Installing and Configuring a Simple Gateway Server

Dashamir Hoxha

gateway-server-en-0.1 (2004-10-01)

Table of Contents

1. Introduction

This tutorial describes the installation and configuration of a small GNU/Linux server that is used as gateway to the Internet and as a web server for a small company or institution. It takes a practical approach, describing all the steps in details, with all the commands that are used in a concrete example.

1.1. Who Should Read This Tutorial

Anybody that wants to learn how to install a gateway GNU/Linux server can read this tutorial. However, it assumes that the reader has some experience with GNU/Linux and it does not explain everything in details. The intended audience is the newbie GNU/Linux admins, not the newbie GNU/Linux users. So, if you have no previous experience with GNU/Linux, it will be hard to understand and to follow the instructions in the tutorial, and you have better to start with some introductory tutorials first, e.g. Introduction to Linux -- A Hands on Guide [http://www.tldp.org/LDP/intro-linux/html/index.html]

In order to help you understand whether you can follow this tutorial easily, try to answer the following questions. If you can answer them positively, then most probably you can follow the instructions easily.

1. Have you ever used GNU/Linux? Do you know what is Fedora?

2. Have you ever installed a GNU/Linux system yourself? Do you know what is a partition? Do you know what is a *swap* partition?

3. Have you ever used the commands of GNU/Linux? Do you know what is a *terminal* ? Have you ever used *ls* , *cd* , *mkdir* , *cp* , *rm* ?

4. Do you know what is *bash* ? Do you know what are *shell scripts* ?

5. Have you ever configured a network interface yourself? Do you know what is an *IP* and a *netmask* ?

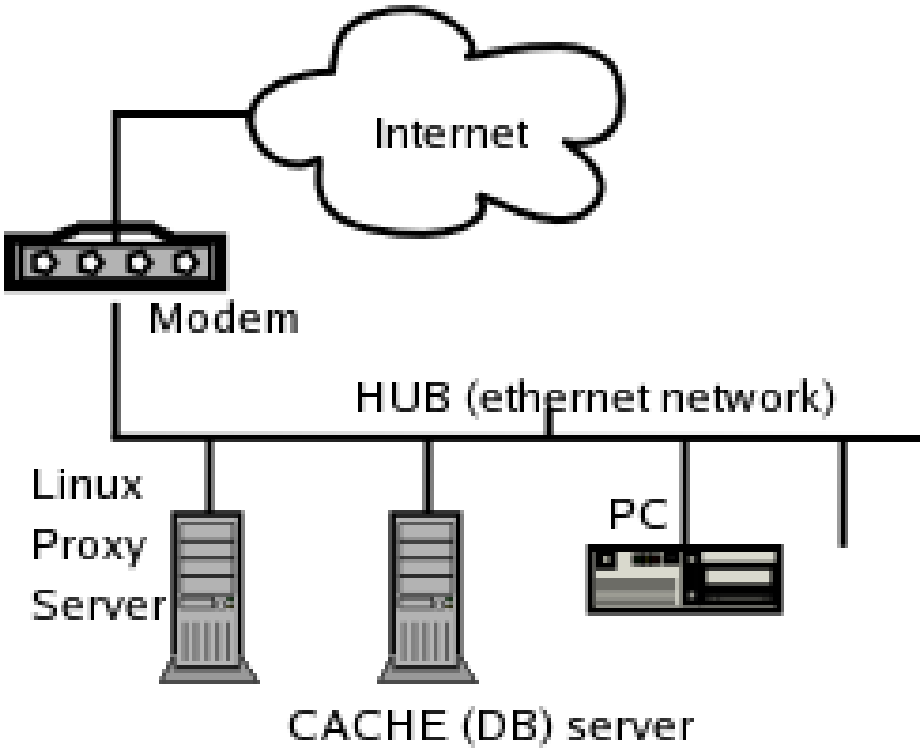
6. Do you know what are services? Have you ever heard about *DNS* , *sendmail* , *apache* ?

7. Have you ever used the *vi* editor?

1.2. The Story

This tutorial is based on a concrete example, which will be described here. Suppose that a small organization (or company, institution, etc.) has a small network of computers and it is connected to Internet using a small router/modem. The diagram below shows how the components of the network are connected physically. The router and the computers are connected in an ethernet network by means of a HUB (which is represented in the picture by the thick line).

