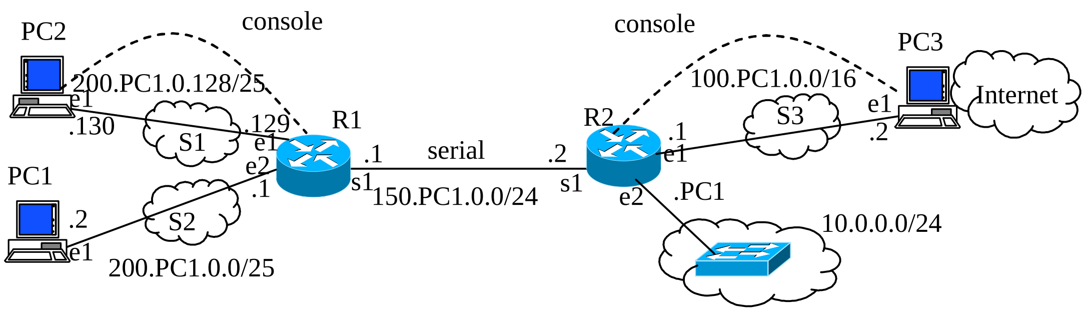
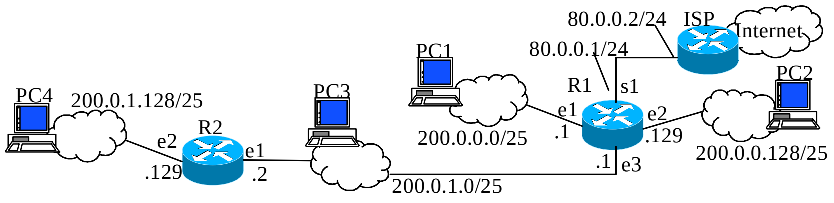


Figura 18

9) Interconnect the networks of various groups using a switch, as shown in the figure. Activate RIP in the net 10.0.0.0/24 and check the routing tables.

5. Previous report



Answer the following question for the network of the figure. Assume that the IP addresses of the figure have already been assigned to the interfaces.

1. Tell which command has to be executed in R1 to have a by default route to the ISP.
2. Say which commands should be executed in R1 and R2 to configure RIP. It is desirable that R1 is informed of the by default route.
3. Say which one will be the routing table of R1 and R2 when RIP had converged.
4. Assume split-horizon is used. Say the contents of the RIP messages that R1 and R2 will send in the 200.0.1.0/25 network.
5. Repeat the two previous questions if we execute “no auto-sum” in the RIP configuration of the routers.

Lab 4. ACLs and NAT with IOS Laboratory

1. Introduction

Access lists (ACL) are used for the filtering of packets depending on certain parameters like source or destination network addresses, source or destination ports, protocol type (ip. Icmp, tcp, udp, etc). One of the applications where they are most used is in network security. With the ACLs you can block the undesired traffic in an interface, be it input or output. The ACLs are not only used in the security field, they are also used to identify packets in applications like NAT (Network Address Translation), in BGP to filter routes when creating routing policies, etc.

There exist different ACLs for different protocol stacks: TCP/IP, IPX/SPX, Apple, etc. This document is centered in the ACLs applied to network security for the TCP/IP protocols stack. Each protocol has a range of the ACL assigned to it. For example, the ACLs from 1 to 199 are used for TCP/IP, while the ones between 800 and 999 are used for IPX/SPX, other ranges are used for DECnet (300-399), XNS (400-599), AppleTalk (600-699), etc.

When we create an access list and we apply it to an input or output interface, we are creating a sequence of instructions that are checked each time a packet enters or leaves through that interface. It’s important to note several characteristics of the ACLs.

First, an ACL is applied to the interfaces, either input or output. You can create an ACL for the output interface and another one for that input interface.

Second, the ACLs are sequences of instructions that are checked against the packet. The order of the instructions is important, since when one line of the sequence outputs true, some action is taken and then we leave the ACL, meaning that we don’t continue checking if another line of the sequence outputs true as well. That’s why it is really important to design the ACL in the sequence that best suits us.

For example, these two lines of an ACL:

- If the packet is icmp, deny it
- If the packet is ip, accept it

are not the same as these sequence:

- If the packet is ip, accept it
- If the packet is icmp, deny it

Assume an ICMP packet arrives. In the first case the packet would be rejected since the first line is met, the packet is ICMP. In the second case the ICMP packet would be accepted because the first line is also met, so the second line wouldn’t be checked.

Another important aspect is that we cannot insert lines on the sequence. If we make a mistake when we create it or we want to insert a line we have to delete the lines until the insertion point.

Finally, also REALLY IMPORTANT, the last line of the access line NEVER appears, meaning it exists in an implicit way and always DENIES EVERYTHING.

Inside the TCP/IP access lists there are two types of ACLs

- Standard IP access lists (1-99)
- Extended IP access lists (100-199)

2. Wildcard mask

The wildcard mask is a 32-bit mask that indicates which bits of the IP address have to be checked and which haven’t. If the bits of the mask are at 0 then they are checked, if they are at 1 then they are not.

For example, if we want to check if an entering packet belongs to the `145.34.5.6` IP address or not, we want to check all the bits of the IP address. This means that the wildcard mask would be `0.0.0.0`. In this case the tuple *`@IP WildcardMask`* is usually substituted by *`host @IP`*. For example the tuple *`145.34.5.6 0.0.0.0`* can be expressed as *`host 145.34.5.6`*.

If we would want not to check any bit, we would put a wildcard mask of `255.255.255.255`. In this case the tuple `@IP WildcardMask` is usually substituted by `any`. For example, the tuple `145.34.5.6 255.255.255.255` can be expressed as `any`.

We can also express networks. For example, you can check all the packets that come from the network `145.34.5.0/24`. This means that we have to check all the packets whose first 24 bits match with the ones in the network address. So, the wildcard mask should be `0.0.0.255`.

3. Standard ACLs

Standard ACLs only use the source addresses to make the comparison. Standard access lists have the numbers (`acl#`) between 1 and 99. The command has the following format:

```
access-list acl# {deny|permit} {@IPsource WildcardMask | host @IPsource | any}
ip access-group acl# {in |out}
```

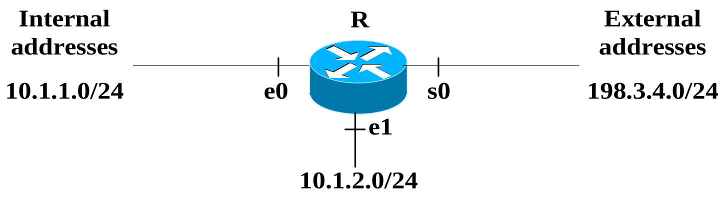
The first command, access-list, creates an access list with number `acl#` and with deny or permit condition over the specified source IP address with the corresponding wildcard mask. Remember that the last line of an ACL never appears and it's always "`access-list acl# deny any`".

The second command, access-group, assigns the access list acl# over the IP protocol over the input or output interface where the command is executed:

```
no access-list acl#
```

Example:

We want to deny in the s0 output interface any IP packet that comes from the network `10.1.1.0/24`.



```
R# configure terminal
R(config)# access-list 1 deny 10.1.1.0 0.0.0.255
R(config)# access-list 1 permit any
R(config)# interface s0
R(config-if)# ip access-group 1 out
R(config-if)# exit
R# show access-lists
```

We first create an access list with number equal to 1 and we deny all the traffic that comes from the network `10.1.1.0/24`. Since the last line would be deny everything (i.e.; the network `10.1.2.0/24`), we allow the rest of addresses. We apply this ACL over the output interface of s0 cause if we did it over the input e0 then we would block the packets from the network `10.1.1.0/24` to the network `10.1.2.0/24`.

4. Extended ACLs

Extended ACLs allow the use of both source and destination addresses to make the check. Also, they allow the specification of the protocol over which the check will be made and in the case it is TCP or UDP also specify the port. Extended access lists have numbers (`acl#`) between 100 and 199. The command has the following command:

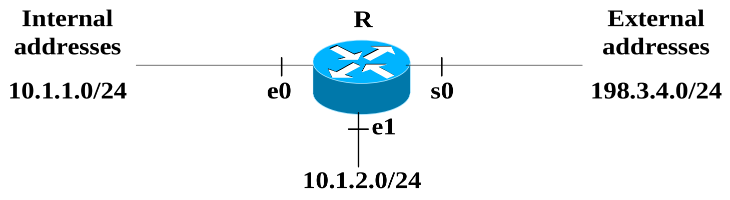
```
access-list acl# {deny|permit} protocol {@IPsource WildcardMask | host @IPsource | any} [operator
portsource] {@IPdest WildcardMask | host @IPdest | any} [operator portdest] [established]
ip access-group acl# {in |out}
```

The first command, access-list, creates the extended access list with number acl# and with the deny or permit condition over the source or destination IP address specified with the corresponding wildcard masks. `protocol` can be ip, icmp, tcp, udp, etc. *`Operator`* can be *{lt,gt,eq,neq}* (less than, greater than, equal, non equal) and `port` is a TCP or UDP port. `established` is only valid with tcp. When we use it, it captures the tcp traffic that belongs to an established connection. For it, the router looks into the packets with the ACK or RST bits activated (the first packet of SYN always has this two bits deactivated).

Remember that the last line of an ACL never appears but it is always "`access-list acl# deny ip any any`".

Example:

We want to deny in the s0 output interface any ICMP packet that comes from the net `10.1.1.0/24` and the access to any telnet port (port 23) from any host of that network.



```
R# configure terminal
R(config)# access-list 101 deny icmp 10.1.1.0 0.0.0.255 any
R(config)# access-list 101 deny tcp 10.1.1.0 0.0.0.255 any eq 23
R(config)# access-list 101 permit ip any any
R(config)# interface s0
R(config-if)# ip access-group 101 out
R(config-if)# exit
R# show access-lists
```

First we create the extended access list number 101, denying access to the ICMP packets, then another line denying access to any host with port 23, finally we allow any other type of traffic. Next, we apply the access list to the s0 output interface.

