Building a Gateway Server

Dashamir Hoxha

1.Abstract

A gateway server is a server through which the computers in a LAN access the Internet. This is usually done through NAT. It should also provide firewall protection for the LAN and it can also serve as a DNS and DHCPD server for the LAN.

Besides these, the gateway server can also provide the following services:

- access control for the LAN (which user can access the Internet and which not)
- accounting and statistics (how much Internet resources each user has consumed)
- management (cut the line of the users that have consumed above the allowed limit)
- traffic control (offer different quality of service to different groups of users)
- load balancing and backup link (connecting to the Internet through more than one line and using them efficiently)
- etc.

Some years ago I have been involved in a project for building gateway servers like this, using slackware on old PCs. In this article I will try to explain the things that I have done on this project and how I did them. Later I discovered that in fact I was re-inventing the wheel, because most of these requirements can be fulfilled quite well by using MikroTik, HotSpot, ChilliSpot, freeRadius, etc. Anyway, the shell scripts and web interfaces that I built for this project can still have any educational values, or they can serve as inspiration for somebody else who would like to build such things himself.

So, if you are a results oriented people, maybe the article is not so interesting for you. However, if you are a hands-on, try-for-yourself kind of people, who would like to experiment with such things, then this article can be just right for you.

2.The Requirements (Features)

The gateway server that we built was named SCARLET. Don't ask me why, it was the choice of my partner and apparently there was no particular reason for naming it like that.

The aim of scarlet was to manage the access to the Internet of the clients of a small ISP. However, it can be used as well to manage the connections of the employees of a company or organization. It has two network interfaces, one that is connected on the client side (LAN), and one that is connected on the Internet side (WAN). It can also have more than two network interfaces. For example, there can be more than one LAN of clients. This is the case when we have more than 250 clients, or when we have different classes of clients, having different limitations and quality of services. It is also possible to have another network interface on the WAN side. In this case scarlet will have two different gateways to the Internet and it can do load balancing between them and backup connection. Load balancing means that part of the network traffic will go through one connection and the rest of it through the other. Backup connection means that when one of the connections fails, the traffic is automatically routed

through the other one.

Scarlet gives also DHCP and DNS services to the computers of the LAN(s). It also has a firewall in order to protect itself and the LAN(s) and has a web proxy (squid) in order to improve the quality and efficiency of web access. All the clients that access the Internet through scarlet are registered. Scarlet knows their MAC addresses and allows only the registered clients to access the Internet. When the time of registration expires, the connection of the client is interrupted automatically. Scarlet can also limit the download and upload speed of the Internet connection of the LAN. It also counts the network traffic of each client (MAC) and builds statistics about the network usage.

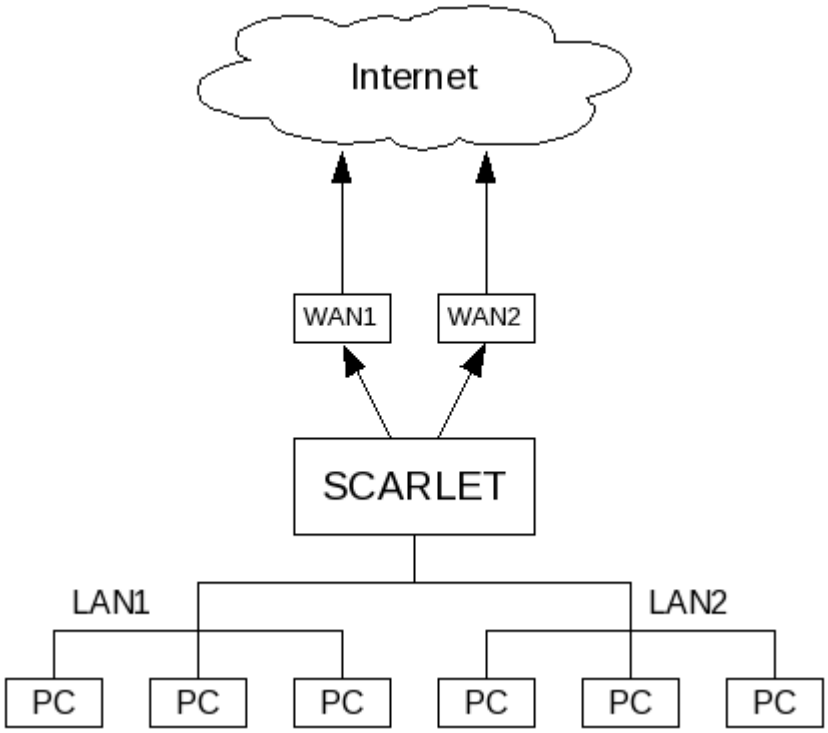


Illustration 1: Diagram of how SCARLET is used.

3.Installing and Accessing Scarlet

In order to develop and test a gateway server we usually need a small lab with at least a few machines and network connections. However, it is possible to simulate such a testing environment using virtual machines and virtual network connections. For this purpose I have used QEMU. Other options like VirtualBox, VMWare, etc. could have been used as well.

3.1.Installing QEMU

From the download page: http://fabrice.bellard.free.fr/qemu/download.html download and install QEMU. The installation is very easy:

```
bash# tar xfz qemu-0.10.2.tar.gz
bash# cd qemu-0.10.2/
```

```
bash# ./configure && make && make install
```

The documentation of QEMU can be found at /usr/local/share/doc/qemu/, or at the web site, or using `man qemu`.

The command `qemu` usually takes as parameter the image of the disk of the computer that is being emulated. This is usually a file in the host system that is treated by *qemu* as the disk of the guest computer. This image file can be created by the command `qemu-img` like this:

```
bash$ qemu-img create images/scarlet.img 3500MB
```

After this, we will have the file scarlet.img which can be used by *qemu* as the hard disk of the guest computer. The size of this virtual hard disk is 3.5GB (sufficient for installing and testing the gateway server). If you try now the command:

```
bash$ qemu images/scarlet.img
```

you will see the emulation window, and then an error message, saying that the disk is not bootable (there is no operating system in it). This is right, because we have installed nothing in it yet.

3.2.Installing SCARLET

First of all we should download scarlet.iso from internet [...].

If we already have a *scarlet* CD, we can copy it like this:

```
bash# dd if=/dev/cdrom of=images/scarlet.iso
```

To install *scarlet* in the virtual disk scarlet.img use this command:

```
bash$ qemu -boot d -cdrom images/scarlet.iso images/scarlet.img
```

The option `-boot d` tells *qemu* to boot from CDROM and the option `-cdrom` tells it to use the given ISO image as a CD. Press *Enter* several times, until you get to the prompt. Then type :

```
# install-scarlet.sh
```

This is the installation script. I swear it is not user friendly at all. It makes a professional installation for sure; fast, efficient and straight to the point. It will partition the disk and then format the parts with *reiserfs*, and then restore there the *scarlet* system. The partitions are restored from the backup files scarlet.hda2.tar.gz and scarlet.hda5.tar.gz (we will see later how to make these backups). Let's have a look at the script:

```
#!/bin/sh
```

```
### install scarlet

### create the partition table of /dev/hda
echo "Erasing the current partition table of /dev/hda"
dd if=/dev/zero of=/dev/hda count=63b
echo "Creating a new partition table for /dev/hda"
fdisk_commands="
n\n p\n 1\n \n +32\n \
n\n p\n 2\n \n +64\n \
n\n e\n 3\n \n \n \
n\n l\n \n +128\n \
n\n l\n \n +128\n \
n\n l\n \n \n \
t\n 1\n 82\n \
w\n"
#echo -e $fdisk_commands  ##debug
echo -e $fdisk_commands | fdisk /dev/hda
fdisk -l /dev/hda

### mount cdrom
mkdir /mnt/cdrom
mount /dev/hdc /mnt/cdrom

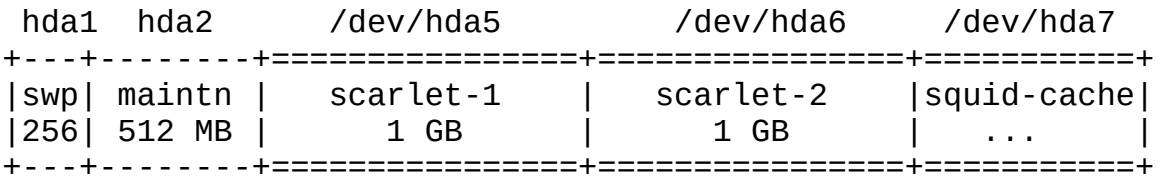
### install the maintenance system in /dev/hda2
echo "Formating /dev/hda2:"
mkreiserfs /dev/hda2
echo "Installing the maintenance system:"
mkdir /hda2
mount /dev/hda2 /hda2
cd /hda2
tar --extract --gunzip --preserve \
    --file=/mnt/cdrom/scarlet.hda2.tar.gz -v
cd /
echo "Installing lilo from the maintenance system:"
chroot /hda2 lilo
### install the scarlet system in /dev/hda5
echo "Formating /dev/hda5:"
mkreiserfs /dev/hda5
echo "Installing the scarlet-1 system:"
mkdir /hda5
mount /dev/hda5 /hda5
cd /hda5
tar --extract --gunzip --preserve \
    --file=/mnt/cdrom/scarlet.hda5.tar.gz -v
cd /

### format the swap partition
echo "Formating the swap partition:"
```

```
mkswap /dev/hda1

### start the maintenance script from the maintenance system
echo "Starting the maintenance script from the maintenance system:"
chroot hda2 /usr/local/maintenance/maintenance.sh
```

A usual installation of scarlet uses the disk as shown in the following diagram:



Scarlet itself is installed in two different partitions, which are labeled *scarlet-1* and *scarlet-2*. There is also a *maintenance* system, a *swap* partition and a partition used for the cache of squid. By default, *scarlet-1* is used, and *scarlet-2* is a backup copy of it. The backup can be done from the *maintenance* system. In case that something happens to the first scarlet, we can quickly boot to the other one, until we find and fix the problem. The maintenance system is used to make backup/restore etc.

After installation, scarlet can be booted with the command:

```
bash$ qemu images/scarlet.img
```

(the option '-boot c' which says to boot from the HDD image is the default).

3.3.Using a Better Network Configuration

After booting scarlet like that, you will notice some error messages about device *eth1* which is missing. The lack of network interface *eth1* will also cause some other problems, because the scarlet system assumes that there are two network interfaces, one external (eth0) and one internal (eth1).

This problem is created because *qemu* by default creates only one network interface for the emulated system. We can correct this problem by using advanced *qemu* options about network configuration. Let's create some scripts for starting Scarlet with qemu.

start-scarlet.sh:

```
#!/bin/bash
### Start scarlet with two network cards.
### Note: it should be run by root, so that tapX interfaces
###       can be created.

### make sure that the script is called by root
if [ $(whoami) != "root" ]
then
  echo "Should be called by root"
```

```
  exit
fi

### emulate a testbox with two network cards
qemu -m 64 -serial /dev/tty8 images/scarlet.img \
     -net nic,vlan=0  -net nic,vlan=1  \
     -net tap,vlan=0,ifname=tap0,script=./tap0.sh \
     -net tap,vlan=1,ifname=tap1,script=./tap1.sh  &
```

tap0.sh:

```
#!/bin/bash
/sbin/ip link set tap0 up
/sbin/ip address flush dev tap0
/sbin/ip address add 192.168.100.1/24 dev tap0
/sbin/ip address add 10.0.0.10/24 dev tap0
```

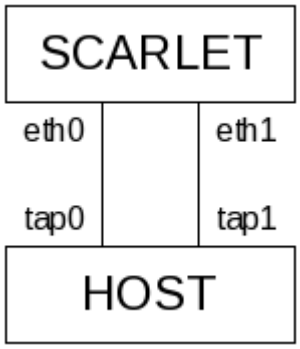
tap1.sh:

```
#!/bin/bash
/sbin/ip link set tap1 up
/sbin/ip address flush dev tap1
/sbin/ip address add 192.168.10.101/24 dev tap1
```

There are a few qemu options to be explained here. First of all, `-m 64` tells `qemu` to use up to 64MB RAM for the emulated scarlet. It really does not need more that this for testing. The option `-serial /dev/tty8` tells `qemu` to connect the serial port of the emulated computer to the device */dev/tty8*. Press now Ctrl+Alt+F8, and you will access the serial console of scarlet (press Ctrl+Alt+F7 to get back). You already know that the `images/scarlet.img` is the HDD of the emulated system.

The option `-net nic` tells qemu to create a virtual network interface for the emulated (guest) system. Since it is used twice, the emulated scarlet will have two network interfaces (*eth0* and *eth1*). The option `-net tap` tells qemu to create a virtual network interface on the host system. Two tap interfaces will be created, *tap0* and *tap1*. The parameter *vlan* is used to create virtual LANs. The parameter `vlan=0` means that the network card should be connected to the virtual hub (or switch) with number 0. So, all the network cards that have the same *vlan number* are connected to the same hub/switch, being thus on the same LAN. We also tell `qemu` to use the script tap0.sh in order to configure the virtual interface *tap0* of the host, and the script tap1.sh for the interface *tap1*.