Figure 1. Physical diagram of the network

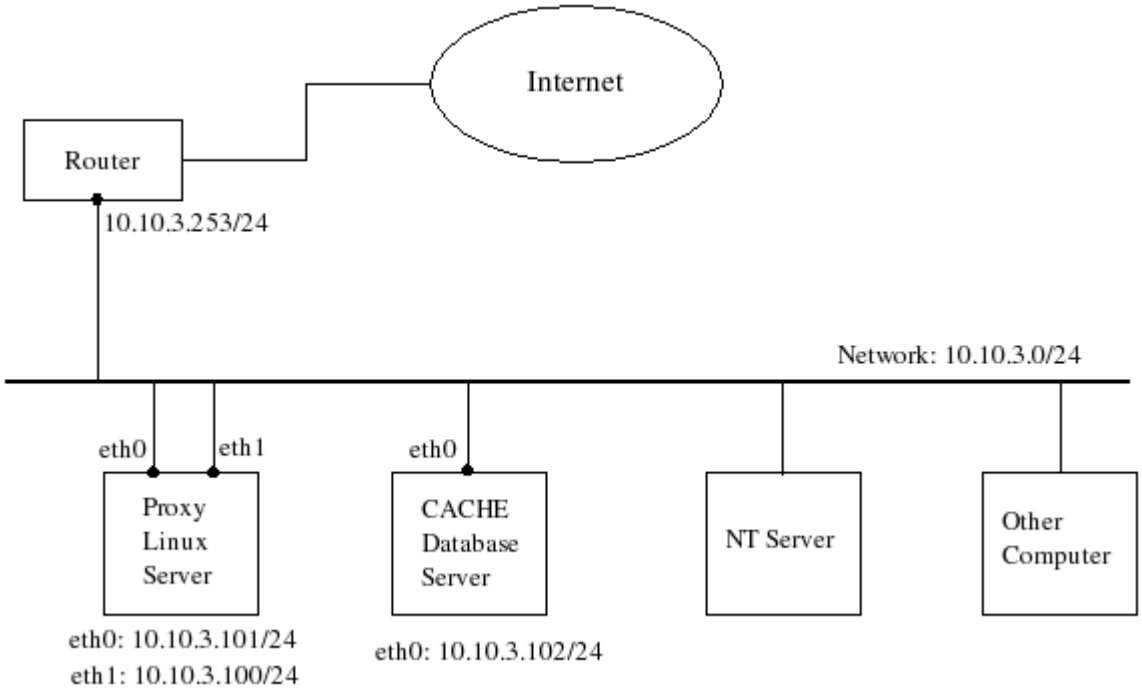


What we want to do is to place a GNU/Linux server between the router and the rest of the local network, so that it serves as a gateway for it. This way it can protect the network with a firewall, it can serve as a web server for the company, etc. We want to use SNAT (Source Network Address Translation, called also *Masquerading*), so that the local computers can access the Internet, but they cannot be accessed from outside. However, we want to be able to

access from outside the port 1972 of a database server that is in the local network. For this, we are going to use DNAT (Destination Network Address Translation, also called *port forwarding*).

Another goal is to make this change in network configuration seamlessly and painlessly, so that the staff of the organization does not experience any interruption in the Internet connection, and we don't have to work all the night in order to do it. For this reason we are going to do it in two steps. First we are going to configure the server according to the following diagram:

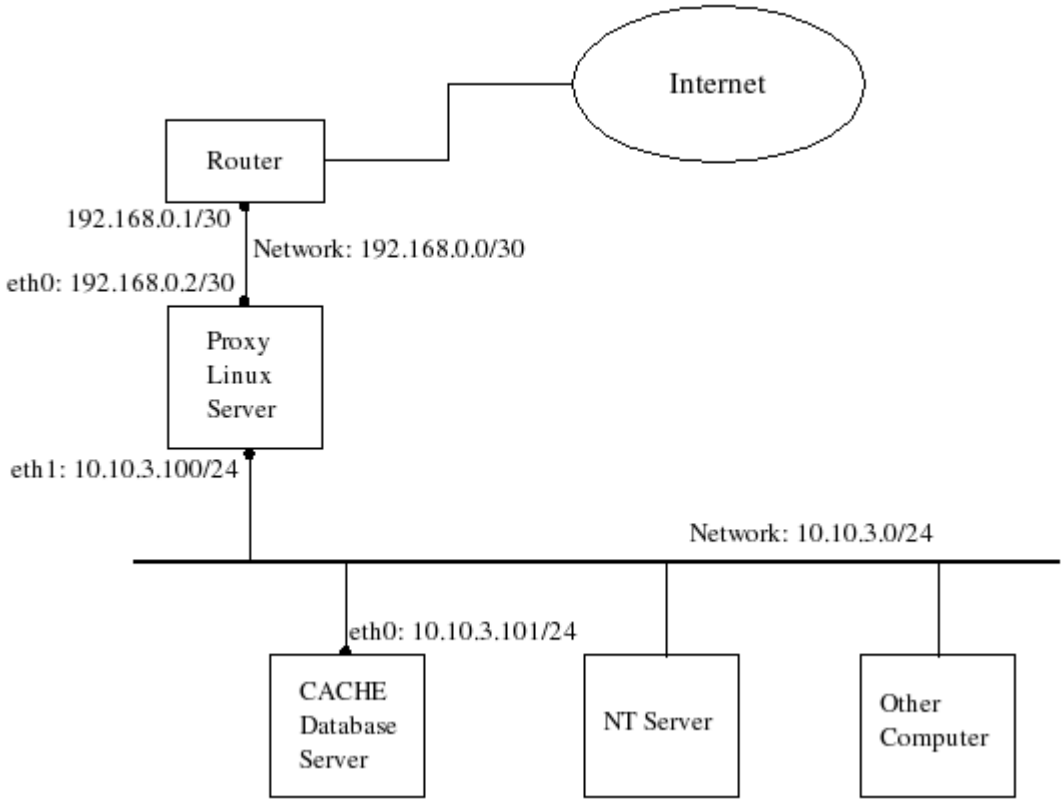
Figure 2. First configuration of the network



In this network configuration the GNU/Linux server can perform all of its functions: gateway, web server, port-forwarding, etc. However the other computers can choose either the GNU/Linux server, or the router itself as gateway. This ensures that the Internet connection is not interrupted for the local network, during the time that the server is installed and tested. Also, the switch to the new gateway can be done gracefully, one by one, without Internet connection interruption.

Once the server is installed and tested, and once all the computers switch to the new gateway, we can change the configuration of the network as shown in the diagram below:

Figure 3. Second configuration of the network



In this configuration the local network can access the Internet only through the GNU/Linux server. However, the local computers don't need to change the gateway, it is the same as before: 10.10.3.100. The physical connection doesn't need to be changed as well: the router, the server and the local machines are still connected to the HUB, same as before.

During the configuration of the server we will take care so that we can switch instantly from the first configuration to the second configuration (we will see later how).

1.3. Requirements

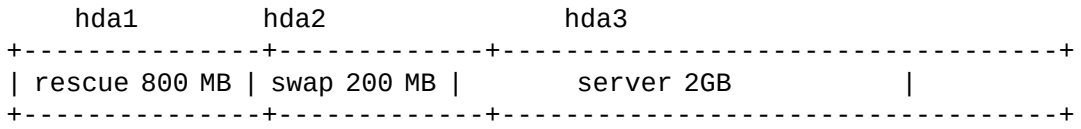
The minimal requirements for the GNU/Linux server are:

- Pentium I, CPU
- 64MB RAM
- 1GB HDD
- Floppy drive (no need for CD-ROM drive)
- Two ethernet network cards.