5. Verification

`R# show ip interface`	Show if there's any ACL in the interface.
`R# show access-lists`	Show the defined ACLs
`R# show running-config`	To check the configuration.

6. NAT

NAT (Network Address Translation) is the process that allows the translation from private addresses to public addresses using the substitution or alteration of the IP addresses or port in the IP and TCP headers of the transmitted packets. For NAT to be able to work we need a router that implements NAT in one or more of its variants: static NAT, dynamic NAT, PAT.
NAT is not always used to translate from private to public addresses. There are some cases where we want to translate from private to private addresses or from public to public addresses. The inside addresses can be both private and public. The typical case is the one where the inside address is a private address and the outside address is a public one. IOS uses the following generic naming when using NAT:

- **Inside local addresses:** the inside IP address assigned to a host in the inside network
- **Inside global addresses:** the IP address of a host in the inside net as it is represented to the outside nets
- **Outside local addresses:** the IP address of an outside host as it appears to the inside network
- **Outside global addresses:** the IP address assigned to an outside host on the outside network

See that the difference between a local and global address is that the inside local address is the one that we want to translate and the inside global address is the address already translated.

7. Static NAT

We use static NAT when the addresses are stored in a lookup table in the router and a direct mapping is established between the inside local addresses and the inside global addresses. This means that for each inside local address exists an inside global address. This mechanism is usually used when the objective is changing from a network address schema to another one or when there are servers that need to maintain a fix IP address to the outside such as DNS or Web servers.

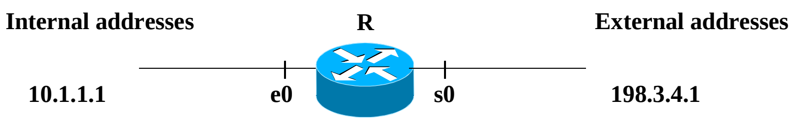
7.1. Static NAT configuration

To configure static NAT we will follow these steps:

- Define the static address mapping:
  ip nat inside source static *local-ip global-ip*

  ip nat inside source static network *local-network global-network mask*
- Specify the inside interface:
  ip nat inside
- Specify the outside interface:
  ip nat outside

Example:



```
R# configure terminal
R(config)# ip nat inside source static 10.1.1.1 198.3.4.1
R(config)# interface e0
R(config-if)# ip nat inside
R(config-if)# exit
R(config)# interface s0
R(config-if)# ip nat outside
R(config-if)# exit
R(config)# exit
R#
```

8. Dynamic NAT

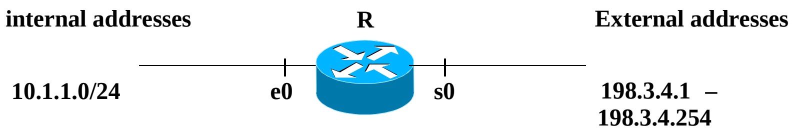
We use dynamic NAT when we have a set of inside global addresses that will be assigned dynamically and temporally to the inside local addresses. This assignation will be made when traffic is received in the router and it has a timer assigned.

8.1. Dynamic NAT configuration

To configure the dynamic NAT we follow these steps:

- Create a set of global addresses:
  ip nat pool *name start-ip end-ip* {**netmask** *mask* | **prefix-length** *prefix-length*}
- Create an ACL that identifies the hosts for translation:
  access-list *access-list-number* **permit** *source {source-wildcard}*
- Configure dynamic NAT based on the source address
  ip nat inside source list *access-list-number* **pool** *name*
- Specify the inside interface:
  ip nat inside
- Specify the outside interface:
  ip nat outside

Example:



```
R# configure terminal
R(config)# ip nat pool fib-xc 198.3.4.1 198.3.4.254 netmask 255.255.255.0
(config)# access-list 2 permit 10.1.1.0 0.0.0.255
R(config)# ip nat inside source list 2 pool fib-xc
R(config)# interface e0
R(config-if)# ip nat inside
R(config-if)# exit
R(config)# interface s0
R(config-if)# ip nat outside
R(config-if)# exit
R(config)# exit
R# show ip nat translations
```

The entries are assigned for 24 hours by default. If you want to modify the value of the timeout use the following command:

```
R(config)# ip nat translation timeout seconds
```

Where “seconds” is the time that will be assigned to the timeout.

9. NAT overload or PAT (Port Address Translation)

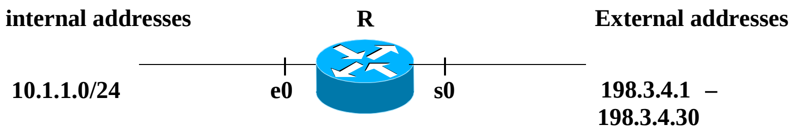
We use PAT (NAT by ports) when we have an inside global address and we want it to be able to redirect an entire big set (hundreds) of inside local addresses. This assignation is made using the pair global address/port. Even though we have at our disposal 65535 ports (16 bits) the PAT router can only use a subset of these ports (depends on the router, but approximately 4000 ports by global address). PAT can be used together with dynamic NAT in such a way that several global addresses with multiple ports direct a greater number of inside local addresses.

9.1. PAT configuration

To configure PAT we will follow these steps:

- Create a set of global addresses (can be only one address):
  ip nat pool *name start-ip end-ip* {**netmask** *mask* | **prefix-length** *prefix-length}*
- Create an ACL that identifies the hosts for translation:
  access-list *access-list-number* **permit** *source {source-wildcard}*
- Configure PAT based on the source address:
  ip nat inside source list *access-list-number* **pool** *name* **overload**
- Specify the inside interface:
  ip nat inside
- Specify the outside interface:
  ip nat outside

Example: we will use up to 30 inside global addresses, each of them doing PAT



```
R# configure terminal
R(config)# ip nat pool fib-xc 198.3.4.1 198.3.4.30 netmask 255.255.255.0
(config)# access-list 2 permit 10.1.1.0 0.0.0.255
R(config)# ip nat inside source list 2 pool fib-xc overload
R(config)# interface e0
R(config-if)# ip nat inside
R(config-if)# exit
R(config)# interface s0
R(config-if)# ip nat outside
R(config-if)# exit
R(config)# exit
R# show ip nat translations
```

In the case we don't have a set of global addresses we can use the address assigned to the "s0" interface in the following way:

```
R(config)# ip nat inside source list 2 interface s0 overload
```

10. Verification of a NAT configuration

We use the following commands to verify that the NAT configuration is correct (from privileged mode):

show ip nat translations

show ip nat translations verbose

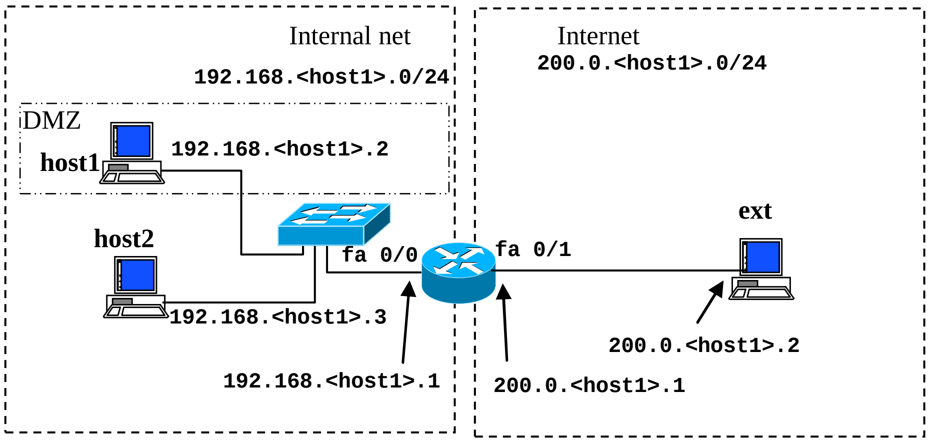
show ip nat statistics

debug ip nat (no debug ip nat)

clear ip nat translation * → eliminates all the NAT translations

11. Practice realisation

11.1. NAT



1. Configure the network of the figure. Note that the network on the left represent a private network, and the network on the right represent the Internet, Note that in the private network there are private addresses (non-routable in Internet): Recall that the ranges of private addresses are 10.0.<host1>.0/8, 172.16.<host1>.0/12 and 192.168.<host1>.0/16. Host1 and host2 must have a default route using the router. Configure the host that represents the Internet (ext) to make it capable of only reaching public addresses. To do so, don't add any default route in the host ext. Write down the configured IP addresses in the following table:

host1/e1	
host2/e1	
R1/fa0/0	
R1/fa0/1	
ext/e1	

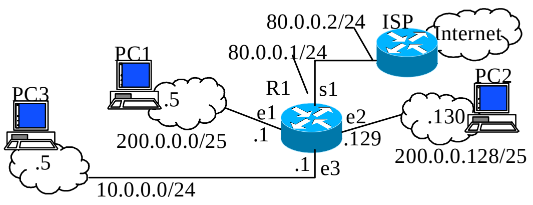
2. Check making *pings* that you have connectivity between host1-host2-router and between ext-router. Check that there's no connectivity between host1/2-ext (since ext cannot answer to the datagrams that arrive with a private source address).
3. Configure PAT to make all the hosts in the inside network able to access the Internet with the public address of the fa0/1 interface of the router (200.0.<host1>.1):
   - Check that both hosts in the inside network can access the Internet.
   - Check the functioning of NAT with `debug ip nat` (execute `no debug ip nat` to deactivate the command).
   - Check the NAT table (`show ip nat translations`).
   - Check that ext is not able to access the hosts of the Inside network (host1/2). Reason why it is not possible.
4. Configure a static NAT 200.0.<host1>.1 to host1. Check using "debug ip nat" in the router that ext can reach host1 using ping.