The logical diagram of the virtual network that is created by *qemu* is like this:



If we run `/sbin/ip addr` on the host, we see that the interface *tap1* has address *192.168.10.101/24* and interface *tap0* has two addresses: *192.168.100.1/24* and *10.0.0.10/24*. If we run `ip addr` inside scarlet , we will see something like this:

```
root@scarlet:/# ip addr
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc htb qlen 1000
    link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.250/24 scope global eth0
    inet 192.168.100.250/24 scope global eth0
    inet 10.0.0.1/24 scope global eth0
3: eth1: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1500 qdisc htb qlen
    link/ether 52:54:00:12:34:57 brd ff:ff:ff:ff:ff:ff
    inet 192.168.10.1/24 scope global eth1
root@scarlet:/#
```

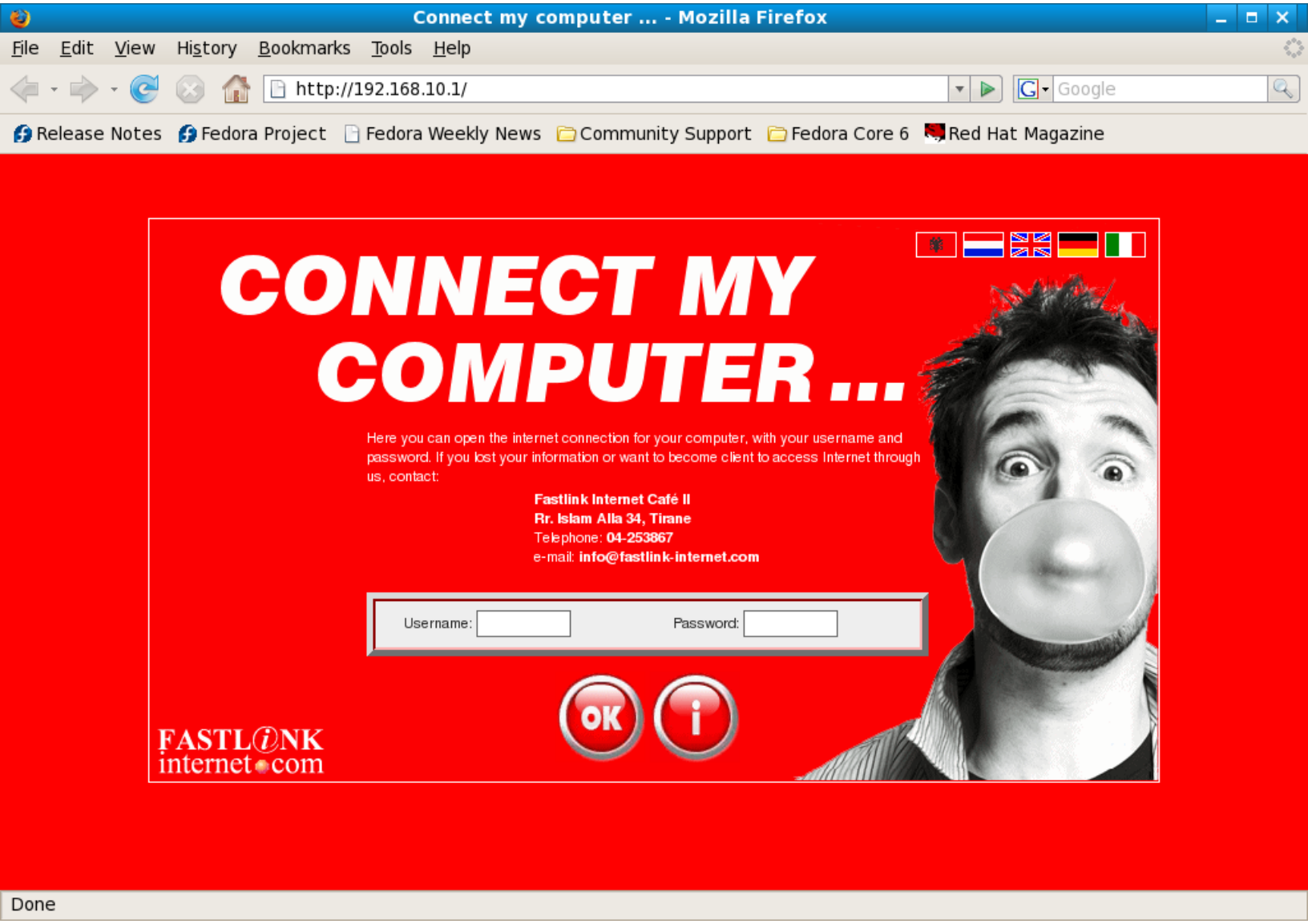
In order to check the "physical" connections of the network that is built by qemu, press Ctrl +Alt+2 to switch to the qemu monitor, then, in the `(qemu)` prompt, give the command `info network`. Finally press Ctrl+Alt+1 to get back to the Linux console. This command should output something like this:

```
(qemu) info network
VLAN 0 devices:
  tap: ifname=tap0 setup_script=./tap0.sh
  ne2000 pci macaddr=52:54:00:12:34:56
VLAN 1 devices:
  tap: ifname=tap1 setup_script=./tap1.sh
```

```
  ne2000 pci macaddr=52:54:00:12:34:57
(qemu)
```

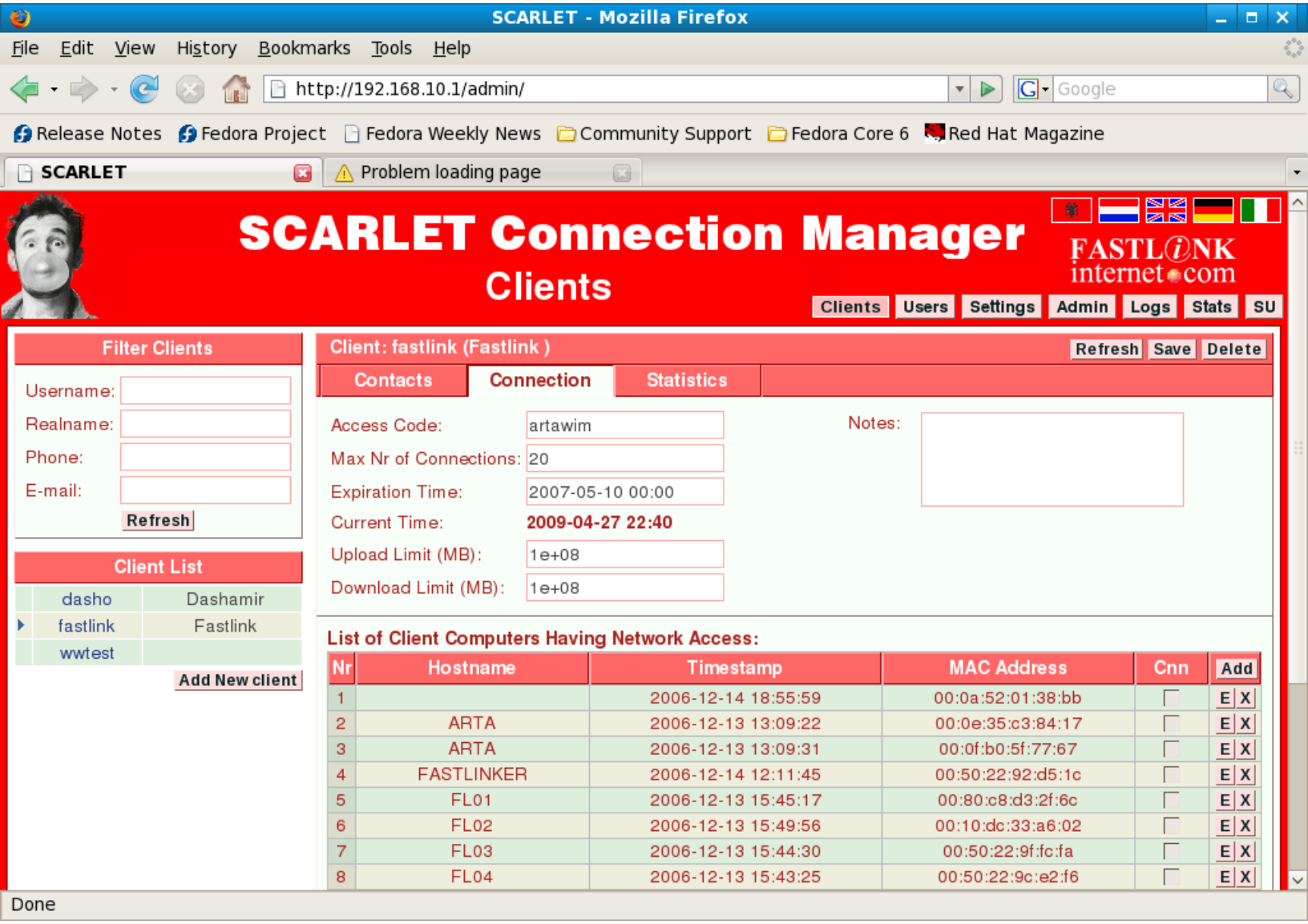
3.4.Accessing Scarlet From the Web Interface

Try to open http://192.168.10.1/ in a browser, and you will see this interesting page:



This is the screen that gets an internet client when he tries to access a web page but he is not registered yet (or his registration has expired).

Try also http://192.168.10.1/admin/ and login with username 'superuser' and password 'admin'. This is a screenshot from the administrative interface of Scarlet:



For more screenshots from Scarlet check the appendix.

3.5.Accessing Scarlet Through SSH

Besides accessing scarlet through the QEMU console and the serial port console (Ctrl+Alt +F8) we can access it through *ssh* as well. However, for security reasons a gateway server has to be restrictive on giving access, so the *ssh* can be accessed only on the port 2222 of the IP *10.0.0.1* on the interface *eth0*. Try to login using the command:

```
bash$ ssh -p 2222 root@10.0.0.1
```

The password of root is 'admin'.

Accessing the emulated system through *ssh* is more convenient than accessing it through the

QEMU console and the serial console because it is possible to copy/paste (commands, configurations, etc.) between the host system and the emulated system. However, if something goes wrong with the network configuration or the firewall of the emulated system and we cannot login anymore, then we can still access it through the serial console and the QEMU console.

While you are logged in to scarlet, go to /var/www/htdocs/scarlet

```
cd /var/www/htdocs/scarlet
```

This directory contains almost all the customizations that are done to a common Slackware system. Most of the content of the directory itself is made up from the web application that is used to configure and manage the gateway remotely. However, there is a subdirectory in it, called server-config, which is used to configure the gateway server itself.

In this directory you will find the configuration file gateway.cfg which contains the configuration parameters of the gateway server. Notice that they are the same configuration parameters that are managed through the web interface. This configuration file is included in all the configuration scripts, in order to get the parameters.

The configuration script update-config-files.sh updates several configuration files of the server: /etc/resolv.conf, /etc/resolv.dnsmasq.conf, /etc/hosts, /etc/dnsmasq.conf, /etc/apache/httpd.conf and /etc/squid/squid.conf. These configuration files are updated by replacing variables with configuration parameters in template files (which can be found in the subdirectory templates/). But what are the modifications that are made in these template files? How they are different from the original configuration files? This can be seen in the .diff files in the templates directory.

The directory firewall/ contains the scripts that are used to build the firewall of the gateway server. This firewall works also as a gateway keeper, because it allows only certain clients to have access to the Internet.

The directory trafic/ contains scripts that are used to do traffic control. Traffic control makes sure that the clients do not have download and upload speeds that are higher than the assigned limits, that interactive traffic gets priority over the bulk traffic, that the connection resources are distributed fairly between the clients, etc.

The directory traffic-counter/ contains scripts that are used to count the traffic (download and upload traffic) of each client and to generate usage statistics for each of them.

Have a look at all these configuration files and scripts and try to understand how they work. If you have difficulties understanding them, don't worry. They are a bit difficult and do contain some advanced stuff. We will try to discuss them later in more details.

4.Preparing a Scarlet Installation CD

We used the CD image scarlet.iso in order to install Scarlet. But how is prepared this image? Suppose that we have made some modifications/improvements to the Scarlet system. How can we prepare a new version of scarlet.iso CD image?

4.1.Making a Scarlet ISO

The scripts that are used to prepare the Scarlet ISO can be checked out from the subversion repository on SourceForge:

```
bash$ svn checkout
https://netaccess.svn.sourceforge.net/svnroot/netaccess/scarlet/ins
tall
```

The image scarlet.iso is prepared by the script make-scarlet-iso.sh . It is actually a modified Slackware bootable CD (the first installation CD of Slackware 11). The modification is that all the Slackware packages are removed from the CD and instead of them the backup files scarlet.hda2.tar.gz, scarlet.hda5.tar.gz and wan.hda2.tar.gz are copied into it. Also initrd is modified so that the scripts install-scarlet.sh and install-wan.sh are copied inside it.

make-scarlet-iso.sh:

```
#!/bin/sh
### Make the ISO image of the installation CD. This CD contains
### the backups scarlet.hda2.tar.gz, scarlet.hda5.tar.gz and
### wan.hda2.tar.gz. It also contains the installation scripts
### install-scarlet.sh and install-wan.sh .
### This CD is a Slackware bootable CD, without any slackware
packages.
### After it is booted, the script install-scarlet.sh or the script
### install-wan.sh can be used to install the systems scarlet or
wan.

### make sure that the script is called by root
if [ $(whoami) != "root" ]; then echo "Should be called by root";
exit; fi

### go to this directory
cd $(dirname $0)

### make initrd.img
./make-initrd.sh

### make backups of scarlet.hda2, scarlet.hda5 and wan.hda2
./make-system-backups.sh

### create the ISO image
mkisofs -R -J -v -d -N -hide-rr-moved -graft-points \
        -no-emul-boot -boot-load-size 4 -boot-info-table \
        -sort slack-cd1/isolinux/iso.sort \
        -b isolinux/isolinux.bin \
        -c isolinux/isolinux.boot \
        -V "Scarlet Install" \
```

```
        -A "Scarlet Install CD" \
        -o images/scarlet.iso \
        -x slack-cd1/isolinux/initrd.img \
        -x slack-cd1/slackware \
        slack-cd1/ \
        scarlet-cd/ \
        /=images/scarlet.hda2.tar.gz \
        /=images/scarlet.hda5.tar.gz \
        /=images/wan.hda2.tar.gz \
        /=install-scarlet.sh \
        /=install-wan.sh

### clean up
rm -rf scarlet-cd/
```

make-initrd.sh :

```
#!/bin/bash
### modify initrd by adding the scripts install-scarlet.sh
### and install-wan.sh

### make sure that the script is called by root
if [ $(whoami) != "root" ]; then echo "Should be called by root";
exit; fi

### go to this directory
cd $(dirname $0)
path=$(pwd)

### copy the original initrd.img, unzip and mount it
cp slack-cd1/isolinux/initrd.img initrd-img.gz
gunzip initrd-img.gz
rm -rf initrd/
mkdir initrd/
mount -o loop initrd-img initrd/

### modify initrd by adding install-scarlet.sh and install-wan.sh
cp install-scarlet.sh initrd/
cp install-wan.sh initrd/
chown root.root initrd/install-scarlet.sh
chown root.root initrd/install-wan.sh
### copy initrd to scarlet-cd
umount initrd/
gzip initrd-img
rm -rf scarlet-cd/
mkdir -p scarlet-cd/isolinux/
```

```
mv initrd-img.gz scarlet-cd/isolinux/initrd.img

### clean
rmdir initrd
```

make-system-backups.sh :

```
#!/bin/bash
### make backups of the partitions hda2 of images/wan.img,
### hda2 of images/scarlet.img and hda5 of images/scarlet.img

### make sure that the script is called by root
if [ $(whoami) != "root" ]; then echo "Should be called by root";
exit; fi

### go to the image/ directory
cd $(dirname $0)
cd images/

### mount the second partition of wan.img to the directory
wan.hda2/
/sbin/losetup -o 271434240 /dev/loop0 wan.img
mkdir -p wan.hda2/
mount /dev/loop0 wan.hda2/

### mount the second partition of scarlet.img to the directory
scarlet.hda2/
/sbin/losetup -o 271434240 /dev/loop1 scarlet.img
mkdir -p scarlet.hda2/
mount /dev/loop1 scarlet.hda2/
### mount the partition 5 of scarlet.img to the directory
scarlet.hda5/
/sbin/losetup -o 806109696 /dev/loop2 scarlet.img
mkdir -p scarlet.hda5/
mount /dev/loop2 scarlet.hda5/

### make backups of wan.hda2/, scarlet.hda2/ and scarlet.hda5/
tar --create --gzip --preserve --one-file-system --verbose \
    --file=wan.hda2.tar.gz     --directory=wan.hda2/  .
tar --create --gzip --preserve --one-file-system --verbose \
    --file=scarlet.hda2.tar.gz --directory=scarlet.hda2/  .
tar --create --gzip --preserve --one-file-system --verbose \
    --file=scarlet.hda5.tar.gz --directory=scarlet.hda5/  .

### clean up
umount wan.hda2/
umount scarlet.hda2/
```

```
umount scarlet.hda5/
/sbin/losetup -d /dev/loop0
/sbin/losetup -d /dev/loop1
/sbin/losetup -d /dev/loop2
rmdir wan.hda2/
rmdir scarlet.hda2/
rmdir scarlet.hda5/
```

