Network administration

Remo Suppi Boldrito

PID_00148471

Universitat Oberta de Catalunya
www.uoc.edu

Copyright © 2009, FUOC. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License"

Index

Introduction

The UNIX (GNU/Linux) operating system is used as an example of a standard communications architecture. From the mythical UUCP (Unix-to-Unix CoPy or service for copying between UNIX operating systems) to the current networks, UNIX has always proven its versatility in aspects related to communication and information exchange. With the introduction of computer networks (Local Area Networks, Wide Area Networks or the latest Metropolitan Area Networks) offering multipoint connections at different speeds (from 56 kbits/sec to 1 Gbit/sec), new services that are based on faster protocols, portable between different computers and better adapted, such as TCP/IP (*transport control program / Internet protocol*), have arisen. [Com01, Mal96, Cis00, Gar98, KD00]

1. Introduction to TCP/IP (TCP/IP suite)

The TCP/IP protocol synthesises an example of a will to communicate and to standardise the communication on a global scale.

The TCP/IP is, in reality, a set of basic protocols that have been added to the original protocol, to meet the different needs in computer-to-computer communication, such as TCP, UDP, IP, ICMP, ARP. [Mal96]

TCP/IP is most frequently used by most current users to remotely connect to other computers (telnet, SSH Secure Shell), to use remote files (NFS *network file system*) or to transfer them (FTP *file transfer protocol*, HTTP *hypertext markup protocol*).

Note

Typical use of TCP/IP remote login:
`telnet localhost` Debian GNU/Linux 4.0
login:

1.1. Services on TCP/IP

The most important traditional TCP/IP services are [Gar98]:

- **File transfer**: the *file transfer protocol* (FTP) allows the user of a computer to obtain files or send them from one computer to another. In order to do this, the user must have an account in the remote computer and identify themselves through their login name and password or the user must connect to computers containing an information repository (software, documentation etc.) under an anonymous account to read those computers on their computer. This is not the same as the more recent Network File Systems (NFS) (or netbios protocols over TCP/IP, a completely insecure "invention" in Windows, which should be replaced with an older but more secure version called netbeui) that make it possible to virtualise the file system in a machine so that it can be accessed interactively from another computer.

- **Remote connection** (login): the terminal network protocol (telnet) allows a user to remotely connect to a computer. The local computer is used as the remote computer's terminal and everything is executed over it, whilst the local computer remains invisible from the perspective of the user that started the session. This service has now been replaced by the SSH (*secure shell*), for security reasons. This can use a remote connection through telnet and the messages are sent as plain text; in other words, if someone "examines" the messages on the network, it is equivalent to looking at the user's screen. SSH encrypts the information (which is an added-value to

the communication) so that the packages on the network cannot be read by any foreign node.

- **Email**: this service makes it possible to send messages to users of other computers. This form of communication has become an essential element for users and allows email messages to be sent to a central server, so that they can then be recovered using specific programs (clients) or read through an internet connection.

The progress in the technology and the increasingly lower cost of computers has meant that determined services have specialised and are now configured on determined computers working in a client-server model. A server is a system that performs specific services for the rest of the network or connected clients. A client is another computer that uses this service. All of these services are generally offered within TCP/IP:

- **File systems in network file systems**: allows a system to access the files through a remote system in a manner that is more integrated than FTP. The storage devices (or part of them) are exported to the system that wishes to access the files and this system can "see" them as if they were local devices. This protocol permits in the server side to establish the rules and ways of accessing the files, which (if properly configured) makes the place where the information physically resides independent from the place where the information is "accessed".

- **Remote printing**: permits users to access printers connected to other computers.

- **Remote execution**: permits a user to execute a program on another computer. There are various ways of executing a program in this way: either through a command (rsh, ssh, rexec) or through systems with RPC (*remote procedure call*), which allows a program on a local computer to execute a function in a program on another computer. The RPC processes have been studied in-depth and there are various implementations, but the most common are Xerox's Courier and Sun's RPC (the latter has been adopted in most UNIX systems).

- **Name servers**: in large-scale networks of computers, there are data that have to be centralised so that they can be easily used; for example, user names, passwords, internet addresses etc. All of this makes it easier for a user to have an account for all the machines in an organisation. For example, Sun's Yellow Pages (NIS in the current *Sun* versions) is designed to handle all these types of data and it is available for most UNIX systems. The DNS (*domain name system*) is another domain-name service but one that keeps a direct relationship between the hostname and the logical identification name of this machine (IP address).

- **Terminal Servers**: connect terminals to a server that executes telnet so as to connect to the central computer. These types of setup are basically useful for reducing costs and improving the connections to the central computer (in some cases).

- **Graphical terminal servers** (*network-oriented window systems*): these permit a computer to visualise graphic information on a display that is connected to another computer. The most common of these systems is X Window.

1.2. What is TCP/IP?

TCP/IP is in fact two communication protocols between computers that are independent to each other.

On the one hand, TCP (*transmission control protocol*) defines the communication rules so that a (host) computer can talk to another computer (if we use the OSI/ISO communications model as a reference, it describes layer 4, see following table).

TCP is a connection-oriented protocol, in other words, it is equivalent to a telephone, and the communication is considered as a data stream.

IP (*Internet protocol*) defines the protocol to identify the networks and establish the pathways between different computers.

In other words, it routes the data between two computers through the networks. It corresponds to layer 3 of the OSI/ISO model and it is a connectionless protocol (see following table). [Com01, Rid00, Dra99]

An alternative to TCP is the UDP protocol (*user datagram protocol*), which treats the data as a message (datagram) and sends packets. It is a connectionless protocol (the recipient computer does not necessarily have to be listening when the other computer establishes communication with it) and it has the advantage of creating less overload on the network than a TCP connection, but it is less reliable (the packets may not arrive or arrive duplicated).

There is another alternative protocol called ICMP (*Internet control message protocol*). ICMP is used for error or control messages. For example, if one tries to connect to a host computer, the local computer may receive an ICMP message indicating "host unreachable". ICMP may also be used to extract information on a network. ICMP is similar to UDP in that it handles messages (datagrams),

but it is simpler than UPD, because it does not have port identification (the ports are mailboxes where the data packets are left and where the server applications read the packets) in the message header.

In the OSI/ISO communications model (OSI, *open systems interconnection reference model*, ISO, *International Standards Organization*), is a theoretical model applied by many networks.There are seven communication layers where each one has an interface for communicating with the preceding and following one.

Level	Name	Use
7	Application	SMTP, *simple mail transfer protocol*, the service itself
6	Introduction	Telnet, FTP implements the service protocol
5	Session	Generally not used
4	Transport	TCP, UDP transformation in accordance with the communication protocol.
3	Network	IP makes it possible to route the packet.
2	Link	Drivers - transformation in accordance with the physical protocol.
1	Physical	Ethernet, ADSL... physically sends the packet

To summarise, TCP/IP is a set of protocols including IP, TCP, UDP that provide a set of low-level functions used by most of the applications. [KD00, Dra99].

Some of the protocols that use the abovementioned services were designed by Berkeley, Sun or other organisations. They are not imcluded (officially) as part of the *Internet protocol suite* (IPS). However, they are implemented using TCP/IP and they are therefore considered as a formal part of IPS. A description of the protocols available by Internet can be found in RFC 1011 (see references on RFC [IET]). There is currently a new version of protocol IPv6, also called IPng (IP *next generation*) which replaces IPv4. This protocol significantly improves the previous ones in elements such as having a greater number of nodes, traffic control, security or improvements in the routing.

1.3. Physical network devices (hardware)

From the physical point of view (layer 1 of the OSI model), the most commonly used hardware for LAN is that known as Ethernet (or FastEthernet or GigaEthernet). Its advantages consist of a lower cost, acceptable speeds (10, 100 or 1,000 megabits per second) and its user-friendly installation.

There are three connection modes, depending on the type of interconnection: thick, thin and twisted pair.

The first two are obsolete (they used coaxial cable) whereas the last is through twisted pair cables and connectors similar to those used by telephones (known as RJ45). The twisted pair connection is known as 10baseT or 100baseT (according to the speed) and it uses repeaters known as hubs as interconnection points. Ethernet technology uses intermediate communication elements (hubs, switches, routers) to configure multiple segments of the network and divide the traffic to improve the performance of the data transfer. Normally, in large organisations, these Ethernet LAN are interconnected through fibre optic cables using FDDI (*fibre distributed data interface*) technology, which is more expensive and more difficult to install, but with which we can obtain transmission speeds equivalent to Ethernet whilst not having the limits on distance involved in Ethernet (FDDI allows for distances of up to 200 km). The costs are justified when they are used between buildings or other network segments that are very congested. [Rid00, KD00].

At the same time, there are other types of hardware that are less common, but no less interesting, such as ATM (*asynchronous transfer mode*). This hardware allows us to set up a LAN with a high level of service quality and it is a good option when we have to set up high-speed and low-latency networks, such as those that require real time video streaming.

There is other hardware supported by GNU/Linux for interconnecting computers, of which we would mention: Frame Relay or X.25 (used in computers that access or interconnect WANs and for servers with large data transfer needs), Packet Radio (interconnection via radio using protocols such as AX.25, NetRom or Rose) or dial-up devices that use serial lines, which are slow but very cheap, through analogical or digital (RDSI, DSL, ADSL etc.) modems. The latter are the ones commonly used domestically or in small and medium-sized businesses, and they require another protocol for the transmission of packets, such as SLIP or PPP. In order to virtualise the diverse hardware on a network, TCP/IP defines an abstract interface through which all the packets that will be sent by a physical device (which includes a network or network segment) are concentrated. Consequently, for each communication device in the machine, we will have a corresponding interface in the operating system's kernel.

Example

In GNU/Linux, Ethernet is called with ethx (where, "x" indicates an order number beginning with 0), the interface to serial lines (modems) is called up with pppx (for PPP) or slx (for SLIP); fddix is used for FDDI. These names are used by the commands to configure them and assign them the identification that will subsequently permit them to communicate with other devices in the network.

In GNU/Linux, this may mean that we have to include the appropriate modules for the appropriate device (NIC *network interface card*) in the kernel or as modules, and this means compiling the kernel after choosing, the appropriate NIC, with, for example, *make menuconfig*, indicating it as internal or as a module (in the latter case, the appropriate module must also be compiled).

The network devices can be seen in the /dev directory, where there is a file (a special file, which may be a block file or a character file, according to the transfer) that represents each hardware device.[KD00, Dra99].

Note

How do we see the network interfaces that are available?