2. Installation

Lets say that the hard-disk of the server is 3GB and it is partitioned as seen in the figure.

Figure 4. Partitions of the disk



The first partition holds a minimal GNU/Linux installation, which can be used as a rescue system in case that something goes wrong in the server and it becomes inaccessible. It needs actually less than 800 MB, however lets have some space in order to backup any configuration files etc. The rescue partition can be omitted, however the installations described in the tutorial assume the above configuration of the disk partitions.

The swap partition is about twice as the size of RAM. The server partition is about 2GB. It can be even less than 1GB, however it is good to have some free space since we are using it as a web server, and maybe we will need to install anything else later.

The installation is based on Fedora1 and it is done from Internet, using a boot floppy and a network drivers floppy.

2.1. Prepare the Installation Server

An installation server contains the installation disks of GNU/Linux and makes them accessible through NFS or FTP or HTTP. Our installation server will make available Fedora1 through HTTP.

Note

This section can be skipped in case that you don't want to prepare your own installation server but instead want to do the installation directly from the Internet. However, an installation server can be useful for other local installations as well.

1. First download Fedora. I will describe a scenario below however it may change according to the distribution, ftp server, etc. For a list of download mirrors where you can find Fedora, see http://fedora.redhat.com/download/mirrors.html .

```
bash# mkdir /var/local/fedora1
bash# cd /var/local/fedora1
bash# lftp ftp://ftp.easynet.nl
lftp ftp.easynet.nl:~> cd mirror/fedora/1/i386/iso/
lftp ftp.easynet.nl:~> ls
-rw-r--r--  1 ftp    easynet 669122560 Sep  3 21:54 yarrow-i386-disc1.iso
-rw-r--r--  1 ftp    easynet 677511168 Mar 14  2003 yarrow-i386-disc2.iso
-rw-r--r--  1 ftp    easynet 508592128 Mar 14  2003 yarrow-i386-disc3.iso
lftp ftp.easynet.nl:~> at 00:30 tomorrow -- get yarrow-i386-disc1.iso &
lftp ftp.easynet.nl:~> at 02:30 tomorrow -- get yarrow-i386-disc2.iso &
lftp ftp.easynet.nl:~> at 04:30 tomorrow -- get yarrow-i386-disc3.iso &
lftp ftp.easynet.nl:~> exit bg
```

   The download is scheduled for after midnight, so that it does not block the network traffic.

2. Create a script that will mount the Fedora1 ISO disks:

```
bash# vi /var/local/fedora1/mount-fedora-discs.sh
```

```
#!/bin/bash

cd /var/local/fedora1
mount -t iso9660 -o loop yarrow-i386-disc1.iso disc1/
mount -t iso9660 -o loop yarrow-i386-disc2.iso disc2/
mount -t iso9660 -o loop yarrow-i386-disc3.iso disc3/
```

   Make it executable:

```
bash# chmod 755 /var/local/fedora1/mount-fedora-discs.sh
```

3. Mount the Fedora1 ISO disks:

```
bash# cd /var/local/fedora1/
bash# mkdir disc1
bash# mkdir disc2
bash# mkdir disc3
bash# ./mount-fedora-discs.sh
```

   Mount them also each time that the computer is rebooted:

```
bash# vi /etc/rc.d/rc.local
```

```
### mount fedora1 discs
/var/local/fedora1/mount-fedora-discs.sh
```

4. Make them accessible from the web:

```
bash# cd /var/www/html/
bash# mkdir fedora1
bash# cd fedora1/
bash# lndir /var/local/fedora1/disc1
bash# lndir /var/local/fedora1/disc2
bash# lndir /var/local/fedora1/disc3
```

Note

Make sure that *apache* is up and running:

```
bash# /sbin/chkconfig --list httpd
bash# /sbin/chkconfig --level 2345 httpd on
bash# /sbin/service httpd status
bash# /sbin/service httpd restart
```

2.2. Prepare Installation Floppies

To prepare the floppies we will use the floppy images that we can download from the installation server. The IP *11.22.33.44* that is used as the IP of the installation server, should be replaced by the IP of your installation server. In case that you couldn't prepare an installation server (for any reason), you can use any public HTTP installation server that can be found at http://fedora.redhat.com/download/mirrors.html . For example, instead of `http://11.22.33.44/fedora1/` you can use http://mirrors.kernel.org/fedora/core/1/i386/os/ .

To prepare the floppies, follow these steps:

1. Download `bootdisk.img` and `drvnet.img` from http://11.22.33.44/fedora1/images/
2. In GNU/Linux, use the command **dd** to write the floppies, like this:

```
bash$ dd if=bootdisk.img of=/dev/fd0 bs=1440k
bash$ dd if=drvnet.img of=/dev/fd0 bs=1440k
```

3. In windows, download `rawrite.exe` from http://11.22.33.44/fedora1/dosutils/ , and use it like this:

```
C:> rawrite
Enter disk image source file name: bootdisk.img
Enter target diskette drive: a:
Please insert a formatted diskette into drive A: and
press --ENTER-- : [Enter]
C:>
```

2.3. Install Rescue System

The installation of *Rescue* can be done using the first installation floppy. At the `boot:` prompt type **linux dd** :

```
boot: linux dd
```

The parameter *dd* tells to the kernel that we have a *driver disk* , so, after the kernel is loaded, it will ask for driver floppies. At this time we insert the network drivers floppy (the second floppy), and after that the kernel will be able to configure and use the network cards, so that it can perform the rest of the installation from the Internet.