The way that we make the backups of the partitions hda2, hda5 etc. is quite interesting. We create some loop devices, then we mount them and suddenly we are able to access the file systems in the partitions that are inside the virtual disk. Then we just make a backup (archive) of the whole file system there.

The way that the loop devices are created seems like magic, using some strange numbers. However these numbers do have sense, they are the number of bytes (offset) where the partition that we want to access starts. But how do we know what is the offset of a partition on the disk? We can find it out from the partition table. Login to Scarlet, give the command `` `fdisk /dev/hda` ``, press 'u' to change the units to sectors then press 'p' to print the partition table. You will see that the unit is a sector of 512 bytes, /dev/hda2 starts on unit 530145 and /dev/hda5 starts on unit 1574433. Then the partition /dev/hda2 starts on byte 512*530145=271434240 of the disk, and the offset of /dev/hda5 is 512*1574433=806109696.

4.2.What Was Created First, Egg or Chicken?

At the beginning of the article we installed a scarlet system from scarlet.iso . On the previous section we said that we make scarlet.iso from a scarlet system. Then, what was created first, a scarlet system or scarlet.iso?

It is like the problem of egg and chicken. Eggs come out from chickens. Chickens come out from eggs. Then, what was created first, eggs or chickens?

Another similar problem is the problem of compilers. All the executable programs have to be compiled by a compiler. A compiler is an executable program. Then, what compiles the compiler? Maybe another compiler? Then, what compiles the other compiler?

I am not sure about eggs-or-chickens and about compilers. However, in the case of Scarlet, I initially installed a plain, simple and minimal Slackware system, using the first Slackware installation CD (Slack11). Then I installed additional required packages and modified the system until it was working as I wished. Then I created scarlet.iso in order to install this system easily to other machines.

The file dependencies.txt contains a list of packages that are required by scarlet, and the file packages.txt contains a list of all the slackware packages that are installed in Scarlet (as listed in /var/log/packages/). Both of these files are appended in the appendix.

Besides the usual Slackware packages, I have also installed the scarlet web interface and the configuration/management scripts, like this:

```
root@scarlet:/# cd /var/www/htdocs/
root@scarlet:/var/www/htdocs# svn ls
```

```
https://netaccess.svn.sourceforge.net/svnroot/netaccess/scarlet/branches/

root@scarlet:/var/www/htdocs# svn checkout https://netaccess.svn.sourceforge.net/svnroot/netaccess/scarlet/branches/release-5 scarlet

root@scarlet:/var/www/htdocs# cd scarlet/

root@scarlet:/var/www/htdocs# vim sysconfig

root@scarlet:/var/www/htdocs# ./install.sh
```

The script install.sh itself uses other scripts that are located on install/, cron/, etc. Some of the configuration files and scripts that are modified or created by it are these:

- /etc/ssh/sshd_config
- /etc/rc.d/rc.local
- /etc/inittab
- /etc/securetty
- /etc/mail/aliases
- /etc/cron.hourly/netaccess.check-time-limits.sh
- /etc/cron.daily/netaccess.check-traffic.sh
- /etc/cron.netaccess.weekly/maintain.sh
- /var/spool/cron/crontabs/root
- /etc/apache/php.ini
- /etc/apache/mod_php.conf
- /etc/HOSTNAME
- etc.

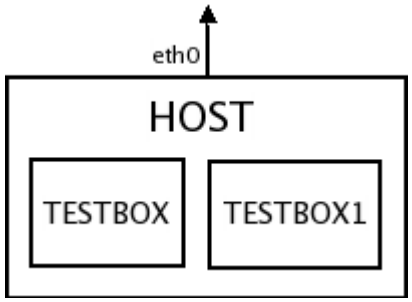
5.Building a Network Testing Virtual Environment

In order to test the gateway router we should install it in an old PC with at least two network cards, using at least another PC as client, ensuring that it has two different gateways, making the necessary network connections (with cables and hubs/switches), etc. This requires some resources. It also takes some time to modify the environment for testing purposes, like adding another network card, decreasing the RAM of the gateway PC, etc. So, it is more practical and efficient to build a virtual testing environment. We have already emulated a virtual network with QEMU in the previous sections. Here we are going to extend it further so that it can match better the network conditions in which Scarlet is supposed to work. For this purpose we are going to use some shell scripts and configurations. Using these scripts, all that is needed for testing is just a computer with sufficient RAM, and optionally a network card for having access to the Internet.

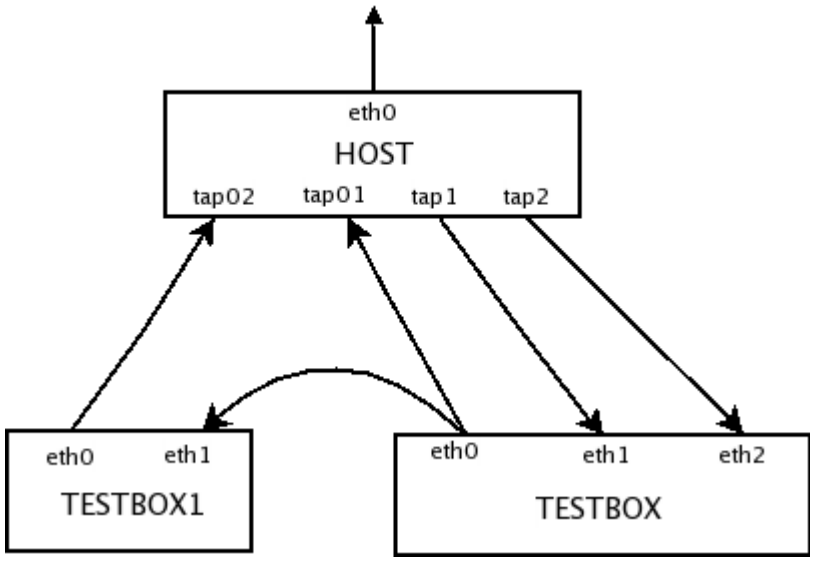
Let us call the real computer the HOST system. Inside it we are going to emulate two guest

systems. On one of them, called TESTBOX, we will install Scarlet, and on the other one, called TESTBOX1, we will install WAN. We can install both TESTBOX and TESTBOX1 using the scarlet.iso CD image, as described previously. For installing TESTBOX we should run the script install-scarlet.sh, and for installing TESTBOX1 we should run the script install-wan.sh . TESTBOX1 (WAN) will be used as a second gateway for TESTBOX (SCARLET).

The figure below gives an idea about this situation: we have two guest systems which are emulated and run simultaneously within the host system.

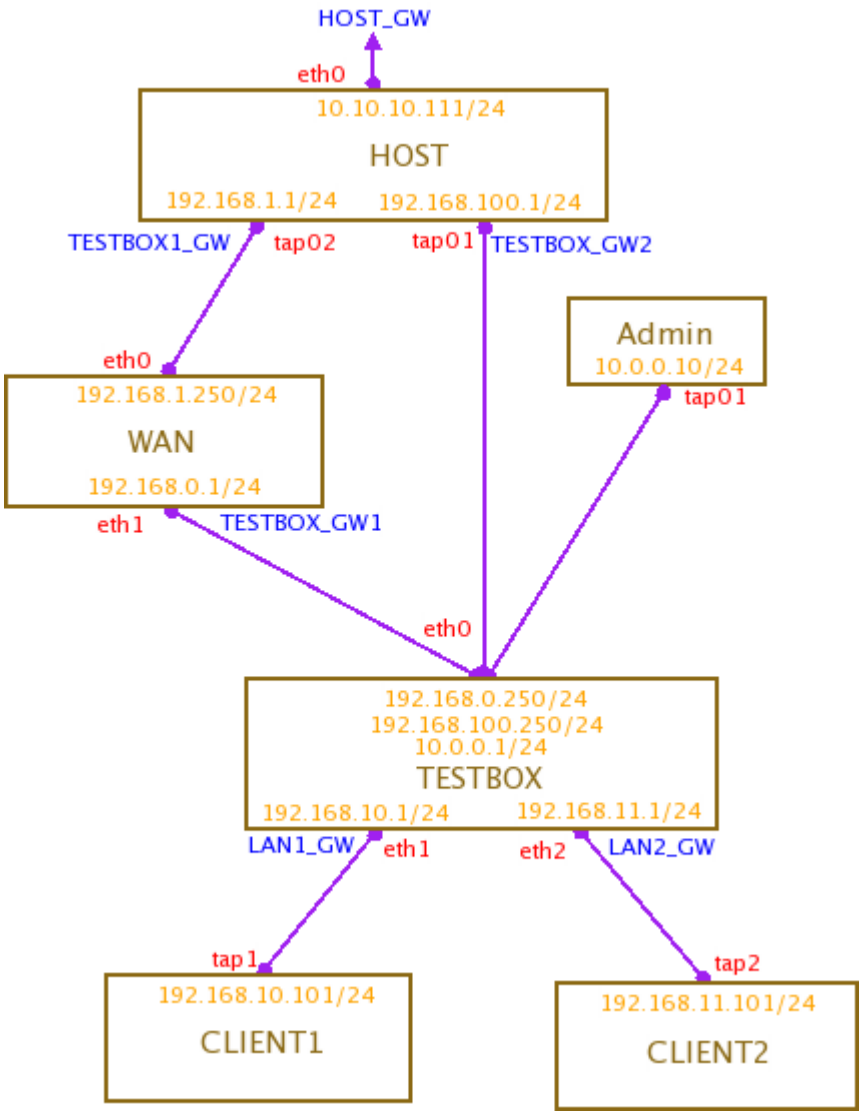


As we have seen already, QEMU gives also the possibility to emulate network interfaces of the TESTBOX and TEXTBOX1 and to connect them to some virtual interfaces of the HOST and to each-other. This figure shows the network connections that should be set up between TESTBOX, TESTBOX1 and HOST:



TESTBOX has 3 network interfaces (eth0, eth1, eth2) which are "physically" connected to the virtual interfaces tap01, tap1 and tap2 of the HOST. TESTBOX1 has the network interface eth0 which is connected to tap02 of the HOST and there is also a connection between the interface eth0 of TESTBOX and the interface eth1 of TESTBOX1. The arrows point in the direction of the network gateways.

By properly configuring these "virtually-physical" network interfaces and connections, we can manage to build a logical network like that which is shown in the following figure, which is sufficient for testing the functionality of Scarlet.



The TESTBOX has two gateways to the Internet, the primary one (TESTBOX_GW1) is through the interface eth1 of TESTBOX1 (which in the logical diagram is labeled WAN), and the secondary one (TESTBOX_GW2) is through the interface tap01 of the HOST. The gateway of TESTBOX1 (WAN) is through the interface tap02 of the HOST. CLIENT1 is a client in the LAN1 (which is connected to eth1 of TESTBOX) and CLIENT2 is a client in LAN2. Both of them use TESTBOX as gateway. 'Admin' is a box that can be used to access TEXTBOX via ssh. CLIENT1, CLIENT2 and Admin can be performed by HOST, which can be set in several roles.

So, we have built a network where the TESTBOX is a gateway server for two LANs, each of them connected to one of its LAN interfaces. The HOST system can be configured to be on the WAN side of the TESTBOX or on each of the LAN networks, in order to test it from each side. It can also access it by ssh, using the Admin IP (because TESTBOX should deny shell access from outside, for security reasons, except from the *access IP*). TESTBOX has also two gateways to the Internet, the primary one through TESTBOX1 and the secondary one through HOST. TESTBOX1 itself is connected to the Internet through HOST.