11.2. ACLs

Continuing with the previous configuration:

5. Configure a standard access list to make host2 the only one capable of accessing the Internet. Take into account that the order in which NAT and ACLs are applied is: first input ACL, then NAT and finally output ACL. Check that the ACL works correctly: host1 must not be able to reach ext but host2 must be. What happens with ext? Does it have access to host1? Why?
6. Change the configuration such that allows accessing from the Internet only to the ssh service (port 22) of host1. To check it, connect from ext to the ssh server of host1, and then try to connect with a telnet to host1. We wish that host2 has access to the Internet, but from host1 must not be possible to initiate a connection with the Internet. Check it confirming that it is possible to telnet from host2 to ext, but not from host1.

12. Previous report



1) Say which commands should be configured in R1 to make the PCs of the network 10.0.0.0/24 able to access the Internet with PAT with the public address that R1 has assigned to the s1 interface.
2) Assume that in R1 the following commands are executed:

```
Router(config)# access-list 1 permit 200.0.0.128 0.0.0.7
Router(config)# interface e1
Router(config-if)# ip access-group 1 out
```

Say which of the following hosts will be able to access to PC1, explain briefly why:

a) PC2

b) PC3

c) A host in the network 200.0.0.128/25 with address 200.0.0.250.

d) A host on the Internet with address 150.0.0.10

3) Say which commands should be added to previous ones to achieve the following:
   a) PC1 can access without restriction to the Internet, but not to the networks 10.0.0.0/24 and 200.0.0.128/24.

   b) From the networks 10.0.0.0/24 and 200.0.0.128/24 it is possible to access without restrictions to PC1.

   c) From the Internet it is only possible to access port 80 of PC1.

   d) PC1 answers to the ping of any host.

Lab 5. Switches

1. Introduction

An Ethernet switch is a layer 2 device that segments the collision domains. The configuration of a switch is totally dependent of the manufacturer. In this lab we will use 2950 CISCO's Ethernet switches. To enter and configure the switch we will follow the same steps we would follow with a CISCO router. We connect via the console port of the switch with a rollover cable and an application that allows the asynchronous communication using the serial port of the host (i.e. hyperterminal or minicom). Once connected, we enter the setup mode or the user exec mode. From the user exec mode we have to enter the privileged mode with the command "`enable`". In this mode we can visualize tables, configuration files (running-config), switch data bases, etc. To configure any functionality we have to use the global configuration mode using the command "`configure terminal`".

2. MAC Table

Each port of a switch is a collision domain. To segment the Ethernet net, a switch uses the MAC table. The switch has the table initially empty. Each time a station sends an Ethernet trace to another host, the switch "learns" to which port a MAC address is connected to. For example, if an Ethernet trace enters the e0 switch port with source address MAC=A and destination address MAC=B, the switch learns that MAC=A is connected to the port e0.

As the hosts send requests to other hosts and this answer, the MAC table keeps filling itself. Since the hosts can change its situation (start being connected to another port), it's not convenient that the MAC table entries are static. That's why the entries have a life time (age). When the life time expires, the entry of the MAC table disappears ("aging out"). That's why the entries are *dynamic.*

- Verification:

```
Switch# show mac-address-table
```

By default, a 2950 CISCO switch has assigned a live time for the MAC table entries of 300 seconds (5 minutes), dynamic mechanism and no static entry on the table.

To see the MAC table of a switch we can use the command "`sh mac address-table`". To see the life time you can use the command "`sh mac address-table aging-time`". To eliminate the entries learnt dynamically you can use the command "`clear mac address-table dynamic`" (all the entries) or "`clear mac address-table dynamic address @MAC`" (eliminate the @MAC address of the table) or `clear mac address-table dynamic interface IFACE`" (for the MACs of an interface) or `clear mac address-table dynamic vlan VLAN-ID`" (all the MACs of a VLAN).

3. VLANs

We define a VLAN as a broadcast network. Each one of the ports of a router is a broadcast network by definition and therefore an IP network. To save router ports broadcast networks (IP networks) can be created in a switch using software. This means that with a port of the router connected to the switch we are going to create as many VLANs (broadcast nets) as the switch allows us to. A 2950 CISCO router can create up to 1024 VLANs.

It is obvious that if a router port has to support N VLANs (N IP networks) the port should have N IP addresses, one for each created VLAN. It is also obvious that to travel from a VLAN to another one you have to obviously go through the router. Meaning that you cannot go from a VLAN to another one going through the switch, in the same way that the level 2 broadcast traffic (for example ARP traces) doesn't propagate between different VLANs. To achieve this level 3 segmentation a specific "trunking" protocol is used. A link in trunk mode belongs to more than one VLAN, in such a way that it allows you to send in one link all the traffic of the VLANs from the switch to the router (this configuration is known with the name *router-on-a-stick*). The traces that are sent in the trunk carry a tag with the number of the VLAN they belong to. Two trunking protocols exist: the one that was used for the first time, proprietary of CISCO, known as ISL, and the standardized by IEEE: IEEE802.1Q. In the CISCO equipment we can find both protocols (the more current equipment tends to carry only the IEEE802.1Q).

When we turn on a CISCO switch, all the ports belong to a native VLAN. The native VLAN, by definition, is the VLAN-ID=1. If a VLAN for specific use is defined it is better to use VLAN-ID different to 1. To define VLANs in a switch we follow these steps:

```
Sw# configure term
Sw(config)# vlan VLAN-ID
Sw(config-vlan)# name NAME
Sw(config-vlan)# exit
```

where VLAN-ID has range 0001 – 1005, WE CREATE the VLAN with NOMBRE and NUMERO. WARNING: VLAN 1, 1002, 1003, 1004 and 1005 are by default VLANs for different level 2 technologies (Ethernet, FFDI, TR, ...)

```
Sw# show vlan
Sw# show vlan id VLAN-ID
```

It lists parameters of all or a determined VLAN. To delete a VLAN:

```
Sw# configure term
Sw(config)# no vlan VLAN-ID
Sw(config-vlan)# exit
```

Once the VLAN is created you have to assign interfaces to the VLAN. Use the switchport command to assign statically ports to a VLAN:

```
Sw(config)# interface fastethernet0/1
Sw(config-if)# switchport mode access → defines VLANs in static mode
Sw(config-if)# switchport access vlan VLAN-ID → assigns the port to the vlan-id created vlan
Sw(config-if)# exit
Sw(config)# exit
Sw# show running-config interface IFACE → verifies the VLAN membership of the interface as it is in the physical memory
Sw# show interfaces IFACE switchport → lists the admin mode (i.e.; static access), the access mode of the VLAN (i.e.; vlan-id), etc
Sw# show vlan → lists information of the created vlans
```

Once the VLAN is created in the switch you have to define the link between the switch and the router as a “trunk” type link. A “trunk link” is a link that belongs to all the created VLANs. It has to be assigned to the native VLAN (VLAN=1). Only Fast Ethernet interfaces can be trunk.

```
Sw(config)# interface fastethernet0/1
Sw(config-if)# switchport mode trunk
Sw(config-if)# exit
Sw(config)# exit
Sw# show interfaces IFACE trunk
```

Now the switch is configured. We still have to configure the router for it to understand the different created VLANs. The link of the router has to be a “trunk link” and it needs to have as many IP addresses as created VLANs. For it we will create subinterfaces in the Fast Ethernet interface of the router. Each subinterface will be assigned to a VLAN and we will be given an IP. On the next example we will create 2 VLANs (VLAN-ID=2 and VLAN-ID=3) in the router. We use the Fast Ethernet 0/0 interface as the starting interface and we will create the Fast Ethernet 0/0.1 and Fast Ethernet 0/0.2 interfaces. We will assign the VLAN-ID to those interfaces with the command *encapsulation*. Finally, we will give an IP to each subinterface:

```
 R(config)#   int fastethernet 0/0
R(config-if)#  no ip address
R(config-if)#  no shutdown
R(config-if)#  int fastethernet 0/0.1
R(config-subif)#  encapsulation  dot1q  VLAN-ID2
R(config-subif)#  ip address @IP2 MASK2
R(config-subif)#  exit
R(config-if)#  int fastethernet 0/0.2
R(config-subif)#  encapsulation  dot1q  VLAN-ID3
R(config-subif)#  ip address @IP3 MASK3
R(config-subif)#  no shutdown
R(config-subif)#  exit
R(config-if)#  exit
R(config)#  exit
R#  sh ip route
```

Observe that in the routing table it has to appear an entry for each subinterface and its IP subnet.

4. Secure ports

There can be situations in which it interests us to fix the MAC addresses in the entry of the MAC table. For example, for security reasons we only want one port of the Ethernet switch to be able to physically connect with the host A. If we connect another host with a different MAC address we want the port to be disabled. With this we increase the security of our network. This solution is called secure ports. By default the secure ports option is disabled, to activate it in an interface:

```
Switch(config-if)# switchport port-security
```

To add secure ports:

- The port has to be in *access* mode. To change the mode of a port:

```
Switch(config-if)# switchport mode {access | dynamic {auto | desirable} | trunk}
```

Description:

Access	Set the port to access mode (either static-access or dynamic-access depending on the setting of the switchport access vlan inteface configuration command). The port is set to access unconditionally and operates as a nontrunking, single VLAN interface that transmits and receives nonencapsulated (non-tagged) frames. An access port can be assigned to only one VLAN.
Dynamic auto	Set the interface trunking mode dynamic parameter to auto to specify that the interface convert the link to a trunk link.
Dynamic desirable	Set the interface trunking mode dynamic parameter to desirable to specify that the interface actively attempt to convert the link to a trunk link.
Trunk	Set the port to trunk unconditionally. The port is a trunking VLAN Layer 2 interface. The port transmits and receives encapsulated (tagged) frames that identify the VLAN of origination. A trunk is a point-to-point link between two switches or between a switch and a router.