Before continuing with the installation, it will ask for the installation method (select *HTTP*), for the installation server and path (set http://11.22.33.44/fedora1/), for the parameters of the network card and the network (IP, NETMASK, GATEWAY, DNS SERVER, etc.). It will retrieve the installation program from the installation server and the rest of the installation is just like a normal (CD-ROM) installation.

Make sure that when you select the packages, you select no packages at all; this does not mean that no packages at all will be installed, it means that only the most necessary packages will be installed and no additional packages (a minimal installation).

Alternatively, the installation of the *Rescue* system can be done almost automatically like this:

1. Create a *kickstart file* :

```
bash$ vi ks.cfg
```

```
# Kickstart file
network --device eth0 --bootproto static --ip 10.10.3.101
      --netmask 255.255.255.0 --gateway 10.10.3.253
      --nameserver 11.22.33.44 --hostname rescue
network --device eth1 --bootproto static --ip 10.10.3.100
      --netmask 255.255.255.0 --gateway 10.10.3.253
      --nameserver 11.22.33.44 --hostname rescue
url --url http://11.22.33.44/fedora1
lang en_US.UTF-8
langsupport --default en_US.UTF-8 en_US.UTF-8
timezone Europe/Tirane
keyboard us
install
rootpw --iscrypted $1$FpTm50gJ$gL82MlzncIjZA6MPUGkBD/
firewall --enabled --http --ssh
authconfig --enableshadow --enablemd5
mouse none
skipx
bootloader --location=mbr --lba32
#use existing partitions
part / --fstype ext3 --onpart hda3
part swap --onpart hda2
reboot

%packages
```

```
%post
```

Tip

A kickstart file can also be created using the Kickstart Configurator, by selecting the menu System ToolsKickstart.

2. Write it to the first floppy:

```
bash$ mount /mnt/floppy
bash$ cp ks.cfg /mnt/floppy
```

3. At the time of installation write this at the `boot:` prompt:

```
boot: linux dd ks=floppy
```

Then the rest of the installation should continue without intervention.

Note

It is better to write `ks.cfg` in both floppies, in case that you forget to swap the floppies after the network drivers are loaded.

2.4. Install Server System

1. Create the file `ks-server.cfg` (Section 2.4.1, “ks-server.cfg”) and write it on both floppies. If there is not enough space in the first floppy, remove some of the files `*.msg` .

2. Modify the file `syslinux.cfg` (Section 2.4.2, “syslinux.cfg”) in the boot floppy (first floppy) by adding the following lines:

```
default server
prompt 1
timeout 300



label server
  kernel vmlinuz
  append dd ks=floppy:/ks-server.cfg initrd=initrd.img ramdisk_size=8192
```

   If it is readonly, make it writable first.

3. Reboot the machine with the boot floppy in and write at the prompt:

```
boot: server
```

   or just pres **[enter]** , since it is the default.

4. Wait until the installation is finished.

2.4.1. ks-server.cfg

This is the kickstart file that is used to automate the installation of the server. The post-install script at the end of the file does the configuration of the server after installation. However, the configuration steps will be described in detail later, so that they can be done manually, in case that something goes wrong at the installation time. As you can see in the *%packages* section, there are more packages than we need, in case that we may need them later.

```
# Kickstart file
network --device eth0 --bootproto static --ip 10.10.3.101 \
      --netmask 255.255.255.0 --gateway 10.10.3.253 \
      --nameserver 11.22.33.44 --hostname gateway
network --device eth1 --bootproto static --ip 10.10.3.100 \
      --netmask 255.255.255.0 --gateway 10.10.3.253 \
      --nameserver 11.22.33.44 --hostname gateway
url --url http://11.22.33.44/fedora1
lang en_US.UTF-8
langsupport --default en_US.UTF-8 en_US.UTF-8
timezone Europe/Tirane
keyboard us
install
rootpw --iscrypted $1$FpTm50gJ$gL82MlzncIjZA6MPUGkBD/
firewall --enabled --http --ssh
authconfig --enableshadow --enablemd5
mouse none
skipx
bootloader --location=mbr --lba32
#use existing partitions
part / --fstype ext3 --onpart hda3
part swap --onpart hda2
reboot

%packages --resolvedeps
@ web-server
@ mail-server
@ dns-server
@ dialup
@ sql-server
@ editors
@ admin-tools
@ system-tools
@ ftp-server
@ text-internet
grub
kernel
samba
samba-swat

%post

### This script configures the system after installation

cd /
```

```
### get config.tgz from /dev/hda1
mkdir /mnt/hda1
mount /dev/hda1 /mnt/hda1
cp /mnt/hda1/backup/config.tgz .
### alternatively, it can be downloaded from the web
# wget http://11.22.33.44/backup/config.tgz

### restore some configuration files
tar xvpfz config.tgz

### install boot menu
/sbin/grub-install --force-lba /dev/hda

### start httpd
/sbin/chkconfig --level 2345 httpd on
/sbin/service httpd start

### configure and start samba
/usr/sbin/useradd samba
chgrp samba /var/www/html
chmod 775 /var/www/html
/sbin/chkconfig --level 2345 smb on
/sbin/service smb start
```

2.4.2. syslinux.cfg

This configuration file helps to perform the installation in several ways, by passing different parameters to the installation program. The parameters *dd* and *ks=floppy:/ks-server.cfg* in the *server* entry, tell to it that we have a *driver disk* to load (which contains the network drivers), and that installation will be based on the kickstart file `ks-server.cfg` in the floppy.

```
default server
prompt 1
timeout 300
display boot.msg
F1 boot.msg
F2 options.msg
F3 general.msg
F4 param.msg
F5 rescue.msg
F7 snake.msg
label server
  kernel vmlinuz
  append dd ks=floppy:/ks-server.cfg initrd=initrd.img ramdisk_size=8192
label linux
  kernel vmlinuz
  append initrd=initrd.img ramdisk_size=8192
label text
  kernel vmlinuz
  append initrd=initrd.img text ramdisk_size=8192
label expert
```

```
  kernel vmlinuz
  append expert initrd=initrd.img ramdisk_size=8192
label lowres
  kernel vmlinuz
  append initrd=initrd.img lowres ramdisk_size=8192
```