The scripts that are used to build a network testing virtual environment can be checked out from the subversion repository at sourceforge:

```
bash$ svn checkout
https://netaccess.svn.sourceforge.net/svnroot/netaccess/scarlet/tes
ting
```

The parameters that are used for the configuration of this virtual testing network are in the file network.cfg. They correspond to the labels on the last diagram.

```
### the parameters that are used
### for the configuration of the test network

### gateways
HOST_GW=10.10.10.1          # gateway of the HOST to internet
TESTBOX_GW1=192.168.0.1     # eth1 of TESTBOX1
TESTBOX_GW2=192.168.100.1   # tap01 of HOST
TESTBOX1_GW=192.168.1.1     # tap02 of HOST
LAN1_GW=192.168.10.1        # eth1 of TESTBOX
LAN2_GW=192.168.11.1        # eth2 of TESTBOX

### IP-s
HOST_WAN_IP=${HOST_GW/%1/111}/24           # eth0 of HOST
HOST_LAN_IP1=${TESTBOX_GW2}/24             # tap01 of HOST
HOST_LAN_IP2=${TESTBOX1_GW}/24             # tap02 of HOST

TESTBOX_WAN_IP1=${TESTBOX_GW1/%1/250}/24   # eth0 of TESTBOX
TESTBOX_WAN_IP2=${TESTBOX_GW2/%1/250}/24   # eth0 of TESTBOX
TESTBOX_LAN1_IP=${LAN1_GW}/24              # eth1 of TESTBOX
TESTBOX_LAN2_IP=${LAN2_GW}/24              # eth2 of TESTBOX

CLIENT1_IP=${LAN1_GW/%1/101}/24            # tap1 of HOST (as eth0
of CLIENT1)
CLIENT2_IP=${LAN2_GW/%1/101}/24            # tap2 of HOST (as eth0
of CLIENT2)

ACCESS_IP=10.0.0.1/24        # eth0 of TESTBOX, used to access
(login to) it
ADMIN_IP=10.0.0.10/24        # tap01 of HOST (as eth0 of ADMIN)
```

The gateways of the boxes are taken as primary parameters in the configuration of the network:

- HOST_GW -- gateway of HOST to the Internet.
- TESTBOX_GW1 -- primary gateway of TESTBOX (interface eth1 of TESTBOX1)
- TESTBOX_GW2 -- secondary gateway of TESTBOX (interface tap01 of HOST)

- TESTBOX1_GW -- gateway of TESTBOX1 (interface tap02 of HOST)
- LAN1_GW -- gateway of LAN1 (interface eth1 of TESTBOX)
- LAN2_GW -- gateway of LAN2 (interface eth2 of TESTBOX)

The other parameters are derived from them. For example, HOST_WAN_IP is derived from HOST_GW by replacing 1 with 111 in the last part of the IP, and also appending a netmask '/24'. The other IP-s are derived similarly.

In order to access the shell of TESTBOX, another network is created by adding another IP to the interfaces *tap01* of HOST and *eth0* of TESTBOX. These IPs are kept in the parameters ADMIN_IP and ACCESS_IP. This additional network is needed because, for security reasons, shell access to TESTBOX is forbidden for the normal networks (WAN or LAN).

The configuration file network.cfg is usually included in all the scripts that deal with the configuration of the network: setting IP-s, adding routes, default routes (gateways), etc. These scripts are network.sh, tap01-setup.sh, tap02-setup.sh, tap1-setup.sh, tap2-setup.sh, etc.

The script network.sh is used to configure the interface *eth0* of the HOST and the routes in the HOST.

```
#!/bin/bash

### include the config file
cd $(dirname $0)
.  network.cfg

### setup eth0
/sbin/ip link set dev eth0 up
/sbin/ip address flush dev eth0
/sbin/ip address add $HOST_WAN_IP dev eth0

### add a default route
/sbin/ip route add to default via $HOST_GW

### setup different routing (gateway) for packets
### coming from clients (out of interfaces tap1 and tap2);
### these packets are marked with the mark 10 by the firewall
sed -e "/201  clients/d" -i /etc/iproute2/rt_tables
echo "201  clients" >> /etc/iproute2/rt_tables

/sbin/ip rule del fwmark 10 table clients 2> /dev/null
/sbin/ip rule add fwmark 10 table clients
/sbin/ip route del default via $HOST_GW dev eth0 \
                                        table clients 2> /dev/null
/sbin/ip route add default via $HOST_GW dev eth0 table clients
```

The scripts tapX-setup.sh are used to configure the interfaces tap01, tap02, tap1 and tap2 of the HOST. These interfaces are created by the emulation program (qemu), and the configuration scripts are called by it as well.

tap01-setup.sh:

```
#!/bin/bash

### include the config file
cd $(dirname $0)
.  network.cfg

### setup the interface
/sbin/ip link set tap01 up
/sbin/ip address flush dev tap01
/sbin/ip address add $HOST_LAN_IP1 dev tap01
/sbin/ip address add $ADMIN_IP dev tap01
```

tap02-setup.sh:

```
#!/bin/bash

### include the config file
cd $(dirname $0)
.  network.cfg

### setup the interface
/sbin/ip link set tap02 up
/sbin/ip address flush dev tap02
/sbin/ip address add $HOST_LAN_IP2 dev tap02
```

tap1-setup.sh:

```
#!/bin/bash

### include the config file
cd $(dirname $0)
.  network.cfg

### setup the interface
/sbin/ip link set tap1 up
/sbin/ip address flush dev tap1
/sbin/ip address add $CLIENT1_IP dev tap1
```

tap2-setup.sh:

```
#!/bin/bash

### include the config file
cd $(dirname $0)
```

```
.  network.cfg

### setup the interface
/sbin/ip link set tap2 up
/sbin/ip address flush dev tap2
/sbin/ip address add $CLIENT2_IP dev tap2
```

The emulation of TESTBOX, TESTBOX1 and the testing network is started by the script start-testing.sh. It basically calls the program `qemu` with the appropriate parameters (for creating virtual network cards etc.). It also modifies appropriately the existing firewall of HOST.

start-testing.sh:

```
#!/bin/bash
### Start the emulation of the gateway/router testing environment.
### Note: it should be run by root, so that the firewall can be
modified
###       and so that tapX interfaces can be created.

### make sure that the script is called by root
if [ $(whoami) != "root" ]; then echo "Should be called by root";
exit; fi

### get the system to be tested
testbox=${1:-images/netaccess.img}

### go to this directory
cd $(dirname $0)

### accept in firewall tapX and enable source NAT
./firewall.sh enable

### emulate a testbox with three network cards
qemu -kernel-kqemu -m 64 -serial /dev/tty8 $testbox \
-net nic,vlan=0  -net nic,vlan=1  -net nic,vlan=2  \
-net socket,vlan=0,listen=:1234 \
-net tap,vlan=0,ifname=tap01,script=./tap01-setup.sh \
-net tap,vlan=1,ifname=tap1,script=./tap1-setup.sh   \
-net tap,vlan=2,ifname=tap2,script=./tap2-setup.sh &

### configure a route to ip 10.0.0.1 of tap01
sleep 10
./network.sh

### wait until the first testbox boots up
sleep 30
```

```
### emulate a second testbox with two network cards,
### which will be gateway for the first testbox
qemu -kernel-kqemu -m 64 images/wan.img \
-net nic,vlan=0,macaddr=52:54:00:12:34:60 \
-net nic,vlan=1,macaddr=52:54:00:12:34:61 \
-net tap,vlan=0,ifname=tap02,script=./tap02-setup.sh \
-net socket,vlan=1,connect=127.0.0.1:1234

### close the firewall for tapX and disable source NAT
./firewall.sh disable
```

The modification of the firewall is done/undone (enabled/disabled) by calling the script firewall.sh. This script inserts in firewall rules for accepting input from the interfaces *tap01*, *tap02*, *tap1* and *tap2*. It also enables forwarding for the packets coming from the interface *tap01* and *tap02* and enables source NAT (masquerading) for the packets that are forwarded through *eth0*, since it should serve as a gateway for the TESTBOX and TESTBOX1.

firewall.sh:

```
#!/bin/bash
### accept/not-accept input from tapX and
### enable or disable source NAT (masquerading) for tap0

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

### check that there is one parameter and get it
if [ -z $1 ]; then usage; fi
ACTION=$1

### include the config file
cd $(dirname $0)
.  network.cfg

### variables
WAN_TAPS="tap01 tap02"
LAN_TAPS="tap1 tap2"
ALL_TAPS="$WAN_TAPS $LAN_TAPS"
IPT=/sbin/iptables

function insert_rules
{
```

```
  ### mark packets comming from wan tap interfaces
  for tap in $WAN_TAPS
  do
    $IPT --table mangle --insert PREROUTING \
         --in-interface $tap --jump MARK --set-mark 10
  done

### accept input from the tap interfaces
for tap in $ALL_TAPS
do
  $IPT --table filter --insert INPUT --in-interface $tap \
       --jump ACCEPT
done

### accept forward from the wan tap interfaces
for tap in $WAN_TAPS
do
  $IPT --table filter --insert FORWARD --in-interface $tap \
       --jump ACCEPT
done

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

### enable source NAT
/sbin/iptables --table nat --insert POSTROUTING \
         --out-interface eth0 --jump MASQUERADE
}

function delete_rules
{
  ### del marking rules for packets coming from wan tap interfaces
  for tap in $WAN_TAPS
  do
    $IPT --table mangle --delete PREROUTING \
         --in-interface $tap --jump MARK --set-mark 10
  done

  ### delete input rules for tap interfaces
  for tap in $ALL_TAPS
  do
    $IPT --table filter --delete INPUT --in-interface $tap \
         --jump ACCEPT
  done

  ### delete forward rules for wan tap interfaces
  for tap in $WAN_TAPS
  do
```

```
    $IPT --table filter --delete FORWARD --in-interface $tap \
         --jump ACCEPT
  done

  ### disable forwarding in the kernel
  echo 0 > /proc/sys/net/ipv4/ip_forward

  ### disable source NAT
  /sbin/iptables --table nat --delete POSTROUTING \
                 --out-interface eth0 --jump MASQUERADE
}

### list the existing rules
function list-rules
{
  /sbin/iptables-save | grep -E 'MASQUERADE|tap'
}

case $ACTION in
enable  )  insert_rules ;;
disable )  delete_rules ;;
*       )  list-rules ;;
esac
```

After the emulation of the testboxes and the testing network has been started, the HOST system can be placed in several roles inside this network (as it has been discussed above), in order to be able to test better the TESTBOX. Switching of the roles is done by calling the script set-role.sh.

set-role.sh:

```
#!/bin/bash
### Sets the role of the main computer
### (by changing the default route etc.).

### make sure that the script is called by root
if [ $(whoami) != "root" ]; then echo "Should be called by root";
exit; fi

### include the config file
cd $(dirname $0)
.  network.cfg

function usage()
{
  echo "Usage: $0 [ls | normal | client1 | client2]"
  echo "Sets the role of the HOST (by changing the default route)."
  exit 0
```

```
}

function show_current_role
{
  gateway=$(/sbin/ip route | grep default | cut -d' ' -f3)
  case $gateway in
    $HOST_GW ) echo "normal" ;;
    $LAN1_GW ) echo "client1" ;;
    $LAN2_GW ) echo "client2" ;;
    * )        echo "unknown, gateway=$gateway" ;;
  esac
}

function set_normal
{
  ### delete the table clients
  /sbin/ip route flush table clients

  ### change the default route
  /sbin/ip route del default 2> /dev/null
  /sbin/ip route add default via $HOST_GW

  ### change resolv.conf as well
  echo "nameserver $HOST_GW" > /etc/resolv.conf
}

function set_client1
{
  ### copy all routes of eth0 to table 'clients'
  /sbin/ip route flush table clients
  /sbin/ip route show | grep eth0 \
    | while read ROUTE ; do /sbin/ip route add $ROUTE table clients
; done

  ### change the default routes
  /sbin/ip route del default 2> /dev/null
  /sbin/ip route add default via $LAN1_GW
  /sbin/ip route del default table clients 2> /dev/null
  /sbin/ip route add default via $HOST_GW table clients

  ### change resolv.conf as well
  echo "nameserver $LAN1_GW" > /etc/resolv.conf
}

function set_client2
{
  ### copy all routes of eth0 to table 'clients'
  /sbin/ip route flush table clients
```

```
  /sbin/ip route show | grep eth0 \
    | while read ROUTE ; do /sbin/ip route add $ROUTE table clients
; done

  ### change the default routes
  /sbin/ip route del default 2> /dev/null
  /sbin/ip route add default via $LAN2_GW
  /sbin/ip route del default table clients 2> /dev/null
  /sbin/ip route add default via $HOST_GW table clients

  ### change resolv.conf as well
  echo "nameserver $LAN2_GW" > /etc/resolv.conf
}


if [ "$1" = "" ]; then usage; fi
role=$1

case $role in
  ls      )  show_current_role ;;
  normal  )  set_normal ;;
  client1 )  set_client1 ;;
  client2 )  set_client2 ;;
  *       )  usage ;;
esac
```

When HOST is set in the role of, say, CLIENT1, then its default gateway is changed to LAN1_GW (through tap1) and its nameserver is changed to TESTBOX. So, HOST now, being in stead of CLIENT1, accesses the Internet using TESTBOX as a gateway. On the other hand, TESTBOX itself accesses the Internet using HOST as a gateway. Doesn't it sound like an infinite loop?

In order to avoid this infinite loop, some special configurations are done. We want to make sure that all the packets that come through the interfaces tap01/tap02 of HOST are always sent out to the Internet through the interface eth0 of HOST (using HOST_GW as gateway), no matter what is their source (TESTBOX or CLIENT1). This would break the network loop.

This is implemented by marking (in the firewall) all the packets that come through the interfaces tap01/tap02, and then using a different routing table for the marked packets, which always has HOST_GW as gateway no matter what is the role of HOST. See the scripts firewall.sh and network.sh for the details of how this is done.

6.Configuration of Services

Let us discuss some details about the configuration of network and services in Scarlet (sshd, dns, dhcpd, httpd, mysql, squid, etc). We can find out these configuration details by looking in the directory /var/www/htdocs/scarlet/server-config/ and in its subdirectory templates/ .

All the configuration parameters are saved in a single file, gateway.cfg :

```
DOWNLOAD_LIMIT=750
UPLOAD_LIMIT=300
LAN_IP1=192.168.10.1/24
LAN_IP2=
LAN_IP3=
ENABLE_DHCPD=false
ENABLE_SQUID=true
OPEN_PORTS=true
PORT_LIST="21 25 53 80 110 119 222 443 1506 1507 1863 1972 2222
4899 5061 5050 5100 6667 9000 9010"
DNS1=192.168.0.1
DNS2=192.168.100.1
DNS3=
G2_PORTS=""
GATEWAY1=192.168.0.1
GATEWAY2=192.168.100.1
WAN_IP1=192.168.0.250/24
WAN_IP2=192.168.100.250/24

DOMAIN=localnet.net
ACCESS_IP=10.0.0.1/24
```

This file can be modified manually, if we login with ssh. It can also be modified from the *admin web interface* of Scarlet. This configuration file is included in all the configuration scripts that use configuration parameters.