`ifconfig -a`

This command shows all of the default interfaces/parameters for each one.

2. TCP/IP Concepts

As we have observed, communication involves a series of concepts that we will now discuss [Mal96, Com01]:

- **Internet/intranet**: the term *intranet* refers to the application of Internet technology (the network of networks) within an organisation, basically to distribute the company's internal information and to have it available within the company. For example, the services offered by GNU/Linux as Internet and Intranet services include email, WWW, news etc.

- **Node**: the (host) node refers to a machine that is connected to the network (in a wider sense, a node may be a computer, a printer, a CD (rack) etc.); in other words, an active and differentiable element in the network that requires or provides some kind of service and/or shares information.

- **Ethernet Network Address** (*Ethernet address* or *MAC address*): a 48-bit number (for example 00:88:40:73:AB:FF –in octal– 0000 0000 1000 1000 0100 0000 0111 0011 1010 1011 1111 1111 –in binary–) that is inside the physical device (hardware) of the Ethernet driver (NIC) and that is recorded by the manufacturer (this number must be the only one in the world, each NIC manufacturer has a pre-allocated range).

Note

Name of the machine:
`more /etc/hostname`

- **Host name**: each node must also have a unique network name. These may simply be names or they may use a scheme based on a hierarchical domain naming scheme. The names of the nodes must be unique, which is easy in small networks, more complex in large networks and impossible on the Internet unless some form of control is implemented. The names must have a maximum of 32 characters within the a-z, A-Z and 0-9 ranges and they may not contain spaces or # beginning with an alphabetic character.

Note

Machine IP address:
`more /etc/hosts`

- **Internet Address** (*IP address*): this consists of four numbers within the range of 0-255 separated by dots (for example, 192.168.0.1) and it is used universally to identify the computers on a network or on the Internet. The names are translated into IP addresses by a DNS (*domain name system*) server, that transforms the node names (legible to humans) in IP addresses (this service is performed by an application called *named*).

Note

Pre-assigned ports in UNIX:
`more /etc/services`
This command shows the ports predefined with support to TCP or UDP communications.

- **Port**: numerical identifier of the mailbox in a node that allows a specific application to read a message (TCP,UDP) (for example, two machines that communicate by telnet, will do so through port 23, but if they have a FTP transaction they will do so through port 21). There may be different ap-

Note

Visualisation of the routing's configuration:
`netstat -r`

plications communicating between two nodes through various different ports simultaneously.

- **Router node** (*gateway*): it is a node that performs the routing (data transfer). A router, depending on its characteristics, may transfer information between two similar or different network protocols and may also be selective.

Note

Domain and our DNS server is:

```
more /etc/default domain
more /etc/resolv.conf
```

- ***Domain name system*** (DNS): makes it possible to ensure one single name and to provide the administration of the databases that perform the translation between the name and Internet address and that are structured in the form of a tree. In order to do this, domains separated by points are defined, of which the highest (from right to left) describes a category, institution or country (COM stands for Commercial, EDU for Education, GOV for Governmental, MIL for Military (government), ORG, non-profit Organisation, XX which could be any two letters to indicate the country, or special cases, such as CAT to indicate Catalan language and culture etc.). The second level represents the organisation and the third and remaining sections indicate the departments, sections or divisions within an organisation (for example, www.uoc.edu or nteum@pirulo.remix.es). The first two names (from right to left), *uoc.edu* in the first case, *remix.es* (in the second) must be assigned (approved) by the SRI-NIC (global organisation that manages the Internet domain registry) and the rest may be configured/assigned by the institution.

Note

arp tables:

```
arp to NameNode
```

- **DHCP, bootp**: DHCP and bootp are protocols that permit a client node to obtain information on the network (such as the node's IP address). Many organisations with many machines use this mechanism to facilitate the administration of large networks or networks in which there are roaming users.

- **ARP, RARP**: in some networks (such as IEEE 802 LAN, which is the standard for Ethernet), the IP addresses are dynamically discovered through the use of two other members of the Internet protocol suite: *address resolution protocol* (ARP) and *reverse address resolution protocol* (RARP). ARP uses broadcast messages to determine the Ethernet address (MAC specification for layer 3 of the OSI model), corresponding to a particular network-layer address (IP). RARP uses broadcast messages (messages that reach all of the nodes) to determine the network-layer address associated with a particular hardware address. RARP is especially important to diskless nodes, for which network-layer addresses are usually unknown at boot time.

- **Socket Library**: in UNIX, all TCP/IP implementation is part of the kernel of the operating system (either within the same or as a module that loads at boot time, as is the case with the device drivers in GNU/Linux).

The way for a programmer to use them is through an API (*application programming interface*) which implements this source-code interface. For TCP/IP, the most common API is the Berkeley Socket Library (Windows uses an equivalent library that is called Winsocks). This library makes it possible to create a communication end-point (socket), associate it to a remote node and port (bind) and offer the communication service (through *connect, listen, accept, send, sendto, recv, recvfrom*, for example). The library also provides a more general communication mode (AF INET family) and more optimised communications for cases in which the process are communicating within the same machine (AF UNIX family). In GNU/Linux, the socket library is part of the C standard library, Libc, (Libc6 in current versions), and it supports AF_INET, AF_UNIX, AF_IPX (for Novell protocols), AF_X25 (for the X.25 protocol), AF_ATMPVC-AF_ATMSVC (for the ATM protocol) and AF_AX25,F_NETROM, AF_ROSE (for *amateur radio protocol*).

3. How to assign an Internet address

This address is assigned by the NIC and it has two section or parts. The one on the left represents network identification and the one on the right represents the node identification. In consideration of the point mentioned above (four numbers between 0-255, or 32 bits or four bytes), each byte represents either the network or the node. The NIC assigns the net and the institution (or provider) assigns the node.

There are some restrictions: **0** (for example, 0.0.0.0) in the network space is reserved for the routing by default and **127** (for example, 127.0.0.1) is reserved for the (local loopback or local host), **0** in the node part refers to this network (for example,192.168.0.0) and **255** is reserved for sending packets to all (broadcast) machines (for example, 198.162.255.255). There may be different types of networks or addresses in the different assignations:

Class A (*network.host.host.host*): 1.0.0.1 to 126.254.254.254 (126 networks, 16 million nodes) define the large networks. The binary standard is: **0** + 7 network bits + 24 node bits.

Class B (*network.network.host.host*): 128.1.0.1 to 191.255.254.254 (16K networks, 65K nodes); (usually, the first node byte is used to identify subnets within an institution). The binary standard is **10** + 14 network bits + 16 node bits.

Class C (*net.net.net.host*): 192.1.1.1 to 223.255.255.254 (2 million of networks, 254 nodes). The binary standard is **110** + 21 network bits + 8 node bits.

Classes D and E (*network.network.network.host*): 224.1.1.1 to 255.255.255.254 reserved for *multicast* (from one node to a set of nodes that form part of the group) and experimental purposes.

Some address ranges have been reserved so that they do not correspond to public networks, and are considered private networks (interconnected computers without external connection; the messages will not be sent through Internet, but through an intranet). These address ranges are **class A** 10.0.0.0 to 10.255.255.255, **class B** 172.16.0.0 to 172.31.0.0 and **class C** 192.168.0.0 to 192.168.255.0.

The broadcast address is special, because each node in a network listens to all the messages (as well as its own address). This address makes it possible to send datagrams (generally routing information and warning messages) to a

network and all nodes on the network will be able to read them. For example, when ARP tries to find the Ethernet address corresponding to an IP, it uses a broadcast message, which is sent to all the machines on the network at the same time. Each node in the network reads this message and compares the IP that is being searched and sends back a message to the sender node if they match.

Two concepts that are related to the point described above are the **subnets and routing** between these subnets. **Subnets** subdivide the node part into smaller networks within the same network, so as to, for example, improve the traffic. A subnet is in charge of sending traffic to certain IP address ranges, extending to the same concept of Class A, B and C networks, but only applying this rerouting in the IP node part. The number of bits interpreted as a subnet identifier is provided by a netmask, which is a 32-bit number (as is an IP). In order to obtain the subnet identifier, we will have to perform a logical AND operation between the mask and the IP, which will provide us with the subnet IP. For example, an institution with a B class network, with number 172.17.0.0, would therefore have a netmask with number 255.255.0.0. Internally, this network is formed by small networks (one per floor in the building, for example). In this way, the range of addresses is reassigned in 20 subnets (floors in our example, except 172.17.1.0, that has a special role), 172.17.1.0 to 172.17.20.0. The point that connects all these floors, called the backbone, has its own address, for example 172.17.1.0.

These subnets share the same network IP, whereas the third is used to identify each of the subnets within it (which is why it will use the netmask 255.255.255.0).

The second concept, **routing**, represents the mode in which the messages are sent through the subnets. For example, let us say there are three departments with Ethernet subnets:

1) Purchases (subnet 172.17.2.0),
2) Clients (subnet 172.17.4.0),
3) Human Resources, (subnet 172.17.6.0)
4) Backbone with FFDI (subnet 172.17.1.0).

In order to route the messages between the computers on the three networks, we need three gateways that will each have two network interfaces to switch between Ethernet and FFDI. These would be:

1) PurchasesGW IPs:172.17.2.1 and 172.17.1.1,
2) ClientsGW IPs:172.17.4.1 and 172.17.1.2
3) HumanResourcesGW IPs:172.17.6.1 and 172.17.1.3, in other words, one IP on the subnet side and another on the backbone side.

When messages are sent between machines in the purchases area, it is not necessary to leave the gateway, as the TCP/IP will find the machine directly. The problem arises when the Purchases0 machine wishes to send a message to HumanResources3. The message must pass through the two respective gateways. When Purchases0 "sees" that HumanResources3 is on another network, it sends the packet through the PurchasesGW gateway, which in turn sends it to HumanResourcesGW, which, in turn, sends it to HumanResources3. The advantage of having subnets is obvious, given that the traffic between all the purchases machines, for example, will not affect the Clients or Human Resources machines (although this is more complex and expensive in terms of designing and building the network).