The default mode is **dynamic desirable**.

The way to achieve a secure port is to specify the maximum number of MAC addresses that can be associated to an Ethernet port and to fix the MAC addresses that we consider secure for that port. But first we have de empty the MAC table with the following command.

- To delete addresses from the MAC table:

```
Switch# clear mac address-table {dynamic [address mac-addr | interface interface-id
| vlan vlan-id]}
```

- To delimit the maximum number of MACs allowed in an interface:

```
Switch(config-if)# switchport port-security maximum max_addrs
```

- If we want to assign a secure MAC in a determined VLAN interface we have to execute:

```
Sw(config-if)# switchport port-security mac-address @MAC
Sw# show mac-address-table static
```

- Next the action to take when a port violation occurs is defined.

```
Sw(config-if)# switchport port-security violation {protect | restrict | shutdown }
```

where “protect” means that we discard the MAC traces that violate the system, “restrict” means that we also send a trap to the network manager (SNMP protocol) and “shutdown” (by default) means that the port is disabled.

- Verification:

```
Switch# show port-security [interface interface-id | address]
Switch# show mac-address-table
Switch# show running-config
```

NOTE:

When violating the security of a port, this remains blocked. To reactivate it, execute:

```
Switch(config-if)# shutdown
Switch(config-if)# no shutdown
```

5. Practice realisation

The lab configuration will be a router connected to a switch with a Fast Ethernet link (has to be Fast Ethernet to allow trunking). To each switch we will connect 3 PCs.

5.1. VLANs and trunking

1. Delete all user defined VLANs (for instance: configure term; no vlan 2). What happens when you try deleting VLAN=1?
2. Configure the topology of Figure 19. Create the stations T_1 and T_2 as belonging to the VLAN=2 and the station T_3 to the VLAN=3. Configure the router to accept VLANs. Write down the configured IP addresses in the following table.

T1/e1	
T2/e1	
R1/fe1.1	
R1/fe1.2	
T3/e1	

3. Check that you can do a ping to all the stations.
4. Check the MAC table of the switch. Identify the MAC address and VLAN of all PCs in the MAC table.
5. Observe the routing table of the router. Which entries does it have, and what is their format?
6. Execute tcpdump in the 3 stations to see the received/transmitted traffic.
7. Make a ping from T_1 to T_2. Which devices see the traffic? Why?
8. Make a ping from T_1 to T_3. Which devices see the traffic? Why?
9. Use the traceroute command between T1 and T2, and between T1 and T3. Reason the differences.

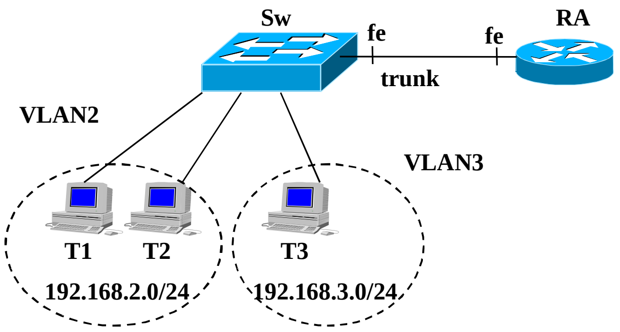
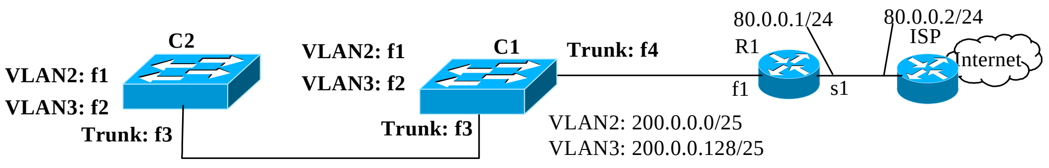


Figure 19

5.2. Secure ports

10. Configure a secure port in one of the stations (i.e. T_1). Configure the by default action to be disabling the port if another station connects to it. Disconnect the station T_1 and connect the station T_2. Observe how the port is disabled and connect again the original station. The behaviour has to be the following: if the action is *shutdown*, it will not accept again the original station and you will have to enable it manually (meaning, entering the interface of the switch and executing the commands *shutdown* and *no shutdown*).

6. Previous report



1. Give the commands to configure the switches and the R1 router of the figure. Assume that the hosted of the R1 router in each network is the lowest numeric address of the network.
2. Assume that in the C2 switch there's a PC1 connected to a VLAN2 port and a PC2 connected to the VLAN3 port. Say through which devices will go the packets if PC1 makes a ping to PC2.

Lab 6. TCP

1. Objectives of the practice

This practice has the objectives of studying the behaviour of the TCP protocol and learning the functioning of the `tcpdump` command, and specially knowing how to interpret this command.

2. Introduction to TCP

TCP is a protocol in the transport level that is used on the Internet for reliable information transmission. TCP is an end to end protocol, ARQ (*Automatic Repeat reQuest*), oriented to the connection, with the following objectives: (i) error recovery, in order to have a reliable transmission; (ii) flow control, to adapt the speed between the two communicating nodes; and (iii) congestion control, to adapt the speed of the network (and avoid the collapse of it).

TCP is a bidirectional protocol, and for each direction it behaves as depicted in Figure 20. As well as the application writes the information that has to be sent in the primary, TCP stores it into a transmission buffer. When the buffer is full, the OS blocks the application until there's space again. TCP keeps getting this information and sends it encapsulated inside the segments. As the segments arrive at the secondary, the information is stored in a reception buffer for the application of the secondary to read it. The objective of flow control is to avoid filling the reception buffer before the receiving application is able to read it (avoiding losses in the receiver). If the network is congested the losses will be produced in the buffer of a router along the way and will be called congestion losses.

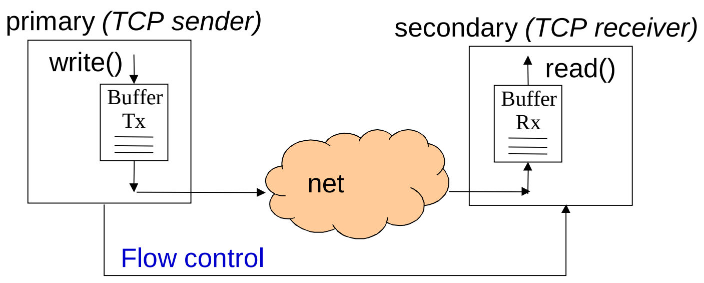


Figure 20: TCP level.

Figure 21 shows the header of a TCP segment. Besides the source and destination ports, the most important fields are the following:

- *sequence number*.
- *acknowledgement*.
- *Length*: size of the header in 32 bit words.
- *flags:* U (*urgent*): it is used in the field *urgent pointer*. A (*ack*): it is used in the field ack; P (*push*). Pass the information as soon as possible to the application. R (reset): abort the connection. S (syn): beginning of the connection. F (fin) ending of the connection.
- *Advertised window*: it is used for flow control.
- *Options*: The most important are: (i) mss (maximum segment size), suggests the size of the information field to the remote machine (typically the MTU of the network – 40). (ii) timestamp: to measure the round-trip delay (*round trip time*, RTT) and (iii) sack (*selective acks*): to give information about the lost packets to be able to do selective retransmission.

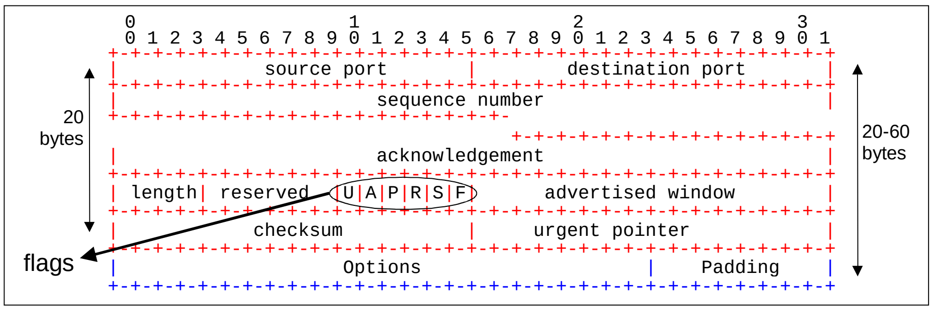


Figure 21: TCP header.

2.1. Establishment and termination of a connection

Figure 22 shows the establishment (*three way handshaking*) and termination phases of a TCP connection. The end that sends the first segment is by definition, the client. This segment does not carry any data, and it has only the syn *flag* activated. The server answers with a segment with the syn and ack *flags* activated, confirming the previous one. When the client sends back the ack the connection will be established. The termination is produced after sending segments with the fin *flag* activated and its respective acks.

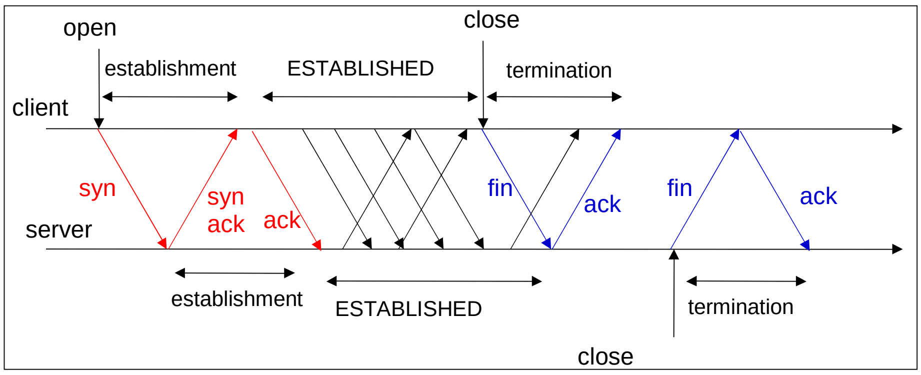


Figure 22: Establishment and termination of a TCP connection.