After something is modified in gateway.cfg the script reconfig.sh must be called, in order to apply the new settings:

```
#!/bin/bash
### reconfigure the network after changing
### any configuration variables

### go to this directory
cd $(dirname $0)

### update the configuration files
./update-config-files.sh

### configure the server
./configure.sh
```

The script update-config-files.sh modifies the configuration files of the server, so that the new configuration settings are used:

```
#!/bin/bash
### modify the configuration files after changing
### any configuration variables

### go to this directory
cd $(dirname $0)

### include the configuration file
. gateway.cfg

### change the resolv.conf files
echo "nameserver $DNS1" > /etc/resolv.conf
echo "nameserver $DNS1" > /etc/resolv.dnsmasq.conf
if [ -n "$DNS2" ]
then
  echo "nameserver $DNS2" >> /etc/resolv.conf
  echo "nameserver $DNS2" >> /etc/resolv.dnsmasq.conf
fi
if [ -n "$DNS3" ]
then
  echo "nameserver $DNS3" >> /etc/resolv.conf
  echo "nameserver $DNS3" >> /etc/resolv.dnsmasq.conf
fi

### update /etc/hosts
LOCAL_IP=${LAN_IP1%/*}
sed -e "s/|DOMAIN|/$DOMAIN/g" \
    -e "s/|LOCAL_IP|/$LOCAL_IP/g" \
    templates/hosts > /etc/hosts

### change the configuration of dnsmasq
LAN_NETWORK=${LAN_IP1%.*/*}
sed -e "s/|DOMAIN|/$DOMAIN/g"  \
    -e "s/|LAN_NETWORK|/$LAN_NETWORK/g"        \
    templates/dnsmasq.conf > /etc/dnsmasq.conf
if [ "$ENABLE_DHCPD" = "false" ]
then
  ### comment any DHCP settings
  sed -e "s+^dhcp+#dhcp+" -i /etc/dnsmasq.conf
fi

### change the configuration of HTTPD
APP_PATH=$(dirname $(pwd))
sed -e "s#|APP_PATH|#$APP_PATH#g"  \
    templates/httpd.conf > /etc/apache/httpd.conf

### change the configuration of proxy (squid)
LAN1=${LAN_IP1%.*/*}.0/24
```

```
OUR_NETWORKS="$LAN1"
sed -e "s+|OUR_NETWORKS|+$OUR_NETWORKS+g"  \
    templates/squid.conf > /etc/squid/squid.conf
```

The configuration files are modified by using the templates in the subdirectory templates/. Using `sed` we replace any template variables with the values that are taken from gateway.cfg. The template variables are denoted by starting and ending them with vertical bars, like this: `|VAR_NAME|` . After the variables are replaced, the configuration file is written to the proper place.

The configuration files that are modified are these:

- /etc/resolv.conf
- /etc/resolv.dnsmasq.conf
- /etc/hosts
- /etc/dnsmasq.conf
- /etc/apache/httpd.conf
- /etc/squid/squid.conf

What modifications are made in these files and how they are different from the original (Slackware) configuration files can be checked on templates/*.diff, which are also appended in the appendix.

7.Network Configuration

Scarlet can have two network gateways (network connections). The configuration of these two connections is done using the parameters GATEWAY1, GATEWAY2, WAN_IP1, WAN_IP2 of the configuration file gateway.cfg . The parameter G2_PORTS has a list of ports that are routed through GATEWAY2. The rest of the network traffic is done through GATEWAY1. This is some kind of load balancing between the network connections. In case that one of the gateways fails, then the other one is used automatically. When the failed connection is back again, then the network traffic is adjusted again so that both of the connections are used again. This is automatic backup connection (when a connection fails, the network traffic is switched to the other one automatically).

The network configuration of scarlet is done by the script network.sh (which is called by configure.sh):

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

### go to this dir
cd $(dirname $0)
```

```
### variables
IP=/sbin/ip

### include the configuration file
. ./gateway.cfg

### set up the links, in  case that they are not up
$IP link set eth0 up
$IP link set eth1 up

### flush any existing ip addresses
$IP address flush eth0
$IP address flush eth1

### flush any existing routes of table main
$IP route flush table main
### add new addresses
$IP address add $WAN_IP1 dev eth0
$IP address add $WAN_IP2 dev eth0
$IP address add $LAN_IP1 dev eth1

### copy the routes of the main table to another routing table
$IP route flush table 2
$IP route show table main | grep -Ev ^default \
  | while read ROUTE ; do $IP route add table 2 $ROUTE ; done

### add the default routes
./set-default-routes.sh $GATEWAY1 $GATEWAY2

### use the second routing table for the packets marked with 2 by
the firewall
$IP rule del fwmark 2 table 2   2>/dev/null
$IP rule add fwmark 2 table 2

### add the access IP to the external (WAN, eth0) interface
$IP address add $ACCESS_IP dev eth0

$IP route flush cache
```

Notice that the script creates two routing tables: the t*able main* and the *table 2*. When the addresses are added to the interfaces some routes are added implicitly to the main routing table. The script copies all these routes (except the default route) to the second table. Then it sets a different gateway (default route) for each routing table. Finally, a rule is added which says that we should use the second routing table for all the packets that are marked by 2 from the firewall (and the main table for the rest of packets). Since the second table has a different gateway than the first one, then these packets go to the Internet passing through the second gateway.

The script firewall/mark-rules.sh builds the iptables' rules that decide which packets are marked by 2:

```
#!/bin/bash
### Mark with 2 the packets that should be routed through the
gateway2.

### get variable $G2_PORTS from the configuration file
. ../gateway.cfg

IPT='/usr/sbin/iptables --table mangle'

$IPT --new-chain MARK-RULES
$IPT --append PREROUTING --jump MARK-RULES
$IPT --append OUTPUT     --jump MARK-RULES

for PORT in $G2_PORTS
do
  $IPT --append MARK-RULES --match tcp --protocol tcp \
       --destination-port $PORT --jump MARK --set-mark 0x2
done
```

It creates the chain of rules MARK-RULES in the table *mangle*, which is used in both PREROUTING and OUTPUT. Then, for each port in G2_PORTS a new rule is added in the chain MARK-RULES, which sets the mark 2 to the tcp packets that have such a port. The configuration parameter G2_PORTS is defined in gateway.cfg and it is a list of space separated port numbers.

The default routes are set by the script set-default-routes.sh (which is called by network.sh):

```
#!/bin/bash
### (re)configure the default routes
### if only one parameter is given, then it is used for both
gateways
### (gateway1 and gateway2 in this case are the same)

if [ "$1" = "" ]
then
  echo "Usage: $0 gateway1 [gateway2]"
  echo "If only gateway1 is given, then it is used as the second
one as well"
  exit 1
fi

GATEWAY1=$1
GATEWAY2=${2:-$GATEWAY1}

### change the default route of the main table
```

```
/sbin/ip route del default  2> /dev/null
/sbin/ip route add default via $GATEWAY1

### change the default route of the second routing table
/sbin/ip route del table 2 default  2> /dev/null
/sbin/ip route add table 2 default via $GATEWAY2

### delete the cache of routes
/sbin/ip route flush cache
```

In order to check for the availability of the gateways, the script watch.sh is used:

```
#!/bin/bash
### Check periodicly the status of the gateways (whether they
### are dead or alive) and modify the network configuration
### accordingly, so that if one line goes down it is backup-ed
### by the other one.

### go to this directory
cd $(dirname $0)

### read the config file
. gateway.cfg

### ping each gateway and find out whether it is on or off
function check-status
{
  if  ping -r -I eth0 -c 2 -W 1 $GATEWAY1 > /dev/null 2> /dev/null
  then G1=on ; else G1=off
  fi

  if  ping -r -I eth0 -c 2 -W 1 $GATEWAY2 > /dev/null 2> /dev/null
  then G2=on ; else G2=off
  fi
}

### compare the status with the status of the previous check,
### return true if the status of the gateways has changed
function status-changed
{
  test "$G1" != "$G_1" -o "$G2" != "$G_2"
}

### change the default routes and the SNAT rules of the firewall
function change-network-config
{
  #echo $G1 -- $G2  # debug
```

```
  IP1=${WAN_IP1%/*}
  IP2=${WAN_IP2%/*}

  if [ "$G1" = "on" ]
  then
    gateway1=$GATEWAY1
    ip1=$IP1
  else
    gateway1=''
    ip1=''
  fi

  if [ "$G2" = "on" ]
  then
    gateway2=$GATEWAY2
    ip2=$IP2
  else
    gateway2=''
    ip2=''
  fi

  ### if at least one of the gateways is on
  ### then change the config of the default routes and SNAT
  if [ "$G1" = "on"  -o  "$G2" = "on" ]
  then
    ./set-default-routes.sh $gateway1 $gateway2
    firewall/source-nat.sh enable eth0 $ip1 $ip2
  fi
}

### save the current status of the gateways
### in order to compare them in the next check
function save-current-status
{
  G_1=$G1
  G_2=$G2
}

### start an endless loop that periodicly checks the status of the
interfaces,
### and if it is changed, updates the network config accordingly
while true
do
  check-status

  if status-changed
  then
```

```
    change-network-config
    save-current-status
  fi

  sleep 6   # wait for a while
done
```

This script checks periodically the status of the gateways, and if there is any change (gateway online or offline), it changes the configuration of the default routes (by calling set-default-routes.sh).

Note: *Actually, just pinging the IP of the gateway is not enough for deciding whether the network connection through this gateway is OK or not. The network connection may be broken somewhere behind the gateway. This remains to be fixed and improved in the future.*

The LAN computers are connected to the network using SNAT (Source Network Address Translation). The script that manages SNAT rules is in the file firewall/source-nat.sh :

```
#!/bin/bash
### Enable or disable source NAT (masquerading).
### If two IPs are given, then the second one is used for
### the packets marked with 2, and the first one for the rest.
### If only one IP is given, then the second one is taken to be
### equal to the first one. If no IPs at all are given, then
### MASQUERADING is used instead.

IPT="/usr/sbin/iptables --table nat"

function usage
{
  echo "Usage: $0 [enable|disable|list] [interface] [ip1] [ip2]"
  echo "    ip1 is used for the default traffic"
  echo "    ip2 is used for the packets that have the mark 2"
  echo "    If ip2 is missing, then ip1 is used as ip2."
  echo "    If both ip-s are missing, then masquerading is used
instead."
  exit
}

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

ACTION=$1
OUT_IF=${2:-eth0}
IP1=$3
IP2=${4:-$IP1}

### enable forwarding in the kernel
```

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

### append snat rules
function append-snat-rules
{
  delete-snat-rules 2>/dev/null

  $IPT --new-chain SNAT-RULES
  $IPT --append POSTROUTING --out-interface $OUT_IF \
       --jump SNAT-RULES

  if [ -z $IP1 ]
  then
    $IPT --append SNAT-RULES --jump MASQUERADE
  else
    $IPT --append SNAT-RULES \
         --match mark --mark 0x2 --jump SNAT --to-source $IP2
    $IPT --append SNAT-RULES --jump SNAT --to-source $IP1
  fi
}

### delete snat rules
function delete-snat-rules
{
  $IPT --delete POSTROUTING --out-interface $OUT_IF \
       --jump SNAT-RULES
  $IPT --flush SNAT-RULES
  $IPT --delete-chain SNAT-RULES
}

### list the existing SNAT rules
function list-snat-rules
{
  /usr/sbin/iptables-save --table nat | grep 'SNAT-RULES'
}

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

A new iptables chain, SNAT-RULES is created in the table *nat*. Then, if the packet is marked by 2, then its source address is set to IP2, otherwise it is set to IP1. If IP2 is not given, then IP1 is used for both marked and unmarked packets. If no IPs are given at all, then MASQURADING is used instead.

8.Managing Ports

In the configuration file gateway.cfg there are two variables about managing ports:

```
OPEN_PORTS=true
PORT_LIST="21 25 53 80 110 119 222 443 1506 1507 1863 1972 2222
4899 5061 5050 5100 6667 9000 9010"
```

These correspond to the section *Ports* of *Settings* in the web interface (see the screenshots in the appendix).

If OPEN_PORTS is true, then all the ports are closed, except for the ones that are listed in the PORT_LIST. If OPEN_PORTS is closed, then all the ports are closed, except for the ones that are listed in the PORT_LIST. This logic is implemented by the script manage-ports.sh :

```
#!/bin/bash
### Manage ports, either allow all except the listed ones
### or forbid all except the listed ones.