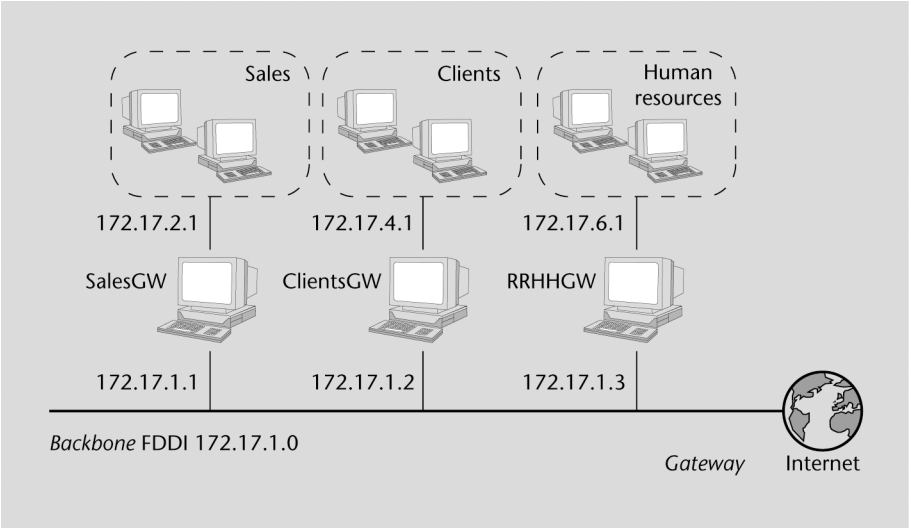

Figure 1. Configuration of segments and gateways in an intranet

IP uses a table to route the packets between the different networks, in which there is a default routing associated to net 0.0.0.0. All the addresses coincide with this one, as none of the 32 bits are necessary; they are sent through the default gateway to the indicated network. In the purchasesGW, for example, the table would be:

Address	Mask	Gateway	Interface
172.17.1.0	255.255.255.0	-	fddi0
172.17.4.0	255.255.255.0	172.17.1.2	fddi0
172.17.6.0	255.255.255.0	172.17.1.3	fddi0
0.0.0.0	0.0.0.0	172.17.2.1	fddi0
172.17.2.0	255.255.255.0	-	eth0

The '-' means that the machine is directly connected and does not need routing. The procedure for identifying whether routing is required or not consists of performing a very simple operation with the two logic ANDs (subnet AND

mask and origin AND mask) and comparing the two results. If they match, there is no routing, but the machine defined as gateway must be sent in each machine, so that this machine routes the message.

For example, a message from 172.17.2.4 to 172.17.2.6 would mean:

172.17.2.4 AND 255.255.255.0 = 172.17.2.0
172.17.2.6 AND 255.255.255.0 = 172.17.2.0

As the results are the same, there would be no routing. On the other hand, if we do the same from 172.17.2.4 to 172.17.6.6 we see that there will be routing through 172.17.2.1 with an interface change (*eth0* to *ffdi0*) to 172.17.1.1 and from here to 172.17.1.2 with another interface change (*fddi0* to *eth0*) and then to 172.17.6.6. The default routing will be used when none of the rules match. If two rules match, the routing that matches the most precisely, in other words, the one with the least zeros, will be used. In order to build the routing tables, we can use the *route* command during machine startup; however, if it is necessary to use more complex rules (or automatic routing), we can use the *routing information protocol* (RIP) command or, between independent systems, the *external gateway protocol* (EGP) or also the *border gateway protocol* (BGP) commands. These protocols are implemented through the *gated* command.

In order to install a machine on an existing network, it is necessary to have the following information, obtained from the network provider or the administrator: node IP address, network IP address, broadcast address, netmask address, router address and DNS address.

If we are setting up a network that will never have an Internet connection, we can choose the addresses that we wish, but it is advisable to maintain an appropriate order corresponding to the size of the network that will be needed, so as to avoid administrative problems within the network in question. We will now see how to define the network and node for a private network (we have to be careful, as, if the machine is connected to the network, we can inconvenience another user to whom this address has been assigned): node address 192.168.110.23, netmask 255.255.255.0, net part 192.168.110., node part .23, net address 192.168.110.0, broadcast address 192.168.110.255.

4. How to configure the network

4.1. Configuration of the network interface controller (NIC)

Once the GNU/Linux kernel has loaded, it executes the init command, which, in turn, reads the configuration file /etc/inittab and begins the start up process. Generally, the inittab has sequences such as: si::sysinit: /etc/init.d/boot, which represents the name of the commands file (script) that controls the booting sequences. Generally, this script calls the other scripts, which include the network startup script.

Example

In Debian, etc/init.d/network is executed to configure the network interface, depending on the boot level; For example, in boot level 2, all the S* files in directory /etc/rc2.d (which are links to the /etc/initd directory) will execute, and on the boot down level, all the K* files in the same directory. In this way, the script is only there once (/etc/init.d) and, depending on the services required in that status, a link is created in the directory corresponding to the node-status.

The network devices are created automatically when the corresponding hardware starts up. For example, the Ethernet driver creates the eth[0..n] interfaces sequentially, when the corresponding hardware is located.

Note

Consult
`man ifconfig`
for the different command options.

The network interface may be configured as of that moment, which requires two steps: assign the network address to the device and boot the network parameters to the system. The command used for this is ifconfig (*interface configure*). An example might be:

```
ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up
```

Which indicates that the eth0 device should be configured with IP address 192.168.110.23 and netmask 255.255.255.0. *Up* indicates that the interface will be activated (to deactivate it, execute ifconfig eth0 down). If no values are specified, the command assumes that the default values should be used. In the previous example the kernel will configure this machine as a C-Type machine with IP=192.168.110.23 and the broadcast address=192.168.110.255.

There are commands, such as ifup and ifdown, that make it possible to configure/unconfigure the network more simply using the /etc/network/interfaces file to obtain all the necessary parameters (consult *man interfaces* for syntax).

In Debian, there is another simpler method for configuring the network (considered high-level), which uses the abovementioned commands ifup, ifdownand the */etc/network/interfaces* file. If we decide to use these commands, we should **not** configure the network at low-level, as these commands are sufficient for configuring/unconfiguring the network.

In order to modify the parameters of the eth0 interface network, we can (consult *man interfaces* in section 5 of the Unix manual included with the operating system for more information):

ifdown eth0	for all network services over eth0
vi /etc/network/interfaces	edit and modify networks/interfaces parameters
ifup eth0	start up the network services over eth0

Let us suppose that we wish to configure an eth0 interface in Debian, which has a fixed IP address 192.168.0.123 and has 192.168.0.1 as the gateway. We must edit /etc/network/interfaces so that it includes a section such as:

```
iface eth0 inet static
        address 192.168.0.123
        netmask 255.255.255.0
        gateway 192.168.0.1
```

If we have installed the resolvconf packet, we can add lines to specify the DNS information. For example:

```
iface eth0 inet static
        address 192.168.0.123
        netmask 255.255.255.0

        gateway 192.168.0.1
        dns-search remix.org
        dns-nameservers 195.238.2.21 195.238.2.22
```

After the interface has been activated, the command line arguments of the options dns-search and dns-nameservers are available for resolvconf for inclusion in resolv.conf. The command line argument remix.org of the dns-search option corresponds to the argument of the search option in resolv.conf (we will look at this in more detail later) and the arguments 195.238.2.21 and 195.238.2.22 of the dns-nameservers option corresponds to the arguments of the nameserver options in resolv.conf (consult man resolv.conf). It is also possible to configure the network at low-level through the ip command (which is equivalent to ifconfig and route). Although this command is much more versatile and powerful (it can be used to establish tunnels, alternate routings etc.), it is more complex and it is recommendable to use the preceding procedures for basic network configurations.

4.1.1. Configuration of network in Fedora style

Red Hat and Fedora use a different file structure for network configuration: /etc/sysconfig/network. For example, to configure the network statically:

NETWORKING=yes HOSTNAME=*my-hostname* FORWARD_IPV4=true GATEWAY=*"XXX.XXX.XXX.YYY"*	*Name of the host defined by the cmd hostname* *True for NAT firewall gateways and routers.* *False for any other case* *Gateway leading out to Internet*

To configure using DHCP, it is necessary to delete the GATEWAY line, as it will be assigned by the server. And if NIS is to be incorporated, a line with the server domain must be added: NISDOMAIN=NISProject1

To configure interface eth0 in the file

/etc/sysconfig/network-scripts/ifcfg-eth0:

DEVICE=eth0
BOOTPROTO=static
BROADCAST=XXX.XXX.XXX.255
IPADDR=*XXX.XXX.XXX.XXX*
NETMASK=255.255.255.0
NETWORK=*XXX.XXX.XXX.0*
ONBOOT=yes Activates the network on boot.

From FC3 on, it is also possible to add:
TYPE=Ethernet
HWADDR=*XX:XX:XX:XX:XX:XX*
GATEWAY=*XXX.XXX.XXX.XXX*
IPV6INIT=no
USERCTL=no
PEERDNS=yes

Or else, for configuring using DHCP :

DEVICE=eth0
ONBOOT=yes
BOOTPROTO=dhcp

To disable DCHP, change BOOTPROTO=dhcp to BOOTPROTO=none. Any change in these files must restart the services with service network restart (or, otherwise, /etc/init.d/network restart).

The following three steps must be taken to change the hostname:

1) Command `hostname new-name`.

2) Change the network configuration in /etc/sysconfig/network editing HOSTNAME=new-name.

3) Restoring all the services (or *rebooting*):

- `service network restart` (or executing /etc/init.d/network restart)
- Restarting the desktop by passing into console mode `init 3` and changing to GUI mode `init 5`.

Verifying if the name is not registered in /etc/hosts. The hostname may be changed during execution time with `sysctl` -w kernel.hostname="new-name".

4.1.2. Configuration of a Wi-Fi (wireless) network

In order to configure Wi-Fi interfaces, we basically use the wireless-tools package (as well as ifconfig or ip). This package uses the iwconfig command to configure a wireless interface, but this can also be carried out through /etc/network/interfaces.