2.5. Fix GRUB (Bootloader Menu)

The bootloader menu will be installed in the Rescue partition, because it is safer there. Follow these steps:

1. Open the file `/boot/grub/grub.conf` and edit it so that it looks like this:

```
bash# vi /boot/grub/grub.conf
```

```
default=1
timeout=10
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
title Rescue
      root (hd0,0)
      kernel /boot/vmlinuz ro root=/dev/hda1
      initrd /boot/initrd-2.4.22-1.2115.nptl.img
title Gateway Server
      root (hd0,2)
      kernel /boot/vmlinuz ro root=/dev/hda3
      initrd /boot/initrd-2.4.22-1.2115.nptl.img
```

2. Now **mount** the rescue partition and **chroot** there:

```
bash# mkdir /mnt/hda1
bash# mount /dev/hda1 /mnt/hda1
bash# /usr/sbin/chroot /mnt/hda1
```

3. After **chroot** , we are in the Rescue system. Modify the file `/boot/grub/grub.conf` same as above.

4. Now re-install grub (it was installed first during installation):

```
bash# /sbin/grub-install --help
bash# /sbin/grub-install --force-lba /dev/hda
```

3. Network Configuration

The network configuration is done by the script `network.sh` (Section 3.2, "/usr/local/config/network.sh") and by the configuration file `network.cfg` (Section 3.3, "/usr/local/config/nework.cfg"), which is included and used by this script (and other configuration scripts as well). The server has two network interfaces, one external (eth0) and one internal (eth1). For the network configuration of the server we need to know the IP and NETMASK of these interfaces, the GATEWAY to the Internet, and the IP of the DB server to which we are going to do port forwarding. This information is stored in the file `network.cfg`.

This way of separating the configuration information from the configuration scripts gives us flexibility for changing the configuration quickly and safely. For example, if we need to change the gateway, all we should do is to modify `network.cfg` accordingly and to run the script `network.sh`:

```
bash# vi /usr/local/config/network.cfg
bash# /usr/local/config/network.sh
```

However, if we have some standard configurations that are repeated time after time, then it is better to prepare some standard configuration files and use one of them accordingly, like this:

```
bash# cd /usr/local/config/
bash# cp network.cfg.1 network.cfg
bash# ./network.sh
```

This is what we actually do in order to switch instantly from the first configuration to the second configuration (and back, if needed), do you remember from the story (Section 1.2, "The Story")?

3.1. Checking

To check that the network is configured properly, the following commands can be used:

- Checking what addresses are assigned to the network interfaces:

```
bash$ /sbin/ip addr help
bash$ /sbin/ip addr ls
```

- Checking the routes:

```
bash$ /sbin/ip route help
bash$ /sbin/ip route ls
```

3.2. /usr/local/config/network.sh

```
#!/bin/bash

### include the configuration file
path=$(dirname $0)
. $path/network.cfg

### set up the links, in  case that they are not up
/sbin/ip link set eth0 up
/sbin/ip link set eth1 up

### flush any existing ip addresses of eth0 and eth1
/sbin/ip address flush eth0
/sbin/ip address flush eth1

### add new addresses
/sbin/ip address add $ETH0_IP/$ETH0_MASK dev eth0
/sbin/ip address add $ETH1_IP/$ETH1_MASK dev eth1

### add a default route
/sbin/ip route add default via $GATEWAY dev eth0
```

3.3. /usr/local/config/nework.cfg

```
### internal interface of the router
GATEWAY=192.168.0.1

### external interface of the web server
ETH0_IP=192.168.0.201
ETH0_MASK=24

### internal interface of the web server
ETH1_IP=192.168.0.200
ETH1_MASK=24

### DB server interface
DB_IP=192.168.0.10
```

3.4. /usr/local/config/network.cfg.1

```
### internal interface of the router
GATEWAY=10.10.3.253

### external interface of the web server
```

```
ETH0_IP=10.10.3.101
ETH0_MASK=24

### internal interface of the web server
ETH1_IP=10.10.3.100
ETH1_MASK=24

### DB server interface
DB_IP=10.10.3.102
```

3.5. /usr/local/config/network.cfg.2

```
### internal interface of the router
GATEWAY=192.168.0.1

### external interface of the web server
ETH0_IP=192.168.0.2
ETH0_MASK=30

### internal interface of the web server
ETH1_IP=10.10.3.100
ETH1_MASK=24

### DB server interface
DB_IP=10.10.3.102
```

3.6. /usr/local/config/network.cfg.3

This is a configuration used in another network. Suppose that we install and test the server in one network and deploy it in another one. The flexibility of the configuration scripts allows us to do this.

```
### internal interface of the router
GATEWAY=192.168.0.1

### external interface of the web server
ETH0_IP=192.168.0.201
ETH0_MASK=24

### internal interface of the web server
ETH1_IP=192.168.0.200
ETH1_MASK=24

### DB server interface
DB_IP=192.168.0.10
```

4. Firewall

In the firewall we will accept only the HTTP port (80), ssh port (22), FTP ports (20, 21), samba ports (137,138,139,445) and the port that will be forwarded to the DB server (1972). Everything else will be rejected. The commands that build the firewall are organized into scripts in the directory `firewall/`. The scripts are constructed such that they can be easily understood and modified, and such that they can allow easy modification of the firewall.

To build the firewall call the script:

```
bash# firewall/iptables.sh
```

It will first destroy the previous firewall (by flashing all the iptables rules) and then build it again by appending new rules.