IPT='/usr/sbin/iptables --table filter'

### get variables $PORT_LIST, $OPEN_PORTS
. ../gateway.cfg

function allow-ports
{
  for PORT in $PORT_LIST
  do
    $IPT --append CHECK-PORTS --match tcp --protocol tcp \
         --destination-port $PORT --jump RETURN
    $IPT --append CHECK-PORTS --match udp --protocol udp \
         --destination-port $PORT --jump RETURN
  done
  $IPT --append CHECK-PORTS --jump REJECT
}

function forbid-ports
{
  for PORT in $PORT_LIST
  do
    $IPT --append CHECK-PORTS --match tcp --protocol tcp \
         --destination-port $PORT --jump REJECT
    $IPT --append CHECK-PORTS --match udp --protocol udp \
         --destination-port $PORT --jump REJECT
  done
  $IPT --append CHECK-PORTS --jump RETURN
}

### create the chain CHECK-PORTS and jump to it
### for each packet that is forwarded out through eth0
```

```
$IPT --new-chain CHECK-PORTS
$IPT --insert FORWARD --out-interface eth0 \
     --match tcp --protocol tcp --jump CHECK-PORTS
$IPT --insert FORWARD --out-interface eth0 \
     --match udp --protocol udp --jump CHECK-PORTS

### call one of the functions allow-ports or forbid-ports,
### depending on the value of $OPEN_PORTS
if [ "$OPEN_PORTS" = "true" ] ; then allow-ports ; else forbid-
ports ; fi
```

In order to manage the ports, the chain CHECK-PORTS is created in the table *filter* and is used for all the packets that are sent out from the LAN through the WAN interface (*eth0)*. When PORT_LIST contains the ports that should be allowed, then the chain CHECK-PORTS contains a RETURN for each port in the list (which means that the packet will continue to be processed by the firewall), and at the end of the chain there is a REJECT for all the ports that are not in the list. When PORT_LIST contains the ports that should be blocked, then the chain contains a REJECT for each port in the list, and at the end of the chain there is a RETURN for all the ports that are not in the list, which means that they will continue to be processed by the firewall.

9.Protecting the Gateway Server

The directory firewall/ contains the scripts that are related to *iptables*. Actually not all of the *iptables* scripts are used to build a firewall, most of them are used to manage connections.

The rules of *iptables* are constructed by the script iptables.sh :

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

### go to this directory
cd $(dirname $0)

### include network configuration variables
. ../gateway.cfg

### update the list of allowed macs from DB
./get-allowed-macs.sh

IPT=/usr/sbin/iptables

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

```
$IPT --table nat    --delete-chain
$IPT --table mangle --flush
$IPT --table mangle --delete-chain

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

### set a mark to each packet that goes to internet
./mark-rules.sh

### forward ports
./forward-ports.sh

### redirect some ports (like 53, 80, etc.) to localhost
./redirect.sh

### enable SNAT-ing
IP1=${WAN_IP1%/*}
IP2=${WAN_IP2%/*}
./source-nat.sh enable eth0 $IP1 $IP2
#./source-nat.sh enable eth0

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

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

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

First of all, it cleans any existing rules, before constructing the new ones.

The script iptables-stop.sh can be used as well to clear all the *iptables* rules:

```
#!/bin/bash
### clear all the iptables rules

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

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

```
$IPT --table mangle --flush

### set default policies to ACCEPT
$IPT --table filter --policy INPUT ACCEPT
$IPT --table filter --policy OUTPUT ACCEPT
$IPT --table filter --policy FORWARD ACCEPT
```

Since the clients on the LAN use SNAT to access the Internet, they are automatically protected from external access, because no body can access them directly. However the gateway server itself needs to be protected both from external (WAN) and internal (LAN) sides. These protecting rules are placed on the chain INPUT of iptables, which checks all the connections that come in to the server.

The rules of the INPUT chain are constructed by the script input-rules.sh :

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

### include network configuration variables
. ../gateway.cfg

IPT='/usr/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 chains WAN and LOCALNET for the connections
### comming from the WAN interface and from the LAN interfaces
$IPT --new-chain WAN
$IPT --new-chain LOCALNET
$APPEND --in-interface eth0 --jump WAN
$APPEND --in-interface ! eth0 --jump LOCALNET

### add the rules for these chains
./wan-rules.sh
./localnet-rules.sh

### reject anything else
```

```
$APPEND --jump REJECT --reject-with icmp-host-prohibited
```

A new chain, named WAN, is created and used for checking the connections that come from the external interface (*eth0*), and another one, named LOCALNET, for the connections that come from the LANs (internal interfaces, *eth1* and *eth2*). The rules of these chains are constructed by the scripts wan-rules.sh and localnet-rules.sh.

wan-rules.sh :

```
#!/bin/bash
### The rules of the chain WAN, which are applied to input from
eth0,
### which is connected to the WAN network.

IPT='/usr/sbin/iptables --table filter'
APPEND="$IPT --append WAN"

### accept httpd (80, 8080)
./port.sh 80 accept WAN
./port.sh 8080 accept WAN

### accept ssh (port 2222)
. ../gateway.cfg
ACCESS_IP=${ACCESS_IP%/*}
$APPEND --match state --state NEW --destination $ACCESS_IP \
        --match tcp --protocol tcp --destination-port 2222 \
        --jump ACCEPT

### 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!
```

localnet-rules.sh :

```
#!/bin/bash
### The rules of the chain LOCALNET.

IPT='/usr/sbin/iptables --table filter'
```

```
APPEND="$IPT --append LOCALNET"

### accept port 67 (dhcp)
./port.sh 67 accept LOCALNET tcp-udp

### accept port 53 (DNS)
./port.sh 53 accept LOCALNET tcp-udp

### accept port 80 (http)
./port.sh 80 accept LOCALNET

### accept port 3128 (proxy/squid)
./port.sh 3128 accept LOCALNET

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

### 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!
```

In order to accept or deny certain ports, the script port.sh is used:

```
root@scarlet# ./port.sh
Usage: ./port.sh [port] [accept|deny|status] [chain] [proto]
    where port is a number (like 80) or a service name (like http)
    chain is: INPUT, FORWARD, LOCALNET, WAN, etc.
    and proto is: tcp or udp or tcp-udp (default is tcp)
```

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] [proto]
    where port is a number (like 80) or a service name (like http)
    chain is: INPUT, FORWARD, LOCALNET, WAN, etc.
    and proto is: tcp or udp or tcp-udp (default is tcp)"
  exit
}

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

PORT=$1
ACTION=$2
CHAIN=${3:-INPUT}
PROTO=${4:-tcp}

## appends or deletes a rule, according to the parameter
function iptables-rule
{
  command=$1
  IPT=/usr/sbin/iptables

  if [ $PROTO = "tcp" -o $PROTO = "tcp-udp" ]
  then
    $IPT --table filter --$command $CHAIN \
         --match state --state NEW \
         --match tcp --protocol tcp --destination-port $PORT \
         --jump ACCEPT
  fi

  if [ $PROTO = "udp" -o $PROTO = "tcp-udp" ]
  then
    $IPT --table filter --$command $CHAIN \
         --match udp --protocol udp --destination-port $PORT \
         --jump ACCEPT
  fi
}

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

This script adds an ACCEPT rule for the port that should be accepted. If the port should be denied, then it is sufficient to delete the ACCEPT rule that was added previously, since the default policy is to deny everything that is not explicitly accepted.

There is also a script for accepting samba ports (in case that we want to share some folders on the server with the local network):

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='/usr/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
./port.sh 139 accept SAMBA
./port.sh 445 accept SAMBA

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

10.Managing Clients

The PCs on the LAN (clients) can connect to the Internet only if their MAC address is registered with Scarlet and is in the list of the allowed MACs. The list of the allowed MACs is in the file scarlet/server-config/allowed-macs , one MAC per line, like this:

```
52:54:00:12:34:56
00:0a:e4:c5:56:28
00-16-6f-81-5c-9c
00:FF:A0:4C:51:31
```

This list of macs is retrieved from the database of the web application by the script get-allowed-macs.sh :

```
#!/bin/bash
### get a list of the macs that are allowed to connect to internet
### and save them in the file 'allowed-macs'

### go to this directory
cd $(dirname $0)
```

```
### include the DB connection params
. ../../db/vars

### get them and save them in allowed-macs
mysql --host=$host --user=$user --password=$passwd \
      --database $database \
      --execute="select mac from macs where connected='true'" \
      --vertical \
  | grep '^mac:' | cut -d' ' -f2 \
  > allowed-macs
```

The file allowed-macs is used in the script forward-rules.sh (which is called by iptables.sh) :

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

### include network configuration variables
. ../gateway.cfg

IPT='/usr/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

### manage ports (allow or forbid them)
./manage-ports.sh

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

### create the chain ALLOWED-MACS
$IPT --new-chain ALLOWED-MACS
for MAC in $(< allowed-macs)
do
  $APPEND --match mac --mac-source $MAC --jump ALLOWED-MACS
done
### add rules for the chain ALLOWED-MACS
$IPT --append ALLOWED-MACS --jump ACCEPT

### reject anything else
$APPEND --jump REJECT --reject-with icmp-net-unreachable
```

The right place for checking the traffic between the clients (PCs in LAN) and the Internet is the chain FORWARD. The connections that are already established are accepted. Then, for each MAC in the list of allowed MACs, we add a rule that accepts the packets originating from that MAC. Everything else is rejected.

However, the DNS and HTTP requests are handled differently. Scarlet serves as a transparent DNS and HTTP proxy for the LAN, so all the DNS and HTTP packets are redirected to the localhost. This is done by the script redirect.sh (which is called from iptables.sh) :

```
#!/bin/bash
### redirect some ports (like 53, 80, etc.) to localhost

IPT='/usr/sbin/iptables --table nat'

### get the configuration variables
. ../gateway.cfg

### redirect any DNS requests (port 53) to the localhost
$IPT --append PREROUTING --protocol udp --destination-port 53 \
     --jump REDIRECT
$IPT --append PREROUTING --protocol tcp --destination-port 53 \
     --jump REDIRECT

### handle the HTTP requests (port 80)
$IPT --new-chain HTTP
$IPT --append PREROUTING --protocol tcp --destination-port 80 \
     --jump HTTP

### handle HTTP requests of the allowed macs
### in the chain HTTP-ALLOWED-MACS
$IPT --new-chain HTTP-ALLOWED-MACS
for MAC in $(< allowed-macs)
do
  $IPT --append HTTP --match mac --mac-source $MAC \
       --jump HTTP-ALLOWED-MACS
done

### the rules of the chain HTTP-ALLOWED-MACS
if [ "$ENABLE_SQUID" = "false" ]
then
  ### direct connection to internet
  $IPT --append HTTP-ALLOWED-MACS --jump ACCEPT
else
  ### don't use proxy for the connections to the local server
  local_ips="$WAN_IP1 $WAN_IP2 $LAN_IP1 $LAN_IP2 $LAN_IP3"
```

```
  for ip in $local_ips
  do
    local_ip=${ip%/*}  ### remove /24 from the ip
    $IPT --append HTTP-ALLOWED-MACS --destination $local_ip \
         --jump ACCEPT
  done
  ### connection through transparent proxy (squid)
  $IPT --append HTTP-ALLOWED-MACS --protocol tcp \
       --jump REDIRECT --to-port 3128
fi

### redirect HTTP requests of the others (not in HTTP-ALLOWED-MACS)
### to the localhost
$IPT --append HTTP --jump REDIRECT
```

All the packets with destination port 53 (DNS requests) are jumped to REDIRECT, which means that their destination address is changed so that they end up to localhost. This is done in the PREROUTING chain, which is checked and applied before routing is done.

Then a new chain named HTTP is created and all the http packets are send to this chain. In this chain are added rules that match each allowed mac and send such packets to the chain HTTP-ALLOWED-MACS. The HTTP packets that do not originate from the allowed macs will match the last rule of the HTTP chain and will be redirected to the localhost. In the chain HTTP-ALLOWED-MACS we either accept just the packets, if the transparent proxy is not enabled, or we redirect them to the port 3128 (of squid), if the transparent proxy is enabled. However we are careful not to send to squid the requests to the local webserver.

11.Advanced Connection Management

Actually these things are not more complicated that what we have already seen, however they are called “advanced” because they have to be done manually, by calling the appropriate scripts (while the other things are managed either automatically or from the web interface). The reason for this is that they may be needed only in rare cases.

11.1.Port Forwarding

We need to do port forwarding when we have some servers in the LAN (for example a web server), behind the gateway server. Since the gateway server uses SNAT, then the web server cannot be accessed from the Internet. However, if we forward the port 80 to the web server, then all the http requests that go to the gateway server will be forwarded back to the web server. The browser will think that it gets the answer from the gateway server, although actually it is the web server that sends replies to it.

Port forwarding in scarlet can be done by calling the script port-forward.sh :

```
root@scarlet # ./port-forward.sh
Usage: ./port-forward.sh PORT EXT_IP INT_IP SERVER_IP [enable|
```

```
disable|status]
    PORT       is the port that will be forwarded (like 80 or http)
    EXT_IP     is the external IP of the server
    INT_IP     is the IP of through which it will be forwarded
    SERVER_IP  is the IP of the server to which it will be
forwarded
```

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

function usage
{
  echo "Usage: $0 PORT EXT_IP INT_IP SERVER_IP [enable|disable|
status]
    PORT       is the port that will be forwarded (like 80 or http)
    EXT_IP     is the external IP of the server
    INT_IP     is the IP of through which it will be forwarded
    SERVER_IP  is the IP of the server to which it will be
forwarded"
  exit
}

### check that there are at least 4 parameters
if [ $# -lt 4 ]; then usage; fi

### get the parameters and remove any netmask from IP-s
PORT=$1
EXT_IP=${2%/*}
INT_IP=${3%/*}
SERVER_IP=${4%/*}
ACTION=$5


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

  # 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 udp --protocol udp --dport $PORT \
      --jump ACCEPT

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

  ### in case that $INT_IP is not gateway for $SERVER_IP
  $IPT --table nat --$command POSTROUTING \
       --protocol tcp --destination $SERVER_IP --dport $PORT \
       --jump SNAT --to-source $INT_IP
}

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

This script can be called from another script, like this:

forward-ports.sh :

```
#!/bin/bash
### ports that are forwarded to local servers

### include network configuration variables
. ../gateway.cfg

### forward port 443 to 192.168.255.2
./port-forward.sh  443  $WAN_IP1  $LAN_IP1  192.168.255.2  enable
```

11.2.IP Forwarding

In this case we assign an additional IP to the WAN interface of the gateway server. However, all the traffic that comes to this IP is forwarded to another server inside the LAN. So, this server is logically on the WAN network, although physically it is on the LAN network.