Example: Configure WiFi in Debian Sarge (Etch) (similar in FC6)

Let's assume that we wish to configure an Intel Pro/Wireless 2200BG wireless network card (very common in many laptops, such as Dell, HP...). The software that controls the cards is usually divided into two parts: the software module that will be loaded in the kernel through the modprobe command and the firmware that is the code that will be loaded in the card and which is given to us by the manufacturer (consult the Intel site for this model). As we are discussing modules, it is interesting to use the Debian module-assistant package which allows us to create and install a module easily (another option would be to install the sources and create the corresponding module). We will compile and install the software (which we can find on the manufacturers' website and is called ipw2200) using the m-a command in the module-assistant package.

```
aptget install module-assistant (install the package)
m-a -t update
m-a -t -f get ipw2200
m-a -t -build ipw2200
m-a -t install ipw2200
```

We can download the compatible firmware version from the site address provided by the manufacturer (in the product documentation) along with the version of the driver we need, which in our case, would be driver version 1.8 and firmware version 2.0.4, obtained from the following address:

http://ipw2200.sourceforge.net/firmware.php

We should then decompress and install the firmware:

```
tar xzvf ipw2200fw2.4.tgz C /tmp/fwr/
cp /tmp/fwr/*.fw /usr/lib/hotplug/firmware/
```

This will copy three packages (ipw2200-bss.fw, ipw2200-ibss.fw and ipw2200-sniffer.fw). The module is then loaded with: modprobe ipw2200, the system reboots and then, from the console, we can execute the dmesg | grep ipw command, which will show us some lines similar to the ones below and which indicate that the module is loading (this can be checked with lsmod):

```
ipw2200: Intel(R) PRO/Wireless 2200/2915 Network Driver, git1.0.8
ipw2200: Detected Intel PRO/Wireless 2200BG Network Connection
...
```

We should then download the wireless tools package that contains iwconfig in order to install wireless tools with aptget, among others, and if we execute iwconfig, something similar to the following will display:

```
eth1 IEEE 802.11b ESSID:"Name-of-the-Wifi"
Mode:Managed Frequency:2.437 GHz
Access Point:00:0E:38:84:C8:72
Bit Rate=11 Mb/s TxPower=20 dBm
Security mode:open
...
```

We must then configure the network file, for example, gedit /etc/network/interfaces, and add the eth1 wifi interface, for example:

```
iface eth1 inet dhcp
     pre-up iwconfig eth1 essid "Name of the Wifi"
     pre-up iwconfig eth1 key open XXXXXXXXXX
```

The pre-up lines execute the iwconfig command before activating the interface. This configuration is used if we wish to use the service in DHCP mode (automatic IP assignation, as we shall see). Instead of DHCP, the word static should be used and the following lines, as an example, must be entered (as in a cable card):

```
address 192.168.1.132
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.1.1
```

Another method for configuring the interface is:

```
iface eth1 inet dhcp
     wireless-essid "Name of the Wifi"
     wireless-key 123456789e
```

We can then start up the network with ifup eth1 and we will be given information on the connection and the state and quality of reception. In order to scan the available WiFi networks (access points), we can use iwlist scan, which will show us information on the available networks, and if we want to connect to a different network, we can use the iwconfig command to change the network or Access Point.

4.2. Configuration of Name Resolver

The next step is to configure the name resolver, which changes names like pirulo.remix.com to 192.168.110.23. The /etc/resolv.conf file is used for this. The format is very simple (one line of text per sentence). There are three key words for this purpose: *domain* (local domain), *search* (list of alternate domains) and *name server* (IP address of the *domain name server*).

Example of /etc/resolv.conf

```
domain remix.com
search remix.com piru.com
name server 192.168.110.1
name server 192.168.110.65
```

This list of name servers often depends on the network environment, which may change depending on where the machine is or where it is connected. The programs for connecting to telephone lines (`pppd`) or obtaining IP addresses automatically (`dhclient`) can modify `resolv.conf` to insert or delete servers; but these characteristics do not always work properly and they can sometimes generate conflicts or incorrect configurations. The `resolvconf` package adequately solves the problem and allows us to configure the name servers easily and dynamically. `resolvconf` is designed to work without the user having to configure anything manually; however, the package is quite new and may require some manual assistance to make it work properly. For more information:

http://packages.debian.org/unstable/net/resolvconf

Another important file is */etc/host.conf*, which can be used to configure the behaviour of the name resolver. This file is very important because it indicates where the node address or name is first resolved. This can be consulted in the DNS server or the local tables within the existing machine (/etc/hosts).

Example of /etc/host.conf

```
order hosts,bind
multi on
```

This configuration indicates that /etc/hosts should be verified first consulting the DNS and it also indicates (2nd line) that all valid addresses found in /etc/hosts should be returned. Consequently, the /etc/hosts file is where the local addresses are placed and it can also be used to access the nodes without having to consult the DNS.

The consulting process is much faster, but the disadvantage is that, if the node changes, the address will be incorrect. In a system that is properly configured, only the local node and an input for the loopback interface should appear.

Example of /etc/hosts

```
127.0.0.1          localhost                  loopback
192.168.1.2        pirulo.remix.com           pirulo
```

Aliases may be used for the name of a machine; this means that this machine may have different names for the same IP address. The loopback interface is a special type of interface that makes it possible for a node to connect to itself (for example, to verify that the network subsystem is working without accessing the net). By default, the IP address 127.0.0.1 has specifically been assigned to the loopback (a `telnet 127.0.0.1` command will connect with the same machine). Configuring aliases is very easy (generally, the network startup script configures them).

Example of loopback

```
ifconfig lo 127.0.0.1
route add host 127.0.0.1 lo
```

In version 2 of the GNU library, there is an important replacement with regard to the functions of the host.conf. file. This improvement includes the centralisation of information on different services for name resolution, which provides many advantages for the network administrator. All the information on name and service consultations has been centralised in the /etc/nsswitch.conf file, which allows the administrator to configure the order and the databases in a very simple manner. In this file, each service appears, one per line, with a set of options, such as the node name resolution option. This indicates that the order for consulting the databases for obtaining the node's IP or its name will be first through the DNS service (which uses the /etc/resolv.conf file to determine the IP of the DNS node) and then, if it cannot be obtained here, the databases of the local (/etc/*hosts*) will be used. Other options for this could be nis or nisplus, which are other information services that are explained in subsequent units. The method for each consultation may also be controlled through actions (between []), for example:

```
hosts: xfn nisplus dns [NOTFOUND = return] files
```

This indicates that, when the DNS is consulted, if there is no registry for this consultation, the program that made the consultation will return a zero. The '!' may be used to deny the action, for example:

```
hosts dns [!UNAVAIL = return] files
```

Note

Example of nsswitch.conf: ...
hosts: dns files
...
networks: files

4.3. Configuration of routing

Another aspect that has to be configured is the routing. Although the process is considered to be very complex, in general, the routing requirements are very simple. In a node with multiple connections, routing consists of deciding where to send and what to receive. A simple node (one single network connection) also needs routing, given that all the nodes have a loopback and a network connection (for example, Ethernet, PPP, SLIP...). As we have explained, there is a table known as a routing table that contains rows with various fields, three of which are especially important: destination address, interface through which the message will be sent and IP address, which will take the next step in the gateway.

Note

Consultation of routing tables:
`route -n`
or also
`netstat -r`

The route command can be used to modify this table so as to carry out the appropriate routing tasks. When a message arrives, the destination address is examined, compared with the entries in the table and sent through the interface with the address that most resembles the packet's destination. If a gateway is specified, it is sent to the appropriate interface.

Let us assume, for example, that our node is in a C class network with the address 192.168.110.0 and the address is 192.168.110.23; and the router connected to the Internet is 192.168.110.3. The configuration will be:

- First, the interface:
  ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up

- Subsequently, indicate that all the datagrams for nodes with 192.168.0.* addresses must be sent to the network device:
  route add -net 192.1 ethernetmask 255.255.255.0 eth0

-net indicates that it is a network route but *-host* 192.168.110.3. may also be used. This configuration will allow it to connect with all the nodes within a network segment (192.1), but, what would happen if we wanted to connect with another node outside this segment? It would be very difficult to have all the appropriate entries for all the machines to which we wish to connect. To simplify this task, we have the *default route*, which is used when the destination address does not match any of the entries in the table. One configuration possibility would be:

route add default gw 192.168.110.3 eth0

(the gw is the IP or name of a gateway or router node).

Another method of doing this would be:

```
ifconfig eth0 inet down disable the interface
ifconfig lo
          Link encap:Local Loopback
... (no entries for eth0 will appear)
route
... (no entry in the routing table will appear)
```

Subsequently, the interface is enabled with another IP and a new route:

```
ifconfig eth0 inet up 192.168.0.111 \
          netmask 255.255.0.0 broadcast 192.168.255.255
route add -net 10.0.0.0 netmask 255.0.0.0 \
          gw 192.168.0.1 dev eth0
```

The bar (\) indicates that the command continues on the following line. The result:

```
ifconfig
      eth0 Link encap:Ethernet HWaddr 08:00:46:7A:02:B0
              inet addr:192.168.0.111 Bcast: 192.168.255.255 Mask:255.255.0.0
              UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
              ...
      lo Link encap:Local Loopback
              inet addr:127.0.0.1 Mask:255.0.0.0
              ...
 route
    Kernel IP routing table
        Destination     Gateway      Genmask      Flags  Metric Ref Use Iface
        192.168.0.0     *            255.255.0.0  U      0      0   0   eth0
        10.0.0.0        192.168.0.1  255.0.0.0    UG     0      0   0   eth0
```

For more information, see the ifconfig (8) and route (8) commands.

4.4. Configuration of inetd

The next step in the configuration of the network is to configure the servers and services that will allow another user to access the local machine or its services. The server programs will use the ports to listen to the requests from the clients, which will be sent to this service as IP:port. The servers may work in two different ways: standalone (in which the service listens to the assigned port and is always active) or through inetd.

The inetd is a server that controls and manages the network connections of the services specified in the /etc/inetd.conf file, which, when a service request is made, starts up the appropriate server and transfers the request.

Two important files must be configured: `/etc/services` and `/etc/inetd.conf`. In the first file, we associate the services, the ports and the protocol, and in the second, the server programs that will respond to a request to a determined port. The /etc/services format is `name port/protocol aliases`, where the first field is the service name, the second is the port where the service is attended and the protocol that it uses, and the next field is an alias of the name. There is a series of default pre-configured services. We will now show an example of /etc/services (# indicates that what follows is a comment):

tcpmux	1/tcp #	# TCP port service multiplexer
echo	7/tcp	
echo	7/udp	
discard	9/tcp	sink null
discard	9/udp sink	sink null
systat	11/tcp	users
...		
ftp	21/tcp	
ssh	22/tcp	# SSH Remote Login Protocol
ssh	22/udp	# SSH Remote Login Protocol
telnet	23/tcp	
# 24 - private		
smtp	25/tcp	mail
...		