Important

If you ever restart the iptables service, like this:

```
bash# /sbin/service iptables restart
```

then it will destroy the firewall built by `firewall/iptables.sh` and will build a firewall according to `/etc/sysconfig/iptables`. In order to avoid any bad consequences of this, modify this config file so that it contains the firewall configuration produced by our script. This can be done like this:

```
bash# /sbin/iptables-save | more
bash# /sbin/iptables-save > /etc/sysconfig/iptables
```

The following subsections list the firewall configuration scripts, without much explanations, since they are well commented.

4.1. iptables.sh

```
#!/bin/bash
### this script configures the firewall

path=$(dirname $0)
IPT=/sbin/iptables

### clear all the tables
$IPT --table filter --flush
$IPT --table filter --delete-chain
$IPT --table nat   --flush
$IPT --table mangle --flush
```

```
### accept by default anything in the OUTPUT chain
$IPT --table filter --policy OUTPUT ACCEPT

### the rules of the INPUT chain
$path/input-rules.sh

### the rules of the FORWARD chain
$path/forward-rules.sh

### enable SNAT-ing
$path/source-nat.sh enable

### if there is any argument, display the rules
if [ $# != 0 ]
then
  /sbin/iptables-save
fi
```

4.2. input-rules.sh

```
#!/bin/bash
### the rules of the INPUT chain

path=$(dirname $0)
IPT='/sbin/iptables --table filter'
APPEND="$IPT --append INPUT"

### drop anything that does not match
$IPT --policy INPUT DROP

### accept anything from localhost
$APPEND --in-interface lo --jump ACCEPT

### accept any type of icmp
$APPEND --protocol icmp --icmp-type any --jump ACCEPT

### accept already established connections
$APPEND --match state --state ESTABLISHED,RELATED --jump ACCEPT

### create the new chain LOCAL-NETWORK
$IPT --new-chain LOCAL-NETWORK
### for the local networks, jump to the chain LOCAL-NETWORK
$APPEND --in-interface ! eth0 --source 192.168.0.0/16 --jump LOCAL-NETWORK
$APPEND --in-interface ! eth0 --source 10.0.0.0/8    --jump LOCAL-NETWORK
### add the rules of the chain LOCAL-NETWORK
$path/local-network-rules.sh

### accept connections for ftp, ftp-data, ssh, http
$path/port.sh 20 accept
$path/port.sh 21 accept
$path/port.sh 22 accept
```

```
$path/port.sh 80 accept

### reject anything else
#$APPEND --jump REJECT --reject-with icmp-host-prohibited
#
# This is commented because the default policy will drop them anyway.
# Also, if it is commented, the script 'port.sh' can be used to
# accept or deny any other ports (by appending or deleting rules).
# If it is uncommented, then any other rules that are added later
# will come after this one, and since this one matches anything, they
# will not be reached. The order of the rules does matter!
```

4.3. forward-rules.sh

```
#!/bin/bash
### the rules of the FORWARD chain

path=$(dirname $0)
IPT='/sbin/iptables --table filter'
APPEND="$IPT --append FORWARD"

### enable forwarding in the kernel
echo 1 > /proc/sys/net/ipv4/ip_forward

### the default is to drop anything that does not match
$IPT --policy FORWARD DROP

### accept any type of icmp
$APPEND --protocol icmp --icmp-type any --jump ACCEPT

### accept already established connections
$APPEND --match state --state ESTABLISHED,RELATED --jump ACCEPT

### forward everything from the local networks
$APPEND --in-interface ! eth0 --source 192.168.0.0/16 --jump ACCEPT
$APPEND --in-interface ! eth0 --source 10.0.0.0/8    --jump ACCEPT

#####################################################################
## Enable some port forwardings to internal servers
#####################################################################

### get $DB_IP from the network config file
. /usr/local/config/network.cfg

### forward some ports to $DB_IP
$path/port-forward.sh 1972 $DB_IP enable
$path/port-forward.sh 1506 $DB_IP enable
$path/port-forward.sh 1507 $DB_IP enable
$path/port-forward.sh 23 $DB_IP enable
```

```
### reject anything else
#$APPEND --jump REJECT --reject-with icmp-host-prohibited
#
# This is commented because the default policy will drop them anyway.
# Also, if it is commented, the script 'port-forward.sh' can be used to
# enable or disable any other port forwardings. If it is uncommented,
# then any other rules that are added later will come after this one,
# and since this one matches anything, they will not be reached.
# The order of the rules does matter!
```

4.4. local-network-rules.sh

```
#!/bin/bash
### The rules of the chain LOCAL-NETWORK.
### This chain is used to accept some additional ports for the local
### networks, besides the ports that are accepted in general.

path=$(dirname $0)
IPT='/sbin/iptables --table filter'
APPEND="$IPT --append LOCAL-NETWORK"

### accept samba ports
$IPT --new-chain SAMBA
$APPEND --jump SAMBA
$path/samba-rules.sh

### accept connections for swat
$path/port.sh 901 accept LOCAL-NETWORK

### return to the previous chain
#$APPEND --jump RETURN
#
# This is commented because the default is to return to the previous chain.
# Also, if it is commented, the script 'port.sh' can be used to
# accept or reject any other ports (by appending or deleting rules).
# If it is uncommented, then any other rules that are added later
# will come after this one, and since this one matches anything, they
# will not be reached. The order of the rules does matter!
```

4.5. samba-rules.sh

```
#!/bin/bash
### The rules of the chain SAMBA.
### This chain is used to accept the samba ports.

### accept the udp ports 137 and 138
```

```
APPEND=’/sbin/iptables --table filter --append SAMBA’
$APPEND --match udp --protocol udp --dport 137 --jump ACCEPT
$APPEND --match udp --protocol udp --dport 138 --jump ACCEPT

### accept the tcp ports 139 and 445
path=$(dirname $0)
$path/port.sh 139 accept SAMBA
$path/port.sh 445 accept SAMBA

### return to the previous chain
$APPEND --jump RETURN
```