2.2. Sequence numbers

In TCP the sequence number identifies the first data byte that carries the segment. The first segment carries the *initial sequence number*, which is a 32 bit random number. From this value, the sequence number is incremented with the number of bytes that the segment carries (see Figure 23). The acknowledgement (ack) identifies the next byte that the secondary expects to receive (and confirms all the previous ones).

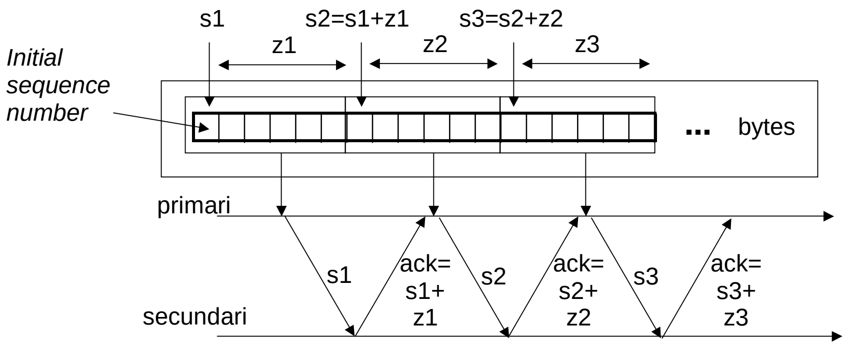
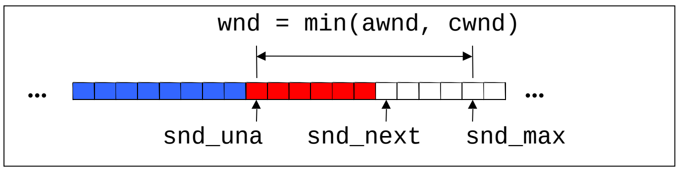


Figure 23: Evolution of the sequence numbers of TCP.

2.3. Window mechanism

TCP has a variable window mechanism that is given by: wnd = min(awnd, cwnd), where awnd is the is the window adverted by the remote node (flow control) and cwnd is the congestion window (see Figure 24). The adverted window is initiated every time a segment is sent to the number of free bytes in the reception queue. In this way, the primary will never send more bytes than the ones the secondary can store. The congestion window (cwnd) has the objective of adapting to the congestion status of the network. Its value is calculated from a set of algorisms. We will explain the most important ones



Legend:

🟦	Sent and confirmed segment
🟥	Sent but not confirmed segment
⬜	Segment still not sent
awnd	Advertised window
cwnd	Congestion window
snd_una	First non-confirmed segment
snd_next	Next segment to send
snd_max	Last segment that can be sent (if no new data is confirmed)

Figure 24: Window mechanism of TCP.

2.4. Congestion window

Typically, when several connections are share an Internet link, the TCP congestion control mechanism is responsible to ensure that each one of them gets a part of the link's transmission speed. If the connections transmit too much, then they will have losses and the connections will have to transmit less to adapt to the effective speed that they can obtain from the link. In this situation, the quantity of information that the connections can transmit is given by the size of the congestion window (cwnd). So, transmitting more or less is equivalent to increase or decrease the size of cwnd.

TCP uses two basic algorithms to calculate the cwnd: *slow start* (SS) and *congestion avoidance* (CA). The objective of SS is incrementing the cwnd as sooner as possible to a value where no losses are produced by congestion. From this point, cwnd is calculated with the CA algorithm. The objective of CA is to slowly increment cwnd to be able to get more transmission speed from the available one. The change from SS to CA is produced when a threshold maintained by the *slow-start threshold, ssthresh* variable is reached. Figure 25 shows the algorithms SS and CA. Note that mss is the size of a segment. So, when cwnd is increased with mss (as done in SS) another non-confirmed segment can be sent. When cwnd is increased with mss(cwnd (as done in CA), we will have to receive cwnd segments for cwnd to be increased in mss.

```
Initialization:
     cwnd = mss ;
     ssthresh = ∞ ;
When an ack that confirms new data is received:
     if(cwnd < ssthresh) /* Slow Start */
          cwnd = cwnd + mss ;
     else           /* Congestion Avoidance */
          cwnd = cwnd + mss*mss/cwnd ;
When the máximum waiting time for the confirmation of a segment
Is received (time-out)
     Retransmit the snd_una segment;
     cwnd = mss ;
     ssthresh = max(2, min(awnd, cwnd) / 2) ;
```

Figure 25 ***Slow Start*** **and** ***Congestion Avoidance*** **algorithms.**

Figure 26 shows the typical evolution of cwnd. When the connection is initiated, TCP starts with SS and the cwnd is incremented quickly until the adverted window (awnd). If the transmission is inside the same LAN, typically there will be no losses and the TCP window will be maintained constant and equal to awnd when cwnd reaches its value. If there are losses (due to the connection passing through a congested link) then *time-outs* of the non-confirmed segments will be produced and the non-confirmed segments will be resent, reducing each time the value of ssthresh to the value that had the window at the time of the timeout. In this case the evolution of cwn follows the shape of a saw tooth like in the figure.

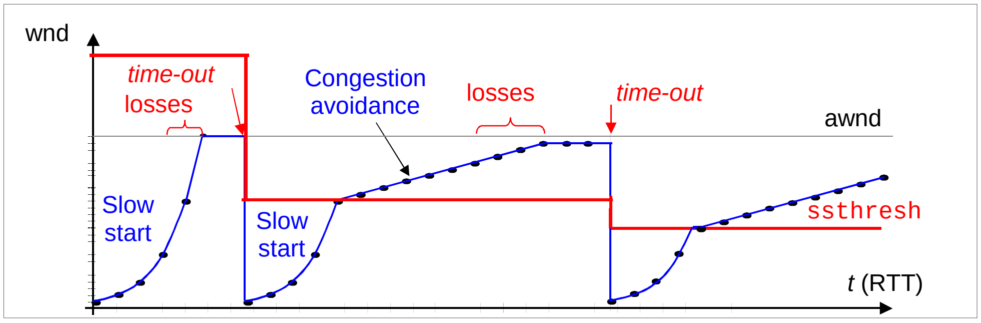


Figure 26: Typical evolution of the TCP window when there is a congested link.

3. The tcpdump command

The tcpdump command allows the capture of the packets that arrive or are sent through an interface according to a certain expression. The packets are captured in the moment they pass through or are received by the *driver* of the interface. By default, tcpdump uses the interface in promiscuous mode, to capture all the packets that reach it (either directed or not to the card where they are captured). The basic format of the command is:

```
tcpdump <options> <expression>
```

The most common options are:

- `-i <interface>`: capture the packets of `<interface>`. For example: `tcpdump –i e0`
- `-n`: To not resolve addresses into names.
- `-x`: To also do a hexadecimal dump of the content of the packet.
- `-X`: To do a hexadecimal and ASCII dump of the content of the packet.
- `-e`: To also print the link level header.
- `-s <n>`: To capture until <n> bytes of each packet (by default it captures 64 bytes).
- `-c <n>`: Capture <n> packets and end.
- `-v`: To be more verbose (give more information of the captured packets). We can use –vv and –vvv to receive even more information.

The most common expressions are:

- `src|dst host|net|port <i>`: capture the packets that have in the source|destination field the <i> host|network|port. For example: `tcpdump src net 10.0.0.0/24`
- `host|net|port <i>`: capture the packets that have in the source or destination field the <i> host|net|port. For example: `tcpdump net 10.0.0.0/24`
- `ip|arp|tcp|udp|icmp`: capture packets of one of these types.

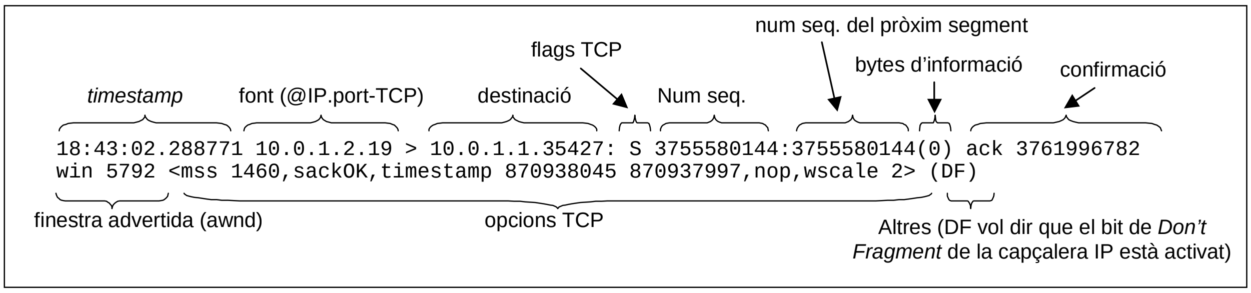
The expressions admit `and`, `or` and `not` operators. For example:

```
tcpdump –ni e0 icmp and host 10.0.0.1 and not host 10.0.0.2
```

will capture all the icmp packets that have as source or destination address 10.0.0.1 but that don't have as source or destination the address 10.0.0.2

3.1. tcpdump dump

Each time tcpdump captures a packet it does a dump indicating the information that tcpdump considers most interesting. The information that the dump shows depends on the type of the captured packet. Figure 27 shows the dump of a TCP segment. If we want more information we can ask tcpdump to also make a hexadecimal dump of the packet (option –x, and ascii with option -X).Figure 28 is an example of it.


num seq. del pròxim segment
flags TCP
bytes d'informació
confirmació
timestamp
font (@IP.port-TCP)
destinació
Num seq.
finestra advertida (awnd)
opcions TCP
Altres (DF vol dir que el bit de Don't Fragment de la capçalera IP està activat)

```
18:43:02.288771 10.0.1.2.19 > 10.0.1.1.35427: S 3755580144:3755580144(0) ack 3761996782
win 5792 <mss 1460,sackOK,timestamp 870938045 870937997,nop,wscale 2> (DF)
```