The /etc/inetd.conf file is the configuration for the master network service (*inetd server daemon*). Each line contains seven fields separated by spaces: *service socket_type proto flags user server_path server_args,* where *service* is the service described in the first column in /etc/services, *socket_type* is the type of socket (possible values are *stream, dgram, raw, rdm*, or *seqpacket*), *proto* is the protocol that is valid for this input (it must match that in /etc/services), *flags* indicates the action that should be taken when there is a new connection on a service that is attending another connection, (*wait* tells *inetd* not to start up a new server or *nowait* means that inetd must start up a new server). *user* will be the local user-name with which the client that has started up the service is identified, *server_path* is the directory where the server is located and *server_args*

are possible arguments that will be passed to the server. An example of some /etc/inetd.conf lines is (# is a comment, so if a service has # before the name, it means that it is not available):

```
...
telnet stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.telnetd
ftp stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.ftpd
# fsp dgram udp wait root /usr/sbin/tcpd /usr/sbin/in.fspd
shell stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rshd
login stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rlogind
# exec stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.rexecd...
...
```

As of Debian Woody 3.0 r1, the inetd function has been replaced by xinetd (recommendable), which needs the `/etc/xinetd.conf` configuration file (see end of unit). If we wish to start up the *inetd* service, we must execute (and create the appropriate links in the `/etc/rcX.d directories`) `/etc/init.d/inetd.real start` (see the end of this chapter for examples of configurations).

Apart from the inetd or xinetd configuration, the typical configuration of network services in a desktop or basic server environment might also include (some of these services will be examined in the chapter on servers):

- **ssh:** secure interactive connection to replace telnet that includes two configuration files `/etc/ssh/ssh_config` (for the client) and `/etc/ssh/sshd_config` (for the server)

- **exim:** multi transfer agent (MTA), includes configuration files: `/etc/exim/exim.conf`, `/etc/mailname`, `/etc/aliases`, `/etc/email-addresses`.

- **fetchmail:** daemon for downloading the mail from a POP3 account, `/etc/fetchmailrc`

- **procmail:** program for filtering and distributing local mail, ~/.procmailrc

- **tcpd:** Filtering services for enabled and disabled machines and domains for connecting to the server (wrappers): `/etc/hosts.allow`, `/etc/hosts.deny`

- **DHCP.** Service for managing (server) or obtaining an IP (client), `/etc/dhcp3/dhclient.conf` (client), `/etc/default/dhcp3-server` (server), `/etc/dhcp3/dhcpd.conf` (server)

- **CVS**: system for managing concurrent versions, `/etc/cvs-cron.conf`, `/etc/cvs-pserver.conf`

- **NFS**: network file system, /etc/exports

- **Samba**: network file system and sharing printers in Windows networks, /etc/samba/smb.conf

- **lpr**: daemon for the printing system, `/etc/printcap` (for the Ipr system -not CUPS-)

- **Apache** and **Apache2**: Web Server, `/etc/apache/*` and `/etc/apache2/*`

- **squid:** Server proxy-cache, `/etc/squid/*`

4.5. Additional configuration: protocols and networks

There are other configuration files that are hardly ever used, but that can be interesting. The /etc/protocols is a file that shows the protocol identifiers with the protocol names; in this way, the programmers can specify the protocols by their names in the programs.

Example of /etc/protocols

ip	0	IP	# internet protocol, pseudo protocol number
#hopopt	0	HOPOPT	# IPv6 Hop-by-Hop Option [RFC1883]
icmp	1	ICMP	# internet control message protocol

The /etc/networks file has a function similar to /etc/hosts, but where the networks are concerned, it shows the network names in relation to its IP address (the route command will show the name of the network and not its address in this case).

Example of /etc/networks

loopnet 127.0.0.0
localnet 192.168.0.0
amprnet 44.0.0.0 ...

4.6. Security aspects

It is important to take into account the security aspects in network connections, as a significant amount of attacks occur through the network. We will discuss this subject in more detail in the unit on security; however, there are

some basic recommendations that should be taken into account in order to minimise the risks immediately before and after configuring the network in our computer:

- Do not activate services in /etc/inetd.conf that will not be used, insert an # before the name to avoid sources of risk.

- Modify the /etc/ftpusers file to deny access to certain users who may have an FTP connection to your machine.

- Modify the /etc/securetty file to indicate from which terminals (a name per line), for example: tty1 tty2 tty3 tty4, it will be possible for the root superuser to connect. The root superuser will not be able to connect from any of the remaining terminals.

- Use the tcpd program. This server is a wrapper that makes it possible to allow/deny a service from a given node and it is placed in /etc/inetd.conf as a service intermediary. The tcpd verifies certain access rules in two files: `/etc/hosts.allow /etc/host.deny`.

If the connection is accepted, it starts up an appropriate service passed as an argument (for example, the FTP service line shown earlier in inetd.conf:

ftp stream tcp nowait *root*`/usr/sbin/tcpd/usr/sbin/in.ftpd`.

tcpd first search /etc/hosts.allow and then inside of `/etc/hosts.deny`. The hosts.deny file contains the rules on which nodes do not have access to a service within this machine. A restrictive configuration is ALL: ALL, as it will only allow access to the services from the nodes declared in `/etc/hosts.allow`.

The `/etc/hosts.equiv` file permits access to this machine without having to enter the password. Using this mechanism is not recommended; users should be advised not to use the equivalent from the user account, through the .rhosts file.

In Debian, it is important to configure /etc/security/access.conf, the file that indicates the rules on who and from where it is possible to log in to this machine. This file has a line by command with three fields separated by a ':' of the permission type: Users: origin. The first will be an +o- (allow or deny), the second a user name/user names, group or user@host, and the third will be the name of a device, node, domain, node or networks addresses or ALL.

Example of access.conf

This command does not permit root logins over *tty1*:

ALL EXCEPT root:tty1 ...

It permits access to *u1, u2, g1* and all those in the remix.com domain:

+:u1 u2 g1 .remix.com:ALL

4.7. IP Options

There are further options with regard to IP traffic that we should mention. This is configured by starting up the corresponding file in the `/proc/sys/net/ipv4/` directory. The file name is the same as the command and a 1 must be placed in the file to activate them, or a 0 to deactivate them.

Example

For example, if we wish to activate ip_forward, we have to execute:

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

The most widely used are: ip_forward used for routing between interfaces or with IP Masquerading; ip_default_ttl, which is the lifetime for an IP packet (64 milliseconds, by default) ip_bootp_agent logical variable (BOOLEAN) which accepts packets (or not) with the origin address of the 0.b.c.d type and the destination of this node, broadcast or multicast.

4.7.1. Commands for solving problems with the network

If there are problems in the configuration of the network, we can begin by verifying the output of the following commands to obtain an initial idea:

```
ifconfig
cat /proc/pci
cat /proc/interrupts
dmesg | more
```

In order to verify the network connection, we can use the following commands (netkit-ping, traceroute, dnsutils, iptables and net-tools must be installed):

```
ping uoc.edu                    # verifies the Internet connection
traceroute uoc.edu              # scans IP packets
ifconfig                        # verifies the host configuration
route -n                        # verifies the routing configuration
dig [@dns.uoc.edu] www.uoc.edu  # verifies the registries in
                                # on the dns.uoc.edu server.
iptables -L -n |less            # verifies packet filtering (kernel >=2.4)
netstat -a                      # shows all the open ports
```

```
netstat -l --inet               # shows all the listening ports
netstat -ln --tcp               # shoos the listening tcp ports (number)
```

5. DHCP Configuration

DHCP stands for dynamic host configuration protocol. It is very simple to configure and it is useful because, instead of having to configure each node in a network individually, this can be done in a centralised manner and administering it is therefore easier. The configuration of a client is very easy, as we only have to install one of the following packages: `dhcp3-client` (version 3, Internet Software Consortium), dhcpcd (Yoichi Hariguchi and Sergei Viznyuk), pump (Red Hat); we then add the word *dhcp* in the section corresponding to the interface that we wish to work under this dhcp client (e.g./etc/network/interfaces must have iface eth0 inet dhcp...).

Configuring the server requires more care, but it is not especially complicated. First, for the server to serve all the DHCP clients (including Windows), we must address some questions concerning the broadcast addresses. In order to do this, first the server must be able to send messages to the 255.255.255.255 address, which is not valid in GNU/Linux. In order to try this, execute:

route add -host 255.255.255.255 dev eth0

If the following message appears: *255.255.255.255: Unknown host*, then the following entry must be added in /etc/hosts: *255.255.255.255 dhcp* and try again:

route add -host dhcp dev eth0

The configuration of dhcpd can be carried out with the graphic interface of linuxconf (not advisable) or by editing /etc/dhcpd.conf. An example of this file is:

```
# Example of /etc/dhcpd.conf:
default-lease-time 1200;
max-lease-time 9200;
option domain-name "remix.com";
deny unknown-clients;
deny bootp;
option broadcast-address 192.168.11.255;
option routers 192.168.11.254;
option domain-name-servers 192.168.11.1,192.168.168.11.2;
subnet 192.168.11.0 netmask 255.255.255.0
{ not authoritative;
          range 192.168.11.1 192.168.11.254
          host marte {
                    hardware ethernet 00:00:95:C7:06:4C;
```

```
                    fixed address 192.168.11.146;
                    option host-name "marte";
                    }
          host saturno {
                    hardware ethernet 00:00:95:C7:06:44;
                    fixed address 192.168.11.147;
                    option host-name "saturno";
              }
    }
```

This will allow the server to assign the address range from 192.168.11.1 to 192.168.11.254, as described for each node. If the corresponding host { ... } segment does not exist, they will be assigned randomly. The IPs are assigned for a minimum time of 1,200 seconds and a maximum of 9,200 (if these parameters do not exist, they will be assigned indefinitely).

Before executing the server, we must verify if the file `/var/state/dhcp/dhcpd.leases` exists (otherwise, it will have to be created with touch `/var/state/dhcp/dhcpd.leases`). To execute the server: /usr/sbin/dhcpd (or we can put it in the startup scripts). With `/usr/sbin/dhcpd -d-f`, we can see the activity in the server within the system's console. [Mou01, Rid00, KD00, Dra99]

It is important not to forget the `not authoritative` phrase, as, otherwise, this server may leave other dhcp servers that serve IP for other segments inactive.

6. IP aliasing

There are some applications in which it is useful to configure multiple IP addresses to a single network device. The ISPs (*Internet service providers*) frequently use this characteristic to provide personalised features (such as World Wide Web and FTP) to their users. For this, the kernel must be compiled with the Network Aliasing and IP (aliasing support) options. After installing the new kernel, the configuration is very easy. The aliases are attached to the virtual network devices associated with the new device with a format such as: device: virtual number.