4.6. port.sh

```
#!/bin/bash
### Accept (or deny, or show status of) tcp connections in the given port
### by appending or deleting a rule in the given chain.

function usage
{
 echo "Usage: $0 [port] [accept|deny|status] [chain]
   where port is a number (like 80) or a service name (like http)
   and chain is: INPUT, FORWARD, LOCAL-NETWORK, SAMBA, etc."
 exit
}

if [ $# = 0 ]; then usage; fi

PORT=$1
ACTION=$2
CHAIN=${3:-INPUT}

## appends or deletes a rule, according to the parameter
function iptables-rule
{
 command=$1
 IPT=/sbin/iptables
 $IPT --table filter --$command $CHAIN \
     --match state --state NEW \
     --match tcp --protocol tcp --destination-port $PORT \
     --jump ACCEPT
}

case $ACTION in
 accept )  iptables-rule append ;;
 deny   )  iptables-rule delete ;;
 *      )  /sbin/iptables-save | grep $CHAIN | grep "dport $PORT" ;;
esac
```

4.7. port-forward.sh

Computers in the local network cannot be accessed from outside, because the proxy server is the only visible point of the local network. However, we need to access from outside the DB server (port 1972) which is in a machine behind the proxy server. This is done using port forwarding; whenever the proxy server gets a packet with a destination port 1972, it forwards it to the DB server in the local network (this is called DNAT -- Destination Network Address Translation). Then, the packets that come from the DB server are forwarded to the outside machine. The outside machine thinks that it is the proxy that is replying to it, although in fact it is a machine behind the proxy that handles its request.

```
#!/bin/bash
### enables or disables port forwardings

function usage
{
 echo "Usage: $0 PORT SERVER_IP [enable|disable|status]
  where PORT is a number (like 80) or a service name (like http)
  and SERVER_IP is the IP of the internal server."
 exit
}

### check that there are at least 2 parameters
if [ $# -lt 2 ]; then usage; fi

PORT=$1
SERVER_IP=$2
ACTION=$3

### append or delete the forward rules according to the parameter
function iptables-fwd-rules
{
 command=$1
 IPT=/sbin/iptables

 ### get ETH0_IP and ETH1_IP from the network configuration file
 . /usr/local/config/network.cfg

 # accept the port for forwarding (both tcp and udp)
 $IPT --table filter --$command FORWARD \
     --match state --state NEW \
     --match tcp --protocol tcp --dport $PORT \
     --jump ACCEPT
 $IPT --table filter --$command FORWARD \
     --match tcp --protocol tcp --dport $PORT \
     --jump ACCEPT

 ### forward the port
 $IPT --table nat --$command PREROUTING \
     --protocol tcp --destination $ETH0_IP --dport $PORT \
     --jump DNAT --to-destination $SERVER_IP

 ### in case that $ETH1 is not gateway for $SERVER_IP
 ### but it doesn't hurt even if $ETH1 is gateway for it
```

```
  $IPT --table nat --$command POSTROUTING \
      --protocol tcp --destination $SERVER_IP --dport $PORT \
      --jump SNAT --to-source $ETH1_IP
}

case $ACTION in
  enable  )  iptables-fwd-rules append ;;
  disable )  iptables-fwd-rules delete ;;
  *       )  /sbin/iptables-save | grep $SERVER_IP | grep "dport $PORT" ;;
esac
```

4.8. source-nat.sh

Source Network Address Translation (SNAT) is a trick (technique) that makes the server act as a proxy for all the networks that uses it as a gateway. This is useful for protecting the local network from the outside attacks, if you own only one real IP, etc.

```
#!/bin/bash
# enable or disable source NAT (masquerading)

function usage
{
  echo "Usage: ${0} [enable | disable | list]"
  exit
}

### check that there is one parameter
if [ -z $1 ]; then usage; fi

ACTION=$1

### get ETH0_IP from the network configuration
. /usr/local/config/network.cfg

### enable forwarding in the kernel
echo 1 > /proc/sys/net/ipv4/ip_forward

### append or delete the SNAT rules
function iptables-snat-rule
{
  command=$1
  IPT=/sbin/iptables

  $IPT --table nat --$command POSTROUTING \
      --out-interface eth0 \
      --jump SNAT --to-source $ETH0_IP
      # --jump MASQUERADE
}

### list the existing SNAT rules
```

```
function list-snat-rules
{
 /sbin/iptables-save --table nat \
    | grep -E 'SNAT|MASQUERADE' \
    | grep --invert-match 'dport'
}

case $ACTION in
 enable  )  iptables-snat-rule append ;;
 disable )  iptables-snat-rule delete ;;
 *       )  list-snat-rules ;;
esac
```

5. Services

5.1. Web Server

Check that apache is working and start it if needed:

```
bash# /sbin/chkconfig --list httpd
bash# /sbin/chkconfig --level 2345 httpd on
bash# /sbin/service httpd status
bash# /sbin/service httpd start
```

5.2. Samba

Samba is used to share the DocumentRoot of the web server, so that everybody in the local network can update the content of the web pages from their windows machines.