Figure 27: TCP segment dump.

```
xc# tcpdump -Xns 100 -i ppp0
...
18:56:02.628170 10.0.1.1.35434 > 10.0.1.2.21: P 1:17(16) ack 21 win 1460 <nop,nop,timestamp
871718463 871714750> (DF) [tos 0x10]
0x0000   4510 0044 bfe9 4000 3f06 65b8 0a00 0101        E..D..@.?.e.....
0x0010   0a00 0102 8a6a 0015 100a f0d1 105f 6243        .....j......._bC
0x0020   8018 05b4 bf2c 0000 0101 080a 33f5 5e3f        .....,......3.^?
0x0030   33f5 4fbe 5553 4552 2061 6e6f 6e79 6d6f        3.O.USER.anonymo
0x0040   7573 0d0a                                      us..
18:56:02.710769 10.0.1.2.21 > 10.0.1.1.35434: . ack 17 win 1448 <nop,nop,timestamp 871718503
871718463> (DF)
0x0000   4500 0034 2d96 4000 4006 f72b 0a00 0102        E..4-.@.@..+....
0x0010   0a00 0101 0015 8a6a 105f 6243 100a f0e1        .......j._bC....
0x0020   8010 05a8 3874 0000 0101 080a 33f5 5e67        ....8t......3.^g
0x0030   33f5 5e3f                                      3.^?
^C
```

Figure 28: Hexadecimal tcpdump dump.

It is necessary to note the following:

- The timestamp has format hours:minutes:seconds. Since the seconds are given with 6 decimals, we have a microsecond resolution (in the example of Figure 27, the packet has been captured at 18:43 and 2 seconds, 288 ms, 771 µs). The dump doesn't say if the captured packet has been received or transmitted. This can be deduced from the addresses. For example, if it has been captured in the machine 10.0.1.2, then the packet has been transmitted.

- To follow properly the trace, tcpdump gives the sequence number of the packet and the sequence number that the next packet will have. With this, we can easily see when the packets are transmitted without the proper order (which normally is an indication that some segments have been lost). Moreover, if tcpdump captures the syn packets, it normalizes the sequence number subtracting the initial sequence number to make it easier to read (as shown in Figure 28). Also, with this normalization the sequence number tells us directly how many information bytes have been sent. If a packet doesn't carry information bytes, typically tcpdump will not show the sequence numbers, only the confirmation (second packet of the trace in Figure 28).

Note that to stop the capturing of packets you have to press CONTROL-C. In the dump of the first segment of Figure 28 we can see that the IP header (you have it in Figure 29) has 20 bytes. We can deduce this cause the first byte of the dump is 45: 4 is the version and 5 is the size of the header in 32 bit words (meaning, 5 x 4 = 20 bytes). If we count 10 groups of 4 hexadecimal figures (the 20 bytes of the IP header) we arrive where the TCP header starts. From the TCP header (Figure 21) deduce that 8a6a is the source port, 0015 is the destination, 100af0d1 is the sequence number, 105f6243 is the confirmation and 8 is the size of the header (32 bytes). So, the TCP header carries 12 bytes of options (the *timestamp* option plus the *padding*).

```
              0                   1                   2                   3
              0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
             +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
          ^  |Version|  IHL  |Type of Service|          Total Length         |  ^
          |  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
   20     |  |         Identification        |Flags|      Fragment Offset    |  |  20-60
  bytes   |  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |  bytes
          |  |  Time to Live |    Protocol   |         Header Checksum       |  |
          |  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
          |  |                       Source Address                          |  |
          |  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
          v  |                    Destination Address                        |  |
             +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  |
             |                    Options                    |    Padding    |  v
             +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 29: IP header.

4. Practice realisation

To make the practice we will capture TCP segments from a connection between a client and a server that goes through a router, as shown in Figure 30. The links are Ethernet.

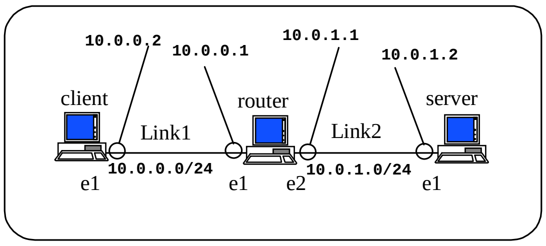


Figure 30: Practice topology.

To be able to capture TCP traces easily, in the "xarxes" image the sack and window-scaling options are deactivated. Otherwise, they can be deactivated with the commands in Figure 31.

```
servidor# sysctl -w net.ipv4.tcp_sack=0

servidor# sysctl -w net.ipv4.tcp_window_scaling=0
```

Figure 31: Deactivation of the options *sack* and *window-scaling* in the server.

Important comments about the "xarxes" image:

a. If the ftp server denies the connection with the client: It is because the inverse resolution of the client's IP address cannot be done (security measure implemented by the server). The problem is solved adding the IP address of the client to the /etc/hosts file of the server (with an arbitrary name, for example "10.0.0.2 client").
b. If the telnet client is really slow when connecting to the server: It is because the client tries to do the inverse resolution of the IP address of the server. Solution: delete the name servers that are in the /etc/resolv.conf file of the client.

c. If TCP segments larger than 1500 bytes are captured at client or server side is because a Linux optimization is being used. The optimization consists of incrementing the efficiency of the driver communication by aggregating several MSS in a single segment. Then, the segment is split at the driver side. This mechanism can be disabled executing:

```
# ethtool -K e0 tso off
# ethtool -K e0 gso off
```

a. In order to freeze the screen scroll, press Ctrl-S. Press Ctrl-Q to continue.

4.1. Analysis of the segments exchanged in a TCP connection

Configure the network of Figure 30. Make sure, making ping, that there's connection between the client and the server.

1. Execute tcpdump to capture the packets in link1 of the client with the option -X (Figure 32). In another client window, establish an ftp connection to the server, user "xc".
2. Execute "netstat –nat" in the client and server (in a third window). Identify the sockets that belong to the connection and their TCP state. Close the connection (command *bye)* and check again the TCP state of the sockets using netstat –nat.
3. Identify the *three-way-handshaking* and the termination of the tcp connection.
4. Look at the messages that are sent each time a command is executed in the ftp client. Relate the commands that you execute in the ftp application with the captured segments. Note how the content of the information segments carries ASCII messages without encryption. Look for the segment where the password has been sent, and where the command "dir" has been executed.

```
client# tcpdump –s 1500 –lnXi e1 port ftp
```

Figure 32: Ftp trace capture.

5. Turn on tcpdump to capture 200 packets in the e1 link of the client and Ethernet of the server, and make a connection with the chargen port (Figure 33). The chargen server (port 19) sends a pseudo-random sequence of ASCII characters at the maximum speed allowed by the link. Note that the server is the one sending the information segments.

From the two traces that you have captured:

6. Estimate the effective transmission speed of the connection. If you need a calculator, remember that you have one in the application bar on the desktop. Note that we can estimate the effective transmission speed from the sequence numbers of the first and last information segments of the trace captured in the client, divided by the time interval that exists between these two segments.
7. Check whether there are losses. Justify the reason why there are, or no, losses.
8. Observe the relation that exists between the received acks and the sequence number of the information segments sent after the ack. Note that in the trace captured in the server the difference keeps increasing, while in the trace captured in the client the difference is 0. Why is that?
9. Justify that in the trace captured in the server, the difference between the sequence number received in the ack and the one in the information segment that is sent next is approximately the number of information bytes of the "in flight" connection. Meaning, bytes sent by the server but that still haven't reached the client, and thus, are stored in the router or lost. Note that the size of the window that TCP is using is this difference plus the number of information bytes of the packets that are sent immediately afterwards (until an ack for new data is received). Relate the evolution of the window with the slow start.
10. Look at the evolution of the window advertised by the client and the server. Why do you think the client window increases but not the server's? Look at the last packets of the trace captured in the server. Compare the size of the window advertised by the client with the one that the server uses (deduced by the trace). Which window do you think is limiting the transmission: the adverted (awnd) or the congestion (cwnd)?

```
server# tcpdump -c 200 -nli e1 port chargen > captura-servidor.dmp
client# tcpdump -c 200 -nli e1 port chargen > captura-client.dmp
client# telnet 10.0.1.2 chargen
```

Figure 33: Capture of the connection trace to the chargen server.

11. In this experiment we need the client to stop reading the socket to fill it, and to send an adverted window (awnd) of 0. To stop the connection with the server and access the telnet prompt you have to press Ctrl-AltGr-]. From the telnet prompt you can exit with the quit command or continue when pressing the enter key. Connect the client to the chargen server executing "tcpdump -nli e1 port chargen" and stop the connection with "Ctrl-AltGr-]" "enter". Observe the evolution of the window adverted by the client. What does the server do when the window is 0?

12. With the chargen connection established, try tcpdump using expressions (see section 3). For example, the following commands capture the segments sent or received by the server, that is, data segments and acks, respectively.

```
# tcpdump -ni e1 tcp and src 10.0.1.2
# tcpdump -ni e1 tcp and dst 10.0.1.2
```

4.2. Lossy connection

To introduce losses we will add a queue with size and speed fixed by us in the output of link1 of the router, as shown in Figure 34. Linux allows to add this queue with the command of Figure 35 (the parameters of the queue can be modified changing `add` for `change` in the same command, and the queue can be deleted changing `add` for `del`). The packets will leave this queue with a speed of 100kbps. If the packets arrive at a higher speed, the queue will fill itself until a maximum of 10.000 bytes. The packets that arrive when the queue is full will be discarded. Note that queue only affects one way of the link:

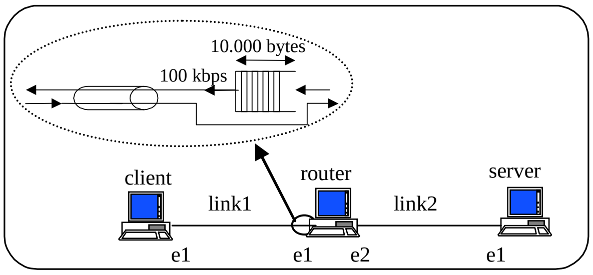


Figure 34: Queue that we will add at the output of the e1 router link.