```
root@scarlet:# ./ip-forward.sh
Usage: ./ip-forward.sh EXTERNAL_IP INTERNAL_IP SERVER_IP
```

```
    EXTERNAL_IP  is the IP that will be forwarded
    INTERNAL_IP  is the IP through which it is forwarded
    SERVER_IP    is the IP to which it is forwarded
```

ip-forward.sh :

```
#!/bin/bash
### forward an IP to another server

function usage
{
  echo "Usage: $0 EXTERNAL_IP INTERNAL_IP SERVER_IP
    EXTERNAL_IP  is the IP that will be forwarded
    INTERNAL_IP  is the IP through which it is forwarded
    SERVER_IP    is the IP to which it is forwarded"
  exit
}

### check that there are at least 3 parameters
if [ $# -lt 3 ]; then usage; fi

### get the parameters and remove any netmask from IP-s
EXTERNAL_IP=${1%/*}  # IP that will be forwarded
INTERNAL_IP=${2%/*}  # IP through which it is forwarded
SERVER_IP=${3%/*}    # IP to which it is forwarded

IPT='/usr/sbin/iptables'

### accept for forward everything related with $SERVER_IP
$IPT --table filter --append FORWARD \
     --destination $SERVER_IP --jump ACCEPT
$IPT --table filter --append FORWARD \
     --source $SERVER_IP --jump ACCEPT

### forward everything with destination $EXTERNAL_IP to $SERVER_IP
$IPT --table nat --append PREROUTING \
     --destination $EXTERNAL_IP --jump DNAT --to-destination
$SERVER_IP

### in case that $INTERNAL_IP is not gateway for $SERVER_IP
### if this is uncomented, then the internal server does not see
### anything else, except $INTERNAL_IP
#$IPT --table nat --append POSTROUTING \
#     --destination $SERVER_IP --jump SNAT --to-source $INTERNAL_IP

### everything comming from $SERVER_IP should be
```

```
### sent out to internet with source $EXTERNAL_IP
#$IPT --table nat --append POSTROUTING \
#     --out-interface eth0 --source $SERVER_IP \
#     --jump SNAT --to-source $EXTERNAL_IP
```

11.3.Source Forwarding

All the traffic that comes from a certain network can be forwarded to a certain internal server. There may be cases when such a thing is needed.

```
root@scarlet:l# ./source-forward.sh
Usage: ./source-forward.sh SOURCE_NET DESTINATION_IP
    SOURCE_NET       the source network (e.g. 192.168.1.0/25)
    DESTINATION_IP   the destination server
```

source-forward.sh :

```
#!/bin/bash
### forward all the packets coming from a given source network
### to another server

function usage
{
  echo "Usage: $0 SOURCE_NET DESTINATION_IP
    SOURCE_NET       the source network (e.g. 192.168.1.0/25)
    DESTINATION_IP   the destination server"
  exit
}

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

SOURCE_NET=$1            # the source network
DESTINATION_IP=${2%/*}   # the destination server

IPT='/usr/sbin/iptables'

### accept for forward everything comming from the $SOURCE_NET
$IPT --table filter --append FORWARD \
     --source $SOURCE_NET --jump ACCEPT
$IPT --table filter --append FORWARD \
     --destination $SOURCE_NET --jump ACCEPT
```

```
### forward everything with source $SOURCE_NET to $DESTINATION_IP
$IPT --table nat --append PREROUTING \
     --source $SOURCE_NET --jump DNAT --to-destination
$DESTINATION_IP
```

12.Controlling the Traffic (tc)

The scripts that are used for traffic control are located in the directory trafic/ . They basically use the utility /sbin/tc . The script that is used to set up the traffic rules is set-rate-limits.sh :

```
#!/bin/bash
### update the upload and download limits

### go to this dir
cd $(dirname $0)

### get the upload and download rates
. ../gateway.cfg

### set upload limit
if [ "$UPLOAD_LIMIT" = "" ]
then
  ./rate-limit.sh off eth0
else
  ./rate-limit.sh on eth0 $UPLOAD_LIMIT
fi

### set download limit
if [ "$DOWNLOAD_LIMIT" = "" ]
then
  ./rate-limit.sh off eth1
else
  ./rate-limit.sh on eth1 $DOWNLOAD_LIMIT
fi
```

It gets the variables $UPLOAD_LIMIT and $DOWNLOAD_LIMIT from the configuration file ../gateway.cfg and calls the script rate-limit.sh to set these rates.

```
root@scarlet: # ./rate-limit.sh
Usage: ./rate-limit.sh [on | off | ls] [dev] [rate]
       dev   can be eth0, eth1, etc.
       rate  can be 48, 96, etc.

This script sets a rate limit in one of the network interfaces
```

```
(eth0, eth1, etc.). If the the limit is set in the external
interface
then it is an upload limit; if it is set on the LAN interface, then
it is a download limit.
```

rate-limit.sh :

```
#!/bin/bash
### This script sets a rate limit in one of the network interfaces
### (eth0, eth1, etc.). If the the limit is set in the external
interface
### then it is an upload limit; if it is set on the LAN interface,
then
### it is a download limit.

### Set the rate limit to your uplink's *actual* speed, minus a few
percent
### If you have a really fast modem, raise 'burst' a bit.

function usage
{
  echo "Usage: $0 [on | off | ls] [dev] [rate]"
  echo "       dev   can be eth0, eth1, etc."
  echo "       rate  can be 48, 96, etc."
  echo
  echo "This script sets a rate limit in one of the network
interfaces"
  echo "(eth0, eth1, etc.). If the the limit is set in the external
interface"
  echo "then it is an upload limit; if it is set on the LAN
interface, then"
  echo "it is a download limit."
  echo
  exit
}

function set_limit
{
  ### check that the rate (third parameter) is supplied
  if [ "$RATE" = "" ]
  then
    echo
    echo "Please supply a rate as well..."
    echo
    usage
    exit 1
```

```
  fi

  ### delete any existing settings
  /sbin/tc qdisc del dev $DEV root 2>/dev/null >/dev/null

  ### add the HTB qdisc with just one class (used for limiting the
rate)
  /sbin/tc qdisc add dev $DEV root handle 1: htb default 1
  /sbin/tc class add dev $DEV parent 1: classid 1:1 htb rate $
{RATE}kbit burst 5k

  ### add a PRIO qdisc below the HTB (in order to
  ### prioritize interactive traffic)
  /sbin/tc qdisc add dev $DEV parent 1:1 handle 10: prio
  # automatically creates classes 10:1,, 10:2, 10:3

  ### add a SFQ qdisc below each priority class (to handle
connections fairly)
  /sbin/tc qdisc add dev $DEV parent 10:1 handle 11: sfq perturb 10
  /sbin/tc qdisc add dev $DEV parent 10:2 handle 12: sfq perturb 10
  /sbin/tc qdisc add dev $DEV parent 10:3 handle 13: sfq perturb 10

  ### Note: with these classes and queues as above, there should be
  ###       no need for filters.

  ### apply the new rate limit
  #/sbin/tc qdisc add dev $DEV root tbf rate ${RATE}kbit latency
50ms burst 3000
}

### get the arguments
if [ $# -lt 2 ]; then usage; fi
ACTION=$1
DEV=$2
RATE=$3

TC="/sbin/tc qdisc"
case $ACTION in
  on  ) set_limit ;;
  off ) /sbin/tc qdisc del dev $DEV root 2>/dev/null >/dev/null;;
  ls  ) /sbin/tc -s qdisc ls dev $DEV ;;
  *   ) usage ;;
esac
```

This script limits the rate on the given interface. It also creates three priority queues on the interface, where the one with the highest priority is for the interactive traffic, and the one with the most low priority for the bulk traffic.

13.Counting the Traffic

The traffic is counted for each MAC in the LAN and statistics are generated (using RRD tools). The scripts that count the traffic and generate the statistics are located in the directory server-config/traffic-counter/ . Initially an RRD (Round Robin Database) file is created for each MAC that is allowed to access the internet. This is accomplished by the scripts init.sh and rrdinit.sh :

init.sh :

```
#!/bin/bash
### initialize the databases for each allowed mac

rm -rf rrd/
mkdir -p rrd
for mac in $(< ../firewall/allowed-macs)
do
  ./rrdinit.sh $mac
done
```

rrdinit.sh :

```
#!/bin/bash
### Create a RRD for the MAC that is given as argument.
### This database keeps a DS for download and one for upload.
### The interval in which the data is fed into the database is 10
sec.
### and the type of the data is ABSOLUTE (after feeding the data
into
### the database the counters that count upload and download
traffic
### are reseted to 0).
###
### The round robin archives that are kept in the database are
these:
###
### - One that keeps each data that is inputed and has 60 items
###   (time interval 10s, time span 10m.)
###
### - One that consolidates 30 primary values for each item and has
24 items
###   (time interval 300s = 5m, time span 120m = 2h)
###
### - One that consolidates 360 values for each item and has 48
items
###   (time interval 3600s = 1h, time span 48h = 2 days)
###
### - One that averages 360*24=8640 values and has 240 items
###   (time interval 1h*24 = 1day, time span 1day * 240 = 8 months)
```

```
if [ "$1" = "" ]
then
  echo "Usage: $0 mac"
  exit 1
fi

mac=$1
dbname="mac_${mac//[:-]/}"

### go to this directory
cd $(dirname $0)

### don't overwrite an existing database
if [ -f "rrd/$dbname" ] ; then exit ; fi

### calculate start time to be a multiple of 86400 (one day)
now=$(date +%s)
day=86400
start=$(( $now / $day * $day ))

### create a new database
rrdtool create rrd/$dbname \
        --start $start --step 10 \
        DS:download:ABSOLUTE:30:0:U \
        DS:upload:ABSOLUTE:30:0:U \
        RRA:AVERAGE:0.5:1:60 \
        RRA:AVERAGE:0.5:30:24 \
        RRA:AVERAGE:0.5:360:48 \
        RRA:AVERAGE:0.5:8640:240
```

To learn more about RRD and its tools, have a look at its documentation on /usr/share/doc/rrdtool-1.2.8/ . You can also read the man pages (*man rrdtool* , *man rrdtutorial* , etc.)

After initializing the RRD files, we can start counting the traffic for each MAC. Counting is started by the script start.sh :

```
#!/bin/bash

### go to this directory
cd $(dirname $0)

### clean up any pid files
rm -f *.pid
```

```
### start tcpdump
#./tcpdump.sh | ./counter.sh &  # debug
./tcpdump.sh | ./counter.sh | rrdtool - >/dev/null &

### start timer
./timer.sh &
```

Here we see some scripts and commands processing data continuously in a pipeline.

The script tcpdump.sh listens the network traffic on the interface *eth1* and outputs it on a short (concise) format, one line for each packet:

```
#!/bin/bash

### run tcpdump on the LAN interface
/usr/sbin/tcpdump -i eth1 -e -n -nn -N -tt -q &

### save the process id to a file
echo $! > tcpdump.pid

###test
#echo $$ > tcpdump.pid
#while true
#do
#  cat test/tcpdump.output
#  sleep 2
#done
```

The output that is produced by this script looks like this:

```
1245873613.101635 52:54:00:12:34:56 > 00:ff:70:3a:98:18, IPv4,
length 226: 10.0.0.1.2222 > 10.0.0.10.39877: tcp 160

1245873613.102139 52:54:00:12:34:56 > 00:ff:70:3a:98:18, IPv4,
length 226: 10.0.0.1.2222 > 10.0.0.10.39877: tcp 160

1245873613.102781 00:ff:70:3a:98:18 > 52:54:00:12:34:56, IPv4,
length 66: 10.0.0.10.39877 > 10.0.0.1.2222: tcp 0

1245873613.103568 00:ff:70:3a:98:18 > 52:54:00:12:34:56, IPv4,
length 66: 10.0.0.10.39877 > 10.0.0.1.2222: tcp 0
```

This output is fed by the pipeline to the script count.sh , which counts the upload/download bytes for each MAC:

```
#!/bin/bash
```

```
### This script watches the network traffic by reading the output
### of tcpdump through a pipe. It processes this output and
accumulates
### the bytes used for upload/download by each MAC.
### It also receives periodically a sigalarm (14) from the script
### 'timer.sh'. When this signal is received, the number of bytes
that
### is counted for each MAC is saved to a file or to a database.

### create upload/download counters (variables) with value 0
### for each database in the directory 'rrd/'
function init_counters()
{
  db_list=$(ls rrd/)
  for dbname in $db_list
  do
    eval ${dbname}_upload=0
    eval ${dbname}_download=0
  done
}

### save the values of the counters in the databases
function dump_counters()
{
  now=$(date +%s)
  now=$(($now / 10 * 10))
  echo
  for dbname in $db_list
  do
    eval download=\$${dbname}_download
    eval upload=\$${dbname}_upload
    echo update rrd/$dbname $now:$download:$upload
  done

  init_counters
}

### ------------------------------------------------------------

### save the process id to a file
echo $$ > counter.pid

### init counters
init_counters

### call the function dump_counters when the sigalarm is received
trap dump_counters 14
```

```
### get the mac of the LAN interface (eth1)
MY_MAC=$(/sbin/ip addr | sed -n -e '/eth1:/,+1 p' | gawk '/link/
{print $2}')

### read package logs from standard input and process them
while read line
do
  ### get from the line $mac1, $mac2 and $bytes
  mac1=${line:18:17}
  mac2=${line:38:17}
  bytes=${line:70:10}
  bytes=${bytes%%:*}

  ### increment the counters
  if [ "$mac1" = "$MY_MAC" ]
  then
    dbname="mac_${mac2//[:-]/}"
    eval let ${dbname}_download+=$bytes
  elif [ "$mac2" = "$MY_MAC" ]
  then
    dbname="mac_${mac1//[:-]/}"
    eval let ${dbname}_upload+=$bytes
  fi

  ### debug
  #echo "$mac1 > $mac2 $bytes"
done
```

This script keeps two counters for each MAC, one for upload and one for download. Each line that comes from tcpdump.sh is processed and the counters are incremented accordingly. Whenever the script gets the signal 14 (sigalarm), it calls the function *dump_counters* which writes the current values of the counters to RRD files and initializes them again (resets the counters). The signal 14 is sent periodically each 10 seconds by the script timer.sh :

```
#!/bin/bash
### send periodically a sigalarm to the counter

period=10

### save the process id to a file
echo $$ > timer.pid

### get the process id of the counter
while [ "$pid" = "" ]
do
  pid=$(cat counter.pid 2> /dev/null)
done
```

```
### send a sigalarm (14) to the counter periodically
while true
do
  sleep $period
  kill -s 14 $pid
done
```

The files counter.pid, tcpdump.pid and timer.pid keep the process id of the scripts. They are used to send signals to each other and also to stop them. Counting the traffic can be used by the script stop.sh :

```
#!/bin/bash

### stop timer
pid=$(cat timer.pid)
kill -9 $pid

### stop tcpdump
pid=$(cat tcpdump.pid)
kill -9 $pid

### clean up any pid files
rm -f *.pid
```

The script counter.sh dumps the counters to the database by outputting *update* command to the pipeline, which are then read and executed by the *rrdtool* . If we called *rrdtool* directly from the script counter.sh this would be much less efficient, because each time that it is called, it has to be loaded and initialized in RAM, before being executed.

After the statistics have been collected, we can generate graphs using the script rrdgraph.sh :