1. First backup the original samba config file, `/etc/samba/smb.conf`, because it has some useful comments:

   ```
   bash# cd /etc/samba/
   bash# mv smb.conf smb.conf.bak
   ```

   Then create a new config file, with a content like this:

   ```
   bash# vi smb.conf
   ```

   ```
   [global]
     workgroup = ORGNAME
     server string = Web Server
     hosts allow = 10.10.3. 192.168. 127.
   ```

```
    guest account = samba
    log file = /var/log/samba/%m.log
    max log size = 50
    security = share
    socket options = TCP_NODELAY SO_RCVBUF=8192 SO_SNDBUF=8192

  [www]
    path = /var/www/html
    public = yes
    only guest = yes
    writable = yes
    printable = no
```

2. Create the user *samba* and set permissions to `/var/www/html` so that the user *samba* can read and write it:

```
bash# /usr/sbin/useradd samba
bash# chgrp samba /var/www/html
bash# chmod 775 /var/www/html
```

3. Start the samba service:

```
bash# /sbin/chkconfig --level 2345 smb on
bash# /sbin/service smb start
```

4. Don't forget that the samba ports should be accepted in firewall:

```
/sbin/iptables -A INPUT -m udp -p udp --dport 137 -j ACCEPT
/sbin/iptables -A INPUT -m udp -p udp --dport 138 -j ACCEPT
/sbin/iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 139 -j
ACCEPT
/sbin/iptables -A INPUT -m state --state NEW -m tcp -p tcp --dport 445 -j
ACCEPT
```

   Insert these commands to some configuration scripts, instead of running them manually time after time (our firewall scripts already take care of this).

5. Check that the samba is working:

```
bash$ smbclient --help
bash$ smbclient -L 10.10.3.100
bash$ smbclient //10.10.3.100/www
Password:
smb: \> ls
smb: \> help
```

6. Reconfiguration

In order to make the configurations permanent, add these lines to `/etc/rc.d/rc.local`, so that they are executed each time that the server is rebooted:

```
bash# vi /etc/rc.d/rc.local
```

```
### configure network interfaces
/usr/local/config/network.sh

### configure iptables (firewall and port forwarding)
/usr/local/config/firewall/iptables.sh
```

For updating the configuration of the server without rebooting (e.g. when something is modified in the configuration files), the script `reconfig.sh` (Section 6.1, "reconfig.sh") is used. Steps needed in case of reconfiguration are these:

1. Copy `network.cfg.1` or `network.cfg.2` to `network.cfg`, in order to enable this configuration and disable the other.
2. If needed, make any modifications in the firewall configuration scripts.
3. Run the script `reconfig.sh` in order to change the network configuration immediately, or reboot.

Note

In case that configuration files of httpd and samba have changed, then these services have to be restarted as well:

```
bash# /sbin/service httpd restart
bash# /sbin/service smb restart
```

Caution

In case that the server is accessed and managed remotely, it is quite possible to lock yourself out, e.g. if you make a wrong configuration of the network interfaces or a mistake in the configuration of the firewall. To avoid such troubles, be cautious about new configurations and test them before making them permanent. A script like `test.sh` (Section 6.2, "test.sh") can be used to test a new configuration. It enables the new configuration, and after a certain time (say 60 secs) it goes back to the old (tried and true) configuration. During this time you can test the new configuration and make sure that it is OK.

6.1. reconfig.sh

```
#!/bin/bash
### reconfigure the network after changing
```

```
### any configuration variables

path=$(dirname $0)

### configure network interfaces
$path/network.sh

### configure iptables (firewall and port forwarding)
$path/firewall/iptables.sh
```

6.2. test.sh

```
#!/bin/bash
### This script is for testing any new configurations.
### When the server is accessed and managed remotely,
### something may go wrong and it is possible to lock
### yourself out. To avoid such troubles, this script
### tests any new configs and after a certain time
### (e.g. 60 secs) it goes back to the old (tried and true)
### configuration. During the sleep time you can test
### the new configuration and make sure that it is OK.

mv network.cfg.2 network.cfg
./reconfig.sh
sleep 60
mv network.cfg.1 network.cfg
./reconfig.sh
```

7. Backup

Most of the configuration files and scripts that are specific for this webserver, are kept in the directory `/usr/local/config/`, so that they can be accessed and backed up easily. Even the config files that are not in this directory have a symbolic link inside it for easy access and for having a clear idea of the config files that are modified after the installation. A listing of this directory looks like this:

```
bash$ cd /usr/local/config/
bash$ ls -R
.:
backup.sh       network.cfg   network.cfg.3  slinks
config-files.txt  network.cfg.1  network.sh    test.sh
firewall        network.cfg.2  reconfig.sh

./firewall:
forward-rules.sh  iptables.sh          port-forward.sh  samba-rules.sh
input-rules.sh   local-network-rules.sh  port.sh        source-nat.sh

./slinks:
```

```
grub.conf  rc.local  smb.conf
```

The backup of configuration files and scripts is done by the script `backup.sh`:

```
#!/bin/bash

path=$(dirname $0)

tar --create --verbose --preserve-permissions --gzip \
   --file config.tgz \
   --files-from=$path/config-files.txt

### it can be like this as well:
#tar cvpfz config.tgz --files-from=$path/config-files.txt
```

The script backs up all the files that are listed in the file `config-files.txt` :

```
bash# vi config-files.txt
```

```
/boot/grub/grub.conf
/etc/samba/smb.conf
/etc/rc.d/rc.local
/usr/local/config/
```

Maybe you want to backup the website of the organization as well. To do this, append the line **`/var/www/html/`** to the file above. Alternatively, append the command **tar cvpfz www.tgz /var/www/html/** to the backup script.

In order to perform another installation of a server like this one, or a re-installation of the same server, the backup file `config.tgz` is placed in the installation server:

```
bash$ ssh username@11.22.33.44 mkdir -p /var/www/html/backup/
bash$ scp config.tgz username@11.22.33.44:/var/www/html/backup/
```

Also, they can be backed up in the Rescue partition (together with other useful data), so that they can be accessed easily in case of re-installation:

```
bash# mkdir /mnt/hda1
bash# mount /dev/hda1 /mnt/hda1
bash# mkdir -p /mnt/hda1/backup
bash# cp config.tgz /mnt/hda1/backup/
bash# cp www.tgz /mnt/hda1/backup/
bash# umount /mnt/hda1
```