```
router# tc qdisc add dev e1 root tbf burst 5000 rate 100kbit limit 10000
```

Figure 35: Command to add the queue of Figure 34.

13. Configure the queue and repeat the commands of Figure 33.
    a) List the trace captured in the server with *less*. Look for the first duplicate ack and check that the server retransmits the lost packet afterwards. Look at the trace captured by the client and check that effectively the packet was lost.
    b) Estimate which is the number of information bytes that exist “in flight” when the losses are produced. Check that it is higher than 10.000 bytes. Check that after the retransmission the tcp window keeps increasing slowly until there are losses again.
14. Estimate the value of RTT in the three-way-handshaking and when the losses are produced. Why is it different? Estimate which value should the RTT have given the delay that the buffer that we have added with the tc command introduces. Check that the RTT matches approximately with this delay.

5. Previous report

The following dump shows the timestamp of the first packet, and the last 5 packets of a tcpdump capture. Given the mentioned dump, answer the following questions:

```
1. 18:37:12.234583
2. …
3. 18:38:28.739407 IP 147.83.30.137.22 > 80.102.159.44.1035: P 4672:4801(129) ack 4805119 win 32480
4. 18:38:28.739652 IP 80.102.159.44.1035 > 147.83.30.137.22: P 4805119:4805151(32) ack 4801 win 2092
5. 18:38:28.739729 IP 80.102.159.44.1035 > 147.83.30.137.22: F 4805151:4805151(0) ack 4801 win 2092
6. 18:38:28.851394 IP 147.83.30.137.22 > 80.102.159.44.1035: F 4801:4801(0) ack 4805152 win 32480
7. 18:38:28.851458 IP 80.102.159.44.1035 > 147.83.30.137.22: . ack 4802 win 2092
```

1) Which are the IP addresses of the client and the server?
2) Give a possible dump for the part that is missing on the first line.
3) How many information bytes (content of the payload field) have the client and the server exactly sent?
4) Estimate the effective speed.
5) Say the TCP status that the client and the server are in the last 5 packets shown in the dump.

Lab 7. Domain Name System (DNS)

1. Introduction

In this practice we will deepen in the DNS protocol. To make the experiments that are detailed in the wording we will use one of the laboratory machines as a DNS server.

2. DNS

Client-server application used to do the name resolution (conversion from a name to an IP address). It consists of a distributed database. The entries are called Resource Records (RR) and can be of type:

- SOA: Start Of Authority.
- NS: NS name.
- MX: the domain mail exchange.
- A: A host address.
- CNAME: Canonical Name Record. E.g. the real hostname of www.foo.org is server.foo.org.

All the DNS messages have the format:

```
-------------------------------------------------
|             Header (12 bytes)                 |
-------------------------------------------------
/             Question (variable)               /
-------------------------------------------------
/             Answer (variable)                 /
-------------------------------------------------
/             Authority (variable)              /
-------------------------------------------------
/             Additional (variable)             /
-------------------------------------------------
```

Where the header is:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |      Flags                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         #Questions            |      #Answers                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         #Authorities          |      #Additional                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Question field:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                       QName (variable)                          /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         QType                 |      QClass                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

And the fields Answer, Authority and Additional are RRs:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                        Name (variable)                          /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Type                 |        Class                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              TTL                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         RDLenth               |      RData (variable)           /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3. Basic commands

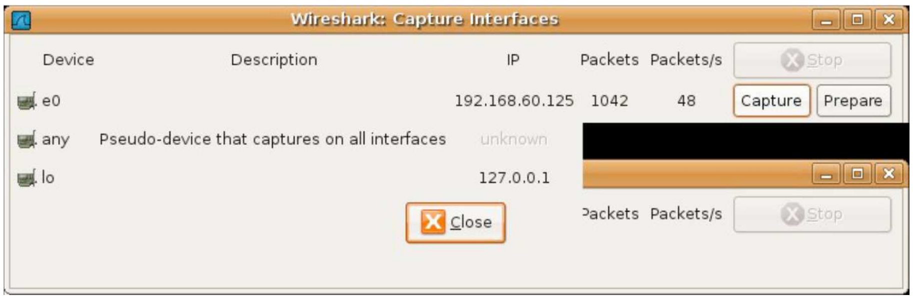
For the realisation of the practice we will use the following commands:

3.1. Wireshark

Wireshark (previously ethereal) is a network analysis tool. It allows you to monitor with a graphical interface the traffic passing through a network interface. In fact, it's the graphic equivalent to the tcpdump command. In this laboratory we will use this tool to see how the application level messages circulate during the realisation of the practice.

To start wireshark you need to execute as root user the command 'wireshark' (you have an available icon in the task bar of the desktop). It is necessary to have superuser privileges to make the tool monitor all the information that circulates through the interface independently of the process and/or user that generates the data, in such a way that it allows to spy the network activity of other users that are working in the machine where wireshark is executing.

Once the program is running, we can go to the menu 'capture->interfaces' (or directly click to the icon at the leftmost side), we will choose an interface (eX) that has an IP address, and we will press 'capture' to initiate the packet capture (see Figure 36). Once the activity that we want to capture has ended, we need to press the button 'stop'.

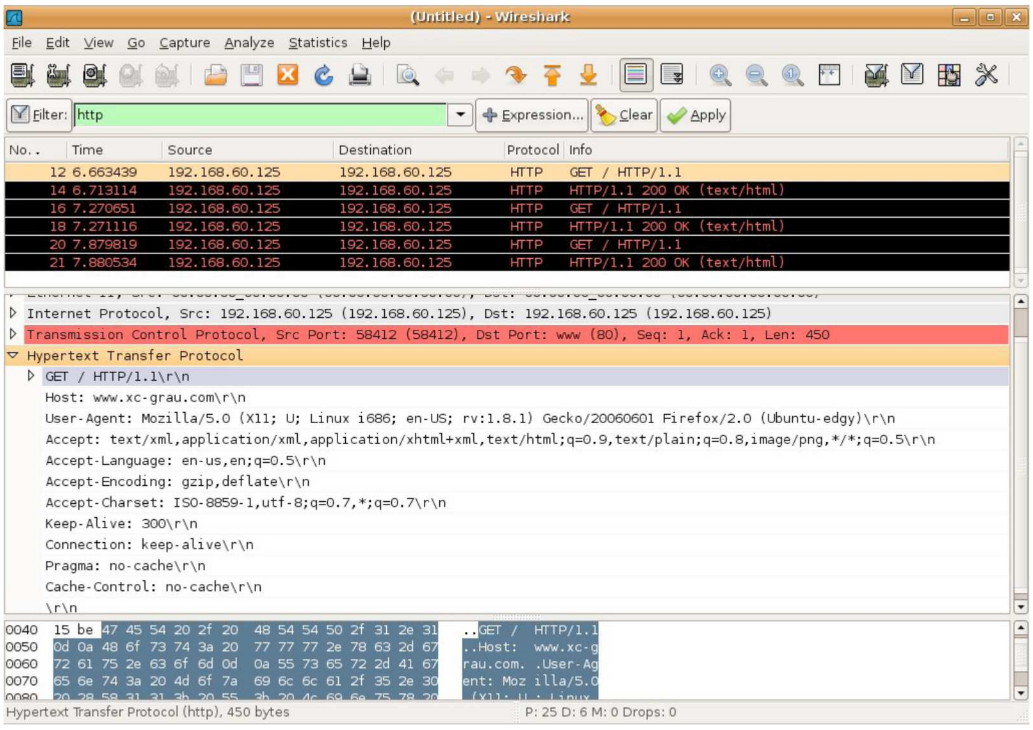


Wireshark: Capture Interfaces

Device	Description	IP	Packets	Packets/s		
					Stop	
e0		192.168.60.125	1042	48	Capture	Prepare
any	Pseudo-device that captures on all interfaces	unknown				
lo		127.0.0.1				

Figure 36: Capture with Wireshark.

Once the capture has been made, we can filter the messages of a certain protocol, as showed in Figure 37. A really important characteristic is the capacity of being able to select the packets that we want to capture. This is done with a text box placed next to Filter. Here we can enter an expression like "dns" to capture messages that carry application level information relative to the http protocol, or more sophisticated expressions like udp.port==53, to capture the UDP segments that have as a destination or source port the port 53.