For example: eth0:0, ppp0:8

Let us say that we have an Ethernet network that supports two different IP subnets simultaneously and that our machine wants to have direct access to them. An example of the configuration would be:

ifconfig eth0 192.168.110.23 netmask 255.255.255.0 up
route add -net 192.168.110.0 netmask 255.255.255.0 eth0
ifconfig eth0:0 192.168.10.23 netmask 255.255.255.0 up
route add -net 192.168.10.0 netmask 255.255.255.0 eth0:0

Which means that we would have two IPs, 192.168.110.23 and 192.168.10.23 for the same NIC. In order to delete an alias, add a '-' at the end of the name (for example, ifconfig eth0:0- 0). [Mou01, Ran05]

A typical case is when we wish to configure a single Ethernet card so that it acts as the interface for different IP subnets. For example, suppose we have a machine that is on a LAN network, LAN 192.168.0.x/24. And we wish to connect the machine to the Internet using a public IP address provided with DHCP using the existing Ethernet card. For example, we can follow the procedure described in the preceding example or edit the /etc/network/interfaces file so that it includes a section similar to the following:

```
iface eth0 inet static
address 192.168.0.1
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255

iface eth0:0 inet dhcp
```

The eth0:0 interface is a virtual interface and its parent interface, eth0, will activate when it does.

7. IP Masquerade

The IP Masquerade is a resource used so that a set of machines may use a single IP address. This permits the hidden nodes (in other words, the ones that use a private IP, such as 198.162.10.1) can go out to the Internet; but they cannot directly accept external calls or services; only through the machine that has the real IP.

This means that some services will not work (for example, talk) and others must be configured in PASV (passive) mode for them to work (for example, FTP). However, WWW, telnet or IRC will work properly. The kernel must be configured with the following options: Network firewalls, TCP/IP networking, IP: forwarding/gatewaying, IP: masquerading. Normally, the most common configuration is to have a machine with a SLIP or PPP connection and to have another network device (for example, an Ethernet card) with a reserved net address. As we have seen and as described in RFC 1918, the following address ranges (IP/Mask) can be used as private IPs: 10.0.0.0/255.0.0.0, 172.16.0.0/255.240.0.0, 192.168.0.0/255.255.0.0. The nodes that must be masqueraded will be on this second network. Each of these machines must have the address of the machine that is masquerading such as default gateway or router. On this machine, we can configure:

- Network route for Ethernet considering that the network has a IP = 192.168.1.0/255.255.255.0:
  `route add -net 192.168.1.0 netmask 255.255.255.0 eth0`

- Default route for the rest of Internet:
  `route add default ppp0`

- All the nodes over the 192.168.1/24 network will be masqueraded:
  `ipchains -A forward -s 192.168.1.0/24 -j MASQ`

- If iptables are used over a kernel, version 2.4 or higher:
  `iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE`

Consult the references in the unit covering security for information on *ipchains* and *iptables*. [Ran05, KD00]

8. NAT with kernel 2.2 or higher

The IP *network address translation*, NAT, is a replacement that has made the features of GNU/Linux IP Masquerade obsolete and that provides new features to the service. One of the improvements included in the TCP/IP stack of GNU/Linux 2.2 is that NAT is integrated into the kernel. In order to use it, we have to compile the kernel with:

CONFIG_IP_ADVANCED_ROUTER, CONFIG_IP_MULTIPLE_TABLES and CONFIG_IP_ROUTE_NAT.

And if we need comprehensive control of the NAT rules (for example, to activate the firewall we must also have

CONFIG_IP_FIREWALL and CONFIG_IP_ROUTE_FWMARK.

In order to work with these new features, we need to use the ip program (which can be obtained at ftp://ftp.inr.ac.ru/ip_routing/). Then, to translate the incoming datagram addresses, we can use:

```
ip route add nat <extaddr>[/<masklen>] via <intaddr>
```

This will translate the destination address of an incoming packet addressed to ext-addr (the address that is visible externally from Internet) to int-addr (the address of the internal network through the gateway/firewall). The packet is routed in accordance with the local route table. Single or block addresses can be translated. For example:

```
ip route add nat 240.0.11.34 via 192.109.0.2
ip route add nat 240.0.11.32/27 via 192.109.0.0
```

The first makes the internal address 192.109.0.2 accessible as 240.0.11.34. The second remaps the 192.109.0.0/31 block to 240.0.11.32/63. In this case, we have used, as an example, translations to class D and E addresses, such as 240.0.*.* so as not to use a public address. The user must replace these addresses (240.0.11.34 and 240.0.11.3263) for the corresponding public addresses to which they wish to translate. [Ran05]

9. How to configure a DialUP and PPP connection

Configuring a dial-up connection using PPP in GNU/Linux is very simple. PPP (*point to point protocol*) makes it possible to establish *IP-Links* between two computers with a modem (that it must be a modem supported by GNU/Linux, as not all modems, especially internal ones or those known as Winmodems, can be configured because many of them need additional software in order to establish communication). [Vas00, Law07, Sec00].

To start with, we must have the following information: the modem init-string (this is not normally necessary but if it is and it is not available, we can use ATZ, which works in most modems or we can consult specialised init-string lists).

We also need the ISP data: connection ID (login name), password and telephone number. The DNS addresses would be advisable, but this is optional in the current versions of pppd. Also, we should verify that the modem is connected properly. With an external modem, we must execute *echo > /dev/ttyS0* and check the LEDs on the modem to see if it is active. Otherwise, try with ttyS1, if the modem is connected to the 2nd serial port. With an internal modem, check the supported hardware manual to see if this modem can be recognised by GNU/Linux; if this is the case, it may be necessary to reconfigure the kernel in order to use it. We can also use *cat /proc/pci* in case it is in the PCI bus. [PPP00]

The easiest way to configure the modem now is through the kppp package (we must install the kdenetwork-ppp* and ppp* packages). On a terminal, execute /usr/bin/kppp. On the window, fill in the following boxes:

Accounts ⇒ New Connection
Dial ⇒ Authentication 'PAP/CHAP'
Store Password ⇒ yes
IP ⇒ Dynamic IP Address
Autoconfigure hostname ⇒ No
Gateway ⇒Default *Gateway* ⇒ Assign the Default Route
DNS ⇒Configuration Automatic ⇒Disable existing DNS
Device ⇒ttyS1(com1) o ttyS2 (com2)
Modem ⇒Query Modem to see the results (if you do not obtain the results, change the ttySx device).

After entering the login name and password, we will be connected to the Internet (to check that we are connected, execute *ping www.google.com* for example). Here, we have used the kppp package, but we could as easily have used linuxconf or gnomeppp indistinctly).

A quick way of configuring pppd in Debian consists of using the pppconfig program, which comes with the package. *pppconfig* configures files such as the preceding ones after asking the user some questions through the menu interface. Another option for using *pppd* consists of executing it from *wvdial*, which comes with the *wvdial* package. Instead of making *pppd* execute chat to dial and negotiate the connection, *wvdial* dials, carries out the initial negotiation and then starts up *pppd* so that it can do the rest. In most cases, with just the telephone number, username and password, *wvdial* can start the connection.

Once PPP has been configured, for it to work with, for example, my_isp, we must edit /etc/network/interfaces so that it includes a section such as the following (the *ifup*, *ifdown* commands use the *pon* and *poff* commands to configure PPP interfaces):

```
iface ppp0 inet ppp
provider mi_isp
```

with this section, ifup ppp0 executes:

```
pon my_isp
```

It is not currently possible to use ifup down to perform a supporting configuration of the PPP interfaces. As pon disappears before pppd has finished establishing the connection, ifup executes the up scripts before the PPP interface is ready to be used. Until this fault is resolved, it will still be necessary to configure the connection later in /etc/ppp/ip-up or /etc/ppp/ip-up.d/.

Many broadband Internet Service Providers (ISP) use PPP to negotiate the connection even when the clients' machines are connected through Ethernet and/or ATM networks. This is achieved through PPP over Ethernet (PPPoE) which is a technique for encapsulating PPP flow within Ethernet frames. Suppose that the ISP is called my_isp. First, we must configure PPP and PPPoE for *my _isp*. The easiest way of doing this consists of installing the *pppoeconf* package and executing *pppoeconf* from the console. We then edit /etc/network/interfaces so that it includes a fragment such as the following:

```
iface eth0 inet ppp
provider mi_isp
```

Sometimes, problems arise with PPPoE that are related to the *maximum transmit unit* (or MTU) in DSL (*digital subscriber line*) lines; you may consult DSL-HOWTO for further details. If the modem has a router, as the modem/router will handle the PPPoE connection on its own and it will appear on the LAN side as a simple Ethernet to Internet gateway.

10. Configuring the network through *hotplug*

The *hotplug* package supports hot swapping when booting (the package in question must have been installed). The network hardware can be hot plugged either at start up, after inserting the card in the machine (a PCMCIA card, for example) or after a utility such as discover has been executed and the necessary modules have been loaded. When the kernel detects new hardware, it starts up the driver for the hardware and then executes the hotplug program to configure it. If the hardware is subsequently removed, the program executes hotplug again, with different parameters. In Debian, when hotplug is called, this executes the scripts of `/etc/hotplug/` and `/etc/hotplug.d/`. The network hardware that was recently connected is configured by `/etc/hotplug/net.agent`. Let us assume that the PCMCIA network card has been connected, which would mean that the eth0 interface would be ready to be used. `/etc/hotplug/net.agent` performs the following:

```
ifup eth0=hotplug
```

Unless a logical interface called hotplug has been added in `/etc/network/interfaces`, this command will have no effect. For this command to configure eth0, we have to add the following lines to `/etc/network/interfaces`:

```
mapping hotplug
script echo
```

If you only want eth0 to hotplug and not other interfaces, use grep instead of echo as follows:

```
mapping hotplug
script grep
map eth0
```

ifplugd activates or deactivates an interface depending on whether the underlying hardware is connected to the network or not. The program can detect a cable connected to an Ethernet interface or an access point associated to a Wi-Fi interface. When ifplugd sees that the status of the connection has changed, it will execute a script, which, by default, executes ifup or ifdown for the interface. ifplugd works in combination with hotplug. When a card is inserted, which means that the interface is ready to be used, `/etc/hotplug.d/net/ifplugd.hotplug` starts up an instance of ifplugd for that interface. When ifplugd detects that the card is connected to a network, it executes ifup for this interface.

In order to associate a Wi-Fi card with an access point, we may have to program it with an appropriate WEP encryption code. If ifplugd is being used to control ifup, as we have explained, then evidently it will not be able to con-

figure the encryption code using ifup, as this is only called once the card has been associated. The simplest solution is to use waproamd, which configures the WEP encryption code according to the available access points that are discovered through a WiFi network search. For more information, consult man waproamd and the information on the package.