```
root@scarlet:# ./rrdgraph.sh
Usage: ./rrdgraph.sh fname interval [ mac]+
  Generates the traffic usage of one or more macs in an interval.
  'fname' is the path and name of the image that will be created
  'interval' is: 10minutes | 2hours | 2days | 1month | 8months
  One or more MACs should be supplied, separated by space.
```

rrdgraph.sh :

```
#!/bin/bash
```

```
function usage
{
  echo "Usage: $0 fname interval [ mac]+"
  echo "  Generates the traffic usage of one or more macs in an
interval."
  echo "  'fname' is the path and name of the image that will be
created"
  echo "  'interval' is: 10minutes | 2hours | 2days | 1month |
8months"
  echo "  One or more MACs should be supplied, separated by space."
  exit 1
}

if [[ $# -lt 3 ]] ; then usage ; fi

### go to this directory
cd $(dirname $0)

### get file name and the time interval to be graphed
fname=$1 ; shift
interval=$1 ; shift

### get the resolution
case $interval in
  10minutes )  resolution=10 ;;     # 10 sec
  2hours    )  resolution=300 ;;    # 5 min
  2days     )  resolution=3600 ;;   # 1 hour
  1month    )  resolution=86400 ;;  # 1 day
  8months   )  resolution=86400 ;;  # 1 day
  *         )  usage ;;
esac

### get start and end times
now=$(date +%s)
end=$(($now/$resolution*$resolution))
start="end-$interval"

### process the MACs
nr_macs=0
DEFs=''
upload='0'
download='0'

while [ "$1" != "" ]
do
  let nr_macs++
  mac=$1 ; shift
  mac=${mac//[:-]/}
```

```
  DEFs="$DEFs DEF:upload_$mac=rrd/mac_$mac:upload:AVERAGE"
  DEFs="$DEFs DEF:download_$mac=rrd/mac_$mac:download:AVERAGE"

  upload="upload_$mac,$upload,+"
  download="download_$mac,$download,+"
done

### get the average
#upload="$upload,$nr_macs,/"
#download="$download,$nr_macs,/"

### convert from Bytes/s to Bits/s
upload="$upload,8,*"
download="$download,8,*"

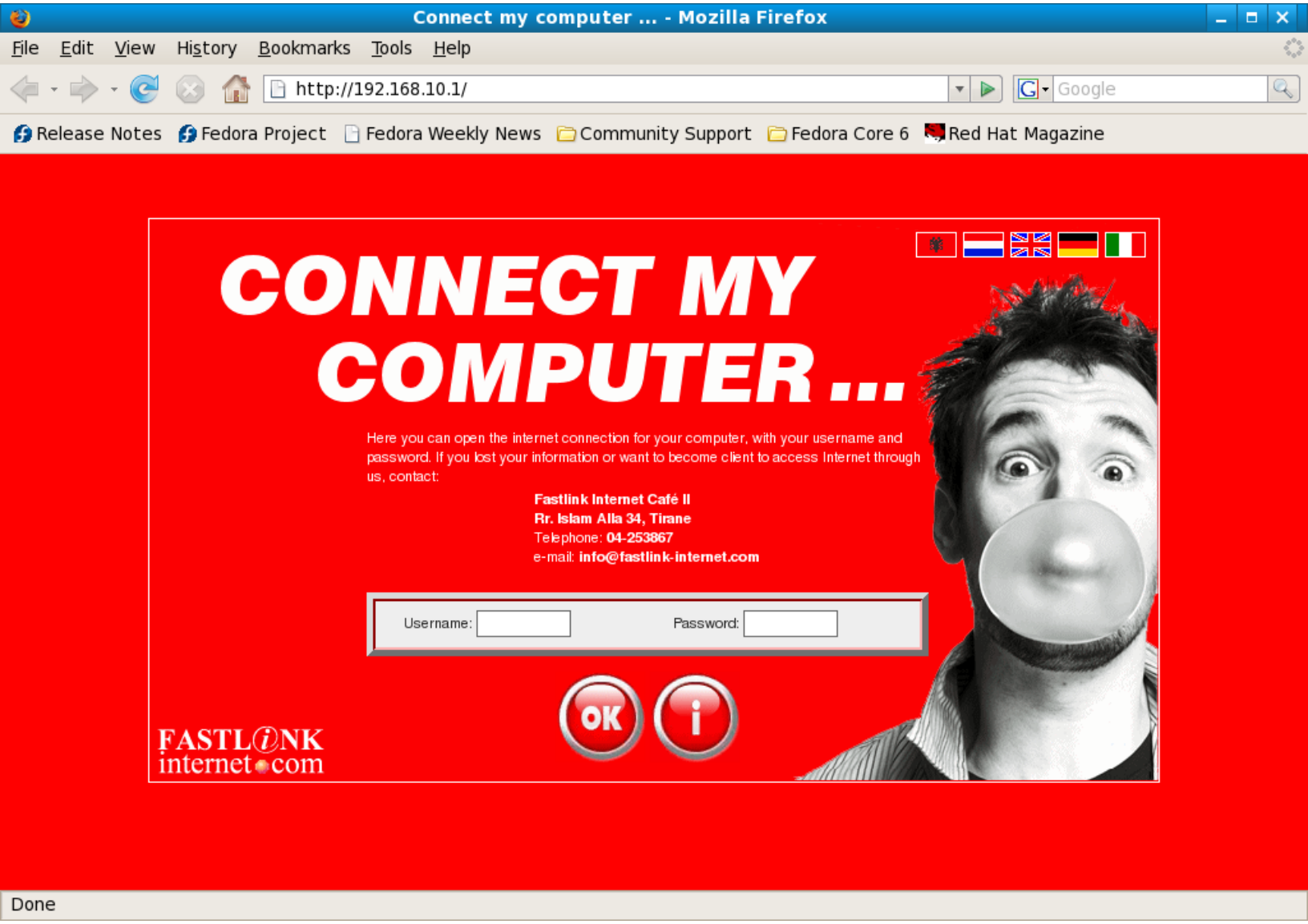
### generate the graphic image
rm -f $fname
#echo \
rrdtool graph $fname \
        --imgformat PNG --interlaced --end $end --start $start \
        --width 550 --vertical-label Bits/s    \
        $DEFs                                  \
        CDEF:upload=$upload                    \
        CDEF:download=$download                \
        AREA:upload#00EEEE:'Upload'            \
        LINE1:upload#00CCAA                    \
        LINE1:download#FF0000:'Download'
```

14.Summary and Future Work

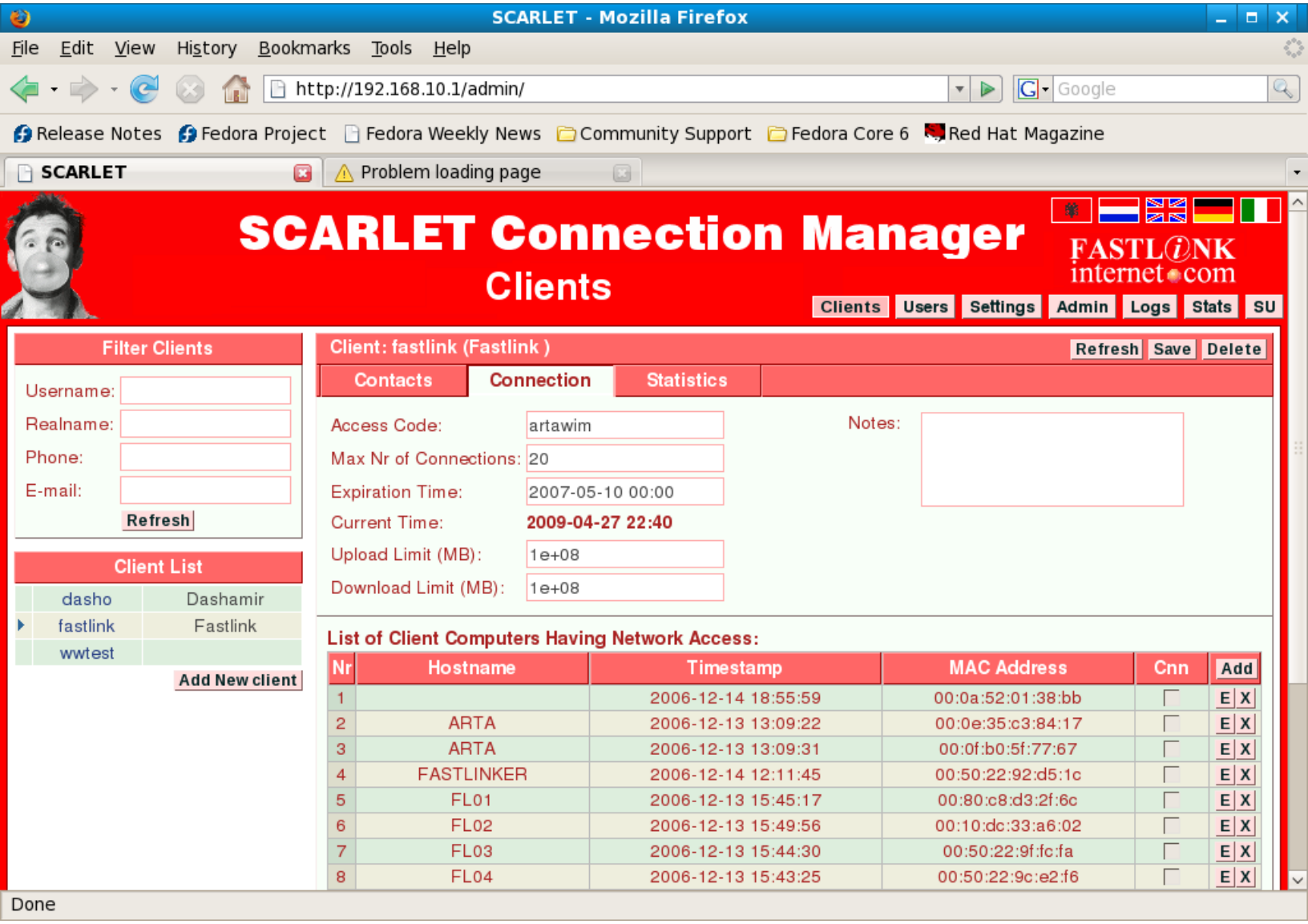
15.Appendix

15.1.Screenshots From SCARLET

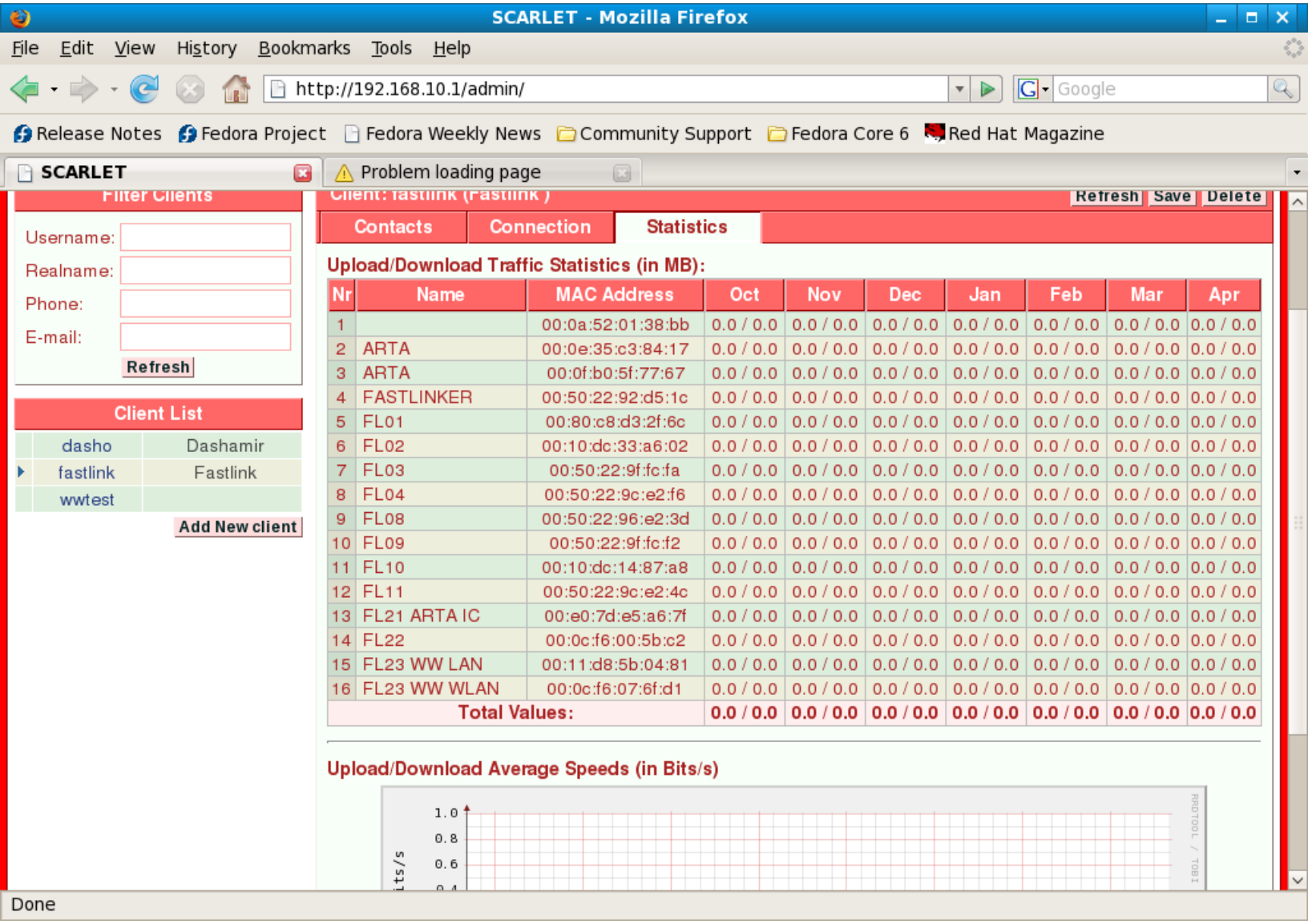
Here are listed some screenshots from the Scarlet web interface.