```
No. . Time       Source          Destination     Protocol Info
   12 6.663439   192.168.60.125  192.168.60.125  HTTP     GET / HTTP/1.1
   14 6.713114   192.168.60.125  192.168.60.125  HTTP     HTTP/1.1 200 OK (text/html)
   16 7.270651   192.168.60.125  192.168.60.125  HTTP     GET / HTTP/1.1
   18 7.271116   192.168.60.125  192.168.60.125  HTTP     HTTP/1.1 200 OK (text/html)
   20 7.879819   192.168.60.125  192.168.60.125  HTTP     GET / HTTP/1.1
   21 7.880534   192.168.60.125  192.168.60.125  HTTP     HTTP/1.1 200 OK (text/html)
```

```
▷ Internet Protocol, Src: 192.168.60.125 (192.168.60.125), Dst: 192.168.60.125 (192.168.60.125)
▷ Transmission Control Protocol, Src Port: 58412 (58412), Dst Port: www (80), Seq: 1, Ack: 1, Len: 450
▽ Hypertext Transfer Protocol
  ▷ GET / HTTP/1.1\r\n
    Host: www.xc-grau.com\r\n
    User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1) Gecko/20060601 Firefox/2.0 (Ubuntu-edgy)\r\n
    Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5\r\n
    Accept-Language: en-us,en;q=0.5\r\n
    Accept-Encoding: gzip,deflate\r\n
    Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
    Keep-Alive: 300\r\n
    Connection: keep-alive\r\n
    Pragma: no-cache\r\n
    Cache-Control: no-cache\r\n
    \r\n
```

```
0040  15 be 47 45 54 20 2f 20  48 54 54 50 2f 31 2e 31   ..GET /  HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20  77 77 77 2e 78 63 2d 67   ..Host:  www.xc-g
0060  72 61 75 2e 63 6f 6d 0d  0a 55 73 65 72 2d 41 67   rau.com. .User-Ag
0070  65 6e 74 3a 20 4d 6f 7a  69 6c 6c 61 2f 35 2e 30   ent: Moz illa/5.0
```

```
Hypertext Transfer Protocol (http), 450 bytes          P: 25 D: 6 M: 0 Drops: 0
```

Figure 37: Filtering the http messages with Wireshark.

3.2. `nslookup` command

Command to interact with a Name Server. It is called as:

```
nslookup
```

Next we have an extract of the man page:

INTERACTIVE COMMANDS

host: Look up information for host. If host is an Internet address and the query type is A or PTR, the name of the host is returned. If host is a name and does not have a trailing period, the search list is used to qualify the name. To look up a host not in the current domain, append a period to the name.

exit: Exits the program.

set keyword[=value]: This command is used to change state information that affects the lookups. Valid keywords are:

> **all**: Prints the current values of the frequently used options to set. Information about the current default server and host is also printed.
>
> **domain=name**: Sets the search list to name.
>
> **[no]search**: If the lookup request contains at least one period but doesn't end with a trailing period, append the domain names in the domain search list to the request until an answer is received.
>
> **type=value**: Change the type of the information query. (Default = A; abbreviations = q, ty)
>
> **[no]recurse**: Tell the name server to query other servers if it does not have the information. (Default = recurse; abbreviation = [no]rec)
>
> **[no]debug**: Turn on or off the display of the full response packet and any intermediate response packets when searching. (Default = nodebug; abbreviation = [no]deb).

For example, to ask the DNS server's address: www.xc-grau.test, assuming that xc-grau.test is in /etc/resolv.conf, we would execute:

```
# nslookup
> www
Server: 192.168.60.125
Address: 192.168.60.125#53
www.xc.test canonical name = pcserver.xc.test.
pcserver.xc.test canonical name = pc125.xc.test.
Name: pc125.xc.test
Address: 192.168.60.125
```

To request a RR of type MX:

```
# nslookup
> set type=MX
> upc.edu
Server: 192.168.60.125
Address: 192.168.60.125#53
Non-authoritative answer:
upc.edu mail exchanger = 10 mx1.upc.es.
```

To request again a RR of type A:

```
# nslookup
> set type=A
...
```

3.3. The file resolv.conf

In the file “/etc/resolv.conf” there is the IP address of the local nameserver. For instance, if the address is 192.168.60.125:

```
root@aula01:/# cat /etc/resolv.conf
search xc.test
nameserver 192.168.60.125
```

where “nameserver” is the local nameserver IP address and “search” is de default domain. If a resolution fails and there are several nameservers (several “nameserver” lines), these are requested sequentially. The default domain is added is the name to solve is not fully qualified.

3.4. The file /etc/bind/db.root

In the file “/etc/bind/db.root” there are the root-servers RRs. These are used to access the DNS tree. This file must not be modified. It is part of the bind installation.

3.5. The zone file of the DNS server

The zone file has the resource records (RR) of the zone. For instance, the file db.grupX.xc in the lab has the information shown in the following figure. Note that this file is a template of a zone file, where the X must be changed to the corresponding value, as explained below. The RR type SOA (Start Of Authority) has configuration parameters. Then there RRs of type NS (nameserver), A (address), CNAME (alias) and MX (Mail eXchange).

```
@       IN      SOA     ns.grupX.xc.test hostmaster.grupX.xc.test (
                              1         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
        IN      NS      ns.grupX.xc.test.
        IN      MX      10 mail1.grupX.xc.test.
        IN      MX      20 mail2.grupX.xc.test.
;
ns.grupX.xc.test.    A       192.168.60.X ;Adreça IP del NS
mail1.grupX.xc.test. A       192.168.60.X ;Adreça IP del MX
mail2.grupX.xc.test. A       192.168.60.X ;Adreça IP del MX
www.grupX.xc.test.   CNAME   pcserver.grupX.xc.test.
smtp.grupX.xc.test.  CNAME   pcserver.grupX.xc.test.
pop3.grupX.xc.test.  CNAME   pcserver.grupX.xc.test.
pcserver.grupX.xc.test.      CNAME   pcX.grupX.xc.test.
pcX.grupX.xc.test.        A       192.168.60.X ;Adreça IP de PCX
```

4. Practice realisation

The objective of this practice is creating a subzone (grupX.xc.test) inside a hypothetical existing domain (xc.test), as shown in Figure 38. From now on X will be the number of the PCs that it is used as name server. In a real situation the authority xc.test will point to grupX.xc.test, and thus, it would be accessible from a root-server. Note in our case xc.test does not exist, therefore, the names of the zone cannot be resolved from the rest of the Internet.

To realise the practice correctly, each group will have to use 2 PCs. One that will act as a DNS server of the subzone grupX.xc.test, and another one that will act as host from this subzone (pxX.grupX.xc.test) to make requests, in the way shown in the following figure.

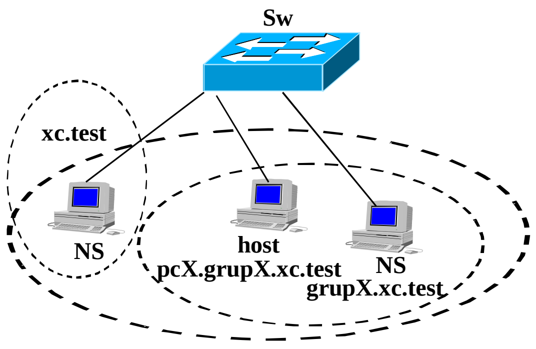


Figure 38: Network to configure. Only the host pcX.grupX.xc.test and the NS grupX.xc.test must be configured.

4.1. Network configuration

1) Configure the 2 PCs of Figure 38 (host pcX.grupX.xc and name server grupX.xc.test) executing the dhcp client (udhcpc). Then run "killall udhcpc" in pcX.grupX.xc.test to kill the dhcp client, otherwise it will periodically update the resolv.conf file, modifying the changes made to this file. Then, change the file /etc/resolv.conf of pcX.grupX.xc.test such that the DNS server grupX.xc.test is being used by default:

```
search grupX.xc.test
nameserver 192.168.60.X
```

where the X in the IP address and grupX.xc.test is the number of the PC used as local name server.

4.2. Configuration of the DNS server of the subzone

The configuration files are in the folder /home/xc/dns, which will be the folder where we will work from now on.

2) Edit the file ‘named.conf’ to reflect the real name of the subdomain in zone file:

```
zone "grupX.xc.test" IN {
  type master;
  file "/root/lab1/db.grupX.xc";
};
```

Where you will have to change X in `grupX.xc.test` by the number of the PC used as name server.

3) Edit the file db.grupX.xc and change the required occurrences of "X" by the number of the PC used as name server.
4) Start the name server executing 'run_named.sh'. The script has to be restarted whenever named.conf or db.grupX.xc is changed. Check that the server is running executing "ps aux | egrep named". Check that no errors occurred executing "tail –f /var/log/messages".

4.3. Observation of the DNS protocol behaviour

5) Using the 'nslookup' tool, answer the following questions. Which registers have you consulted in each occasion? (nslookup allows to change the type of request with the command 'set type=(A,NS,MX,CNAME...)'). assume that grupX and pcX are the ones corresponding to your group):
   - 'www.grupX.xc.test' is the alias of which machine?
   - Which one is the IP address of the server 'www. grupX.xc.test'?
   - Which are the mail servers of the domain 'grupX.xc.test'?
6) Turn on the wireshark in the machine that acts as DNS server of your zone to capture the DNS traffic that is generated, and observe what happens with the previous petitions. Navigate through the packet fields to answer the following:
   - How many messages are generated in each resolution?
   - Is it a recursive or iterative resolution?
   - Identify the addresses of all requested DNS servers. Is a root-server used?
   - Investigate the contents of the fields (question, answer, authority, additional) of all DNS reply messages.
7) Capture the DNS traffic generated upon solving the name www.microsoft.com. Navigate through the packet fields to answer the following:
   - How many messages are generated in each resolution?
   - Is it a recursive or iterative resolution?
   - Identify the addresses of all requested DNS servers. Is a root-server used?
   - Investigate the contents of the fields (question, answer, authority, additional) of all DNS reply messages.
   - How many IP addresses correspond to the name? What are their canonical names?
8) Activate debug mode in nslookup (set debug). Repeat the resolution of www.microsoft.com. Compare the information provided by nslookup with the content of the captured messages.
9) Change the mode of your client (nslookup) to non-recursive (using the command 'set norecurse'). What changes when repeating the steps resolution of www.microsoft.com?
10) Try to solve the name configured by your colleagues in another group. What messages are generated? Is the resolution possible? Why?
11) Open the web browser and wireshark in the host. Connect to www.fib.upc.edu. Given that the PCs in the lab use a Proxy-web to access the Internet, are there any name resolutions? Why?

5. Previous report

1. Which files will be necessary to modify in the machine that acts as a DNS server of your subzone in the lab?
2. What are the recursive and iterative modes of the DNS?
3. How many messages and what content is expected upon solving the name www.grupX.xc.test?
4. How many messages and what content is expected upon solving the name www.microsoft.com?