11. Virtual private network (VPN)

A VPN (*virtual private network*) is a network that uses Internet to transport data, but stops any external members from accessing that data.

This means that we have a network with connected VPN nodes tunnelled through another network, through which the traffic passes and with which no one can interact. It is used when remote users wish to access a corporate network to maintain the security and privacy of the data. Various methods can be used to configure a VPN, such as SSH (SSL), CIPE, IPSec, PPTP; they can be consulted in the bibliography (we recommend consulting VPN PPP-SSH HOWTO, by Scott Bronson and VPN-HOWTO by Matthew D. Wilson). [Bro01, Wil02].

In order to perform the configuration tests in this section, we will use OpenVPN, which is a solution based on SSL VPN and can be used for a wide range of solutions, for example, remote access, VPN point to point, secure WiFi networks or distributed corporate networks. OpenVPN implements OSI layer 2 or 3 using SSL/TLS protocols and supports authentication based on certificates, smart cards and other confirmation methods. OpenVPN is not a proxy applications server and does not operate through a web browser.

In order to analyse it, we will use an option in OpenVPN called OpenVPN for Static key configurations, which provides a simple method for configuring a VPN that is ideal for tests or point-to-point connections. The advantages are the simplicity and the fact that it is not necessary to have a X509 public key infrastructure (PKI) certificate to maintain the VPN. The disadvantages are that it only permits one client and one server, as, because the public key and private key are not used, there may be the same keys as in previous sessions and there must be a text-mode key in each peer and the secret key must be previously exchanged for a secure channel.

11.1. Simple example

In this example, we will configure a VPN tunnel on a server with IP=10.8.0.1 and a client with IP=10.8.0.2. The communication will be encrypted between the client and server on a UDP port 1194, which is the default port in OpenVPN. After installing the package (http://openvpn.net/install.html), we must generate the static key:

```
openvpn --genkey --secret static.key
```

Then, we must copy the static.key file in the other peer over a secure channel (using ssh or scp, for example). The server configuration file of the openVPN_server for example:

```
dev tun
ifconfig 10.8.0.1 10.8.0.2
secret static.key
```

The client configuration file for example openVPN_client

```
remote myremote.mydomain
dev tun
ifconfig 10.8.0.2 10.8.0.1
secret static.key
```

Before verifying that the VPN works, we must verify the firewall to check that port 1194 UDP is open on a server and that the virtual interface tun0 used by OpenVPN is not blocked either over the client or over the server. Bear in mind that 90% of the connection problems faced by new OpenVPN users are related in some way to the firewall.

In order to verify the OpenVPN between two machines, we must change the IPs for the real ones and the domain for the corresponding one, and then execute the server side.

```
openvpn [server config file]
```

Which will provide an output such as:

```
Sun Feb 6 20:46:38 2005 OpenVPN 2.0_rc12 i686-suse-linux [SSL]
[LZO] [EPOLL] built on Feb 5 2005
Sun Feb 6 20:46:38 2005 Diffie-Hellman initialized with 1024
bit key
Sun Feb 6 20:46:38 2005 TLS-Auth MTU parms [ L:1542 D:138
EF:38 EB:0 ET:0 EL:0 ]
Sun Feb 6 20:46:38 2005 TUN/TAP device tun1 opened
Sun Feb 6 20:46:38 2005 /sbin/ifconfig tun1 10.8.0.1 pointo-
point 10.8.0.2 mtu 1500
Sun Feb 6 20:46:38 2005 /sbin/route add -net 10.8.0.0 netmask
255.255.255.0 gw 10.8.0.2
Sun Feb 6 20:46:38 2005 Data Channel MTU parms [ L:1542
D:1450 EF:42 EB:23 ET:0 EL:0 AF:3/1 ]
Sun Feb 6 20:46:38 2005 UDPv4 link local (bound): [undef]:1194
Sun Feb 6 20:46:38 2005 UDPv4 link remote: [undef]
Sun Feb 6 20:46:38 2005 MULTI: multi_init called, r=256 v=256
Sun Feb 6 20:46:38 2005 IFCONFIG POOL: base=10.8.0.4 size=62
Sun Feb 6 20:46:38 2005 IFCONFIG POOL LIST
```

```
Sun Feb 6 20:46:38 2005 Initialization Sequence Completed
```

And the client side:

```
openvpn [client config file]
```

In order to check that it works, we might ping 10.8.0.2 from the server and `ping 10.8.0.1` from the client. For more information, please check http://openvpn.net/howto.html.

To add compression to the link, we must add the following line to the two configuration files:

```
comp-lzo
```

In order to protect the connection through a NAT router/firewall alive and carry on the IP changes through a DNS, if one of the peers changes, add the following to the two configuration files:

```
keng-timer-rem
persist-tun
peepalive 10 60
pirsist-key
```

To execute as a daemon with the privileges of the nobody user/group, add the following to the configuration files:

```
user nobody
group nobody
Daemon
```

12. Advanced configurations and tools

There is a set of additional packages (that replace the conventional ones) and tools that either improve the machine's security (recommended in hostile environments) or help to configure the network (and the system in general) in a more user-friendly style.

These packages may be of great help to the network administrator for avoiding intrusions or avoiding local users exceeding their permissions (usually not carried out by the local user but by someone assuming their identity) or for helping new users to configure the services properly.

In this sense, we must examine:

- **Advanced TCP/IP configuration**: the sysctl command can be used to modify the parameters of the kernel during execution or at start up, to adjust them to the needs of the system. The parameters that may be modified are the ones in the /proc/sys/ directory and they can be consulted with sysctl -a. The simplest way of modifying these parameters is through the /etc/syscntl.conf. configuration file. After carrying out the modification, we must restart the network:
  /etc/init.d/networking restart
  In this section, we will examine some modifications for improving the network's performance (improvements depending on conditions) or the system's security (consult the references for more details) [Mou01]:
  net.ipv4.icmp_echo_ignore_all = 1

- Does not respond to ICMP packages, such as the ping command for example, which could mean that there is a denial-of-service (DoS) attack.
  net.ipv4.icmp_echo_ignore_broadcasts = 1

- Avoids congestion in the network not responding to the broadcast.
  net.ipv4.conf.all.accept_source_route = 0
  net.ipv4.conf.lo.accept_source_route = 0
  net.ipv4.conf.eth0.accept_source_route = 0
  net.ipv4.conf.default.accept_source_route = 0

- Inhibits the IP source routing packages, which could represent a security threat (in all the interfaces).
  net.ipv4.tcp_syncookies = 1
  net.ipv4.conf.all.accept_redirects = 0

- Permits the rejection of a DoS by SYNC packages, which would consume all the system's resources, forcing the user to reboot the machines.
  net.ipv4.conf.lo.accept_redirects = 0
  net.ipv4.conf.eth0.accept_redirects = 0
  net.ipv4.conf.default.accept_redirects = 0

- Useful for avoiding ICMP redirect acceptance attacks (these packages are used when the routing does not have the appropriate route) in all the interfaces.
  net.ipv4.icmp_ignore_bogus_error_responses = 1

- Sends alerts on all the error messages in the network.
  net.ipv4.conf.all.rp_filter = 1
  net.ipv4.conf.lo.rp_filter = 1
  net.ipv4.conf.eth0.rp_filter = 1
  net.ipv4.conf.default.rp_filter = 1

- Enables protection against IP spoofing in all the interfaces.
  net.ipv4.conf.all.log_martians = 1
  net.ipv4.conf.lo.log_martians = 1
  net.ipv4.conf.eth0.log_martians = 1
  net.ipv4.conf.default.log_martians = 1
  Generates a log of all the spoofed packets, source routed packets and redirect packets.

- The following parameters will permit the machine to attend the TCP connections faster and better.
  net.ipv4.tcp_fin_timeout = 40, By default, 60.
  net.ipv4.tcp_keepalive_time = 3600, By default, 7.200.
  net.ipv4.tcp_window_scaling = 0
  net.ipv4.tcp_sack = 0
  net.ipv4.tcp_timestamps = 0, By default, all at 1 (enabled).

- **Iptables**: the latest versions of GNU/Linux (kernel 2.4 or higher) include a new feature for building package filters called netfilter [Mou01]. This new feature is controlled by a tool called iptables that has better characteristics than its predecessor (ipchains). As we will see in the unit on security, it is extremely easy to build a firewall with this tool for detecting and warding off the most common attacks, such as DoS, IP/MAC spoofing etc. Before it is activated, we have to verify that the kernel is version 2.4 or later, which is the one that is configured to support ipfilter (which means that it is necessary to compile the kernel to activate the option *network packet filtering* [*CONFIG_NETFILTER*], and all the specific suboptions). The specific rules must be activated when booting (for example, through /etc/init.d and the appropriate link in the appropriate rc directory) and will have a format similar (check the references on capacities and complete syntax) to:

```
iptables -A Type -i Interface -p protocol -s SourceIP --
source-port Port -d DestinationIP --destination-port Port
-j Action
```

- **GnuPG**: this tool makes it possible to encrypt data for subsequent sending (emails, for example) or storage, it can also generate digital signatures (it meets the RFC2440 standard) and it does not use patented algorithms, which means that is open source, but we lose compatibility with other tools (for example, PGP 2.0), which use algorithms such as IDEA and RSA. For compiling and/or installing the tool, follow the instructions of the programmers at http://www.gnupg.org/. Firstly, we must create a pair of keys (public and private) by executing, in root, the `gpg --gen-key` command twice and answering the questions that appear. Generally, these keys will be stored in */root*. Then we export (to a website, for example) the public key so that other users can use it to encrypt the mail/information that may only be seen by the user that generated the public key. For this, we must use `gpg --export -ao UID`, which will generate an ASCII file of the UID user's public key.
  In order to import another user's public key, we can use `gpg --import filename`, and to sign a key (which is to tell the system that we are satisfied that the signed key is from who it says it is), we can use `gpg --sign-key UID`. To verify a key, we can use `gpg --verify file/data` and to encrypt/decrypt a key, gpg -sear UID file g, gpg -d file, respectively. [Gnu]

- **Logcheck**: one of a network administrator's main tasks is to check the log files daily (more than once a day) to detect any possible attacks/intrusions or events that may be evidence of these questions. This tool selects compressed information on problems and potential risks (from the log files) and then sends this information to the corresponding administrator, by email, for example. The package includes utilities for executing in independent mode and remembering the last entry verified for the subsequent executions. For information on the configuration/installation, you may consult the references. [Log]

- **PortSentry** and **Tripwire**: these tools help the network administrator to carry out their security tasks. PortSentry makes it possible to detect and respond to port searching processes (the preliminary step before attacking or spamming) in real time and to make various decisions with regard to the actions that are being performed. Tripwire is a tool that will help administrators by warning them of possible modifications and changes in the files, to avoid possible (severe) damage. This tool compares the differences between the current files and a database previously generated to detect changes (insertions and deletions), which is very useful for detecting possible modifications to vital files such as, for example, configuration files. Consult the references on the installation/configuration of these tools. [Tri]

- **Xinetd**: this tool significantly improves the efficiency and performance of inetd and tcp-wrappers. One of the biggest advantages of xinetd is that it can avoid denial-of-access (DoA) attacks through the control mechanisms for services based on the identification of client addresses, during the accessing time and (logging) time. It should not be assumed that Xinetd is the most appropriate option for all the services (for example, it is better if FTP and SSH execute only as daemons), as many of these processes will overload the system and there are secure access mechanisms that do not create interruptions in the system's security. [Xin]
  Compiling and/or installing is simple; we only have to configure two files: `/etc/xinetd.conf` (the configuration file of Xinetd) and `/etc/rc.d/init.d/xinetd` (the Xinetd startup file). The first file contains two sections: *defaults*, which is where we find the parameters that will apply to all the *services*, which will be the ones that activate through Xinetd.

A typical example of the configuration might be:

```
# xinetd.conf
# The default configuration options that are applied to all the
# servers may be modified for each service
defaults
{
instances = 10
log_type = FILE /var/log/service.log
log_on_success = HOST PID
log_on_failure = HOST RECORD
}
# The name of the service must be located in /etc/services to obtain
# the right port
# If the server/Port is not a standard one, use "port = X"
service ftp
{
socket_type = stream
protocol = tcp
wait = no
user = root
server = /usr/sbin/proftpd
}
#service telnet
#{
# socket_type = stream
# protocol = tcp
# wait = no
# user = root
# no_access = 0.0.0.0
# only_from = 127.0.0.1
# banner_fail = /etc/telnet_fail
# server = /usr/sbin/in.telnetd
#}
service ssh
{
socket_type = stream
protocol = tcp
wait = no
user = root
port = 22
server = /usr/sbin/sshd
server_args = -i
}
service http
{
socket_type = stream
protocol = tcp
wait = no
user = root
```

```
server = /usr/local/apache/bin/httpd
}
#service finger
#{
# socket_type = stream
# protocol = tcp
# wait = no
# user = root
# no_access = 0.0.0.0
# only_from = 127.0.0.1
# banner_fail = /etc/finger_fail
# server = /usr/sbin/in.fingerd
# server_args = -l
#}
# End of /etc/xinetd.conf
```

The above mentioned services (#) will not be available. In the *defaults* section, we can install parameters such as maximum number of simultaneous service requests, the type of registry (log) that we require, from which nodes the requests will be received by default, the maximum number of IP requests that will be attended or the services that execute as superservers (imapd or popd), such as:

```
default {
instances = 20
log_type = SYSLOG
authpriv log_on_success = HOST
log_on_failure = HOST
only_from = 192.168.0.0/16
per_source = 3
enabled = imaps
}
```

The *service* section, one for each service, such as:

```
service imapd {
socket_type = stream
wait = no
user = root
server = /usr/sbin/imapd
only_from = 0.0.0.0/0 #allows every client
no_access = 192.168.0.1
instances = 30
log_on_success += DURATION USERID
log_on_failure += USERID
nice = 2
redirect = 192.168.1.1 993 #Makes it possible to redirect the traffic of port 993
to node 192.168.1.1
bind = 192.168.10.4
#Makes it possible to indicate the interface to which the service is associated to avoid
service spoofing problems.
}
```

The `/etc/init.d/xinetd` file makes it possible to start up the server (with the appropriate link, according to the selected runlevel, for example, 3, 4 and 5). It is convenient to change the attributes of both files to guarantee that they are not subsequently modified or disabled with: `chmod 700 /etc/init.d/xinetd; chown 0.0 /etc/init.d/xconfig; chmod 400 /etc/xinetd.conf; chattr +i /etc/xinetd.conf`.

- **Linuxconf**: this is a configuration and administration tool of a GNU/Linux system, but it is considered obsolete for most popular distributions, although it can still be found in some distributions. More information at http://www.solucorp.qc.ca/linuxconf/.

- **Webmin**: this is another tool (webmin-core, webmin-dhcp, webmin-inetd, webmin-sshd packages etc.) that makes it possible to configure and add aspects related to the network through a web interface (we must have installed the Apache server, for example). Although it is still being developed in many distributions, it is not included by default. For more information, please visit http://www.webmin.com/. To execute the tool after it has been installed from a browser, call the URL https://localhost:10000, which will ask you to accept the SSL certificate and the username (root user initially) and the corresponding password.

- **System-config-***: in Fedora, there are a variety of graphic tools that are called system-config-"something" and where "something" is what they have been designed for. In general, if we are in a graphical environment, we can reach each of them using a menu; however, each of these tools means we have to remember the menu. One tool that centralises all the system configs is system-config-control in one single entry in the menu and one single graphical interface from which we can make our selections using a set of icons. For this, we have to go to Applications -> Add/Remove Software and this will start up, in root mode, in the graphical interface of the Pirut software (the Fedora Extras repository must be enabled). In the Pirut interface, the available packages can be searched for using, for example, system-config-*; make the selection for the system-config-control* and click on Apply. Among other options, we can configure almost all of the features of the network and services here.

- **Networkmanager**: it is a tool that makes it possible to manage wireless networks and cable networks easily, simply and without any complications, but it is not the most appropriate for servers (only for desktops). Installing the tool is very easy: `apt-get install network-manager-xx`, where xx is gnome or kde depending on the installed desktop. To configure the tool, we must fill in all the entries in (Debian) /etc/network/interfaces except for the loopback interface, for example, by only leaving:
  auto lo
  iface lo inet loopback
  This step is not obligatory but it does make the process for discovering networks/interfaces quicker. On Debian, there is an extra step that must be taken, as the user must integrate within the netdev group, for reasons related to the permissions. To do this, we must execute (as the root user, or if not, with the sudo command first) `adduser current_user netdev` and reboot the system (or restart the network with `/etc/init.d/net-`

`working restart` and logging out and back in, so that the current user is included in the netdev group).

- Other tools: (some of these are explained in the unit on security) Nmap(explore and audit for network security purposes), Nessus(evaluate the network security remotely), Wireshark http://www.wireshark.org/download.html (ex-Ethereal) (network protocols analyser), Snort(intrusion detection system, IDS), Netcat(simple and powerful utility for debugging and exploring a network), TCPDump(monitoring networks and information acquisition), Hping2(generates and sends ICMP/UDP/TCP packages to analyse how a network works).

Activities

1) Define the following network scenarios:
a) Isolated machine.
b) Small local network (4 machines, 1 gateway).
c) 2 segmented local networks (two groups of 2 machines and one router each and a general gateway).
d) 2 interconnected local networks (two groups of 2 machines + a gateway each).
e) 2 machines connected through a private virtual network. Indicate the advantages/disadvantages of each configuration, for which types of infrastructure they are appropriate and which important parameters are needed.

2) Configure the network in options a, b and d.

Annex. Controlling the services linked to an FC6 network.

An important aspect for all the services is how they are started up. FC6 includes a series of utilities for managing the service daemons (including the network ones). As we have seen on the chapter on local administration, the runlevel is the operating mode that will specify which daemons will be executed. In FC we can find: runlevel 1 (single user), runlevel 2 (multiuser), runlevel 3 (multiuser with network), runlevel 5 (X11 plus /runlevel 3). Typically, we would execute runlevel 5 or 3 if we do not need any graphical interfaces. In order to determine the level that is being executed, we can use /sbin/runlevel, and to know which level will start up by default `cat /etc/inittab | grep :initdefault:`which will give us information such as id:5:initdefault: (we can also edit /etc/inittab to change the default value.)

To visualise the services that are executing, we can use `/sbin/chkconfig` –list and to manage them, we can use system-config-services in the graphic mode or ntsysv in the command line. To enable individual services, we can use chkconfig; for example the following command enables the crond service for levels 3 and 5: `/sbin/chkconfig --level 35 crond on`. Regardless of how the services were started up, we can use /sbin/service –status-all or individually /sbin/service crond status to see the status of each service. And we can also manage this (start, stop, status, reload, restart), for example `service crond stop` to stop it or `service crond restart` to restart it.

It is important to not disable the following services (unless you know what you are doing): **acpid**, **haldaemon**, **messagebus**, **klogd**, **network**, **syslogd**. The most important services linked to the network (although this is not an exhaustive list and some have been left out, most of the services are listed here) are:

NetworkManager, **NetworkManagerDispatcher**: is a daemon with which we can easily change networks (Wifi and Ethernet basically). If we only have one network, it does not have to be executed.

avahi-daemon, **avahi-dnsconfd**: is an implementation of zeroconf and it is useful for detecting devices and services on local networks without DNS (it is the same as *mDNS*).

bluetooth, **hcid**, **hidd**, **sdpd**, **dund**, **pand**: Bluetooth wireless network is for portable devices (it is not wifi, 802.11). For example, keyboards, mouse, telephones, speakers/headphones etc.

capi, **isdn**: network based on ISDN hardware.

Iptables: this is the standard firewall service in Linux. It is essential for security if we have a network connection (cable, DSL, T1).

Ip6tables: the same applies but for the protocol and networks based on Ipv6.

netplugd: Netplugd can monitor the network and execute commands when the status changes.

netfs: it is used to automatically mount the file systems through the network (NFS, Samba etc.) during startup.

nfs, **nfslock**: these are the standard daemons for sharing file systems through the network in Unix/Linux/BSD-type operating systems.

ntpd: server of time and date through the network.

portmap: is a complementary service for NFS (file sharing) and/or NIS (authentication).

rpcgssd, **rpcidmapd**, **rpcsvcgssd**: it is used for NFS v4 (new version of NFS).

sendmail: this service can be used to manage the mails (MTA) or support services such as IMAP or POP3.

smb: this daemon makes it possible to share files over Windows systems.

sshd: SSH allows other users to connect interactively and securely to the local machine.

yum-updatesd: FC network updating service.

xinetd: alternative service of inetd that presents a set of features and improvements, such as, for example, launching multiple services through the same port (this service may not be installed by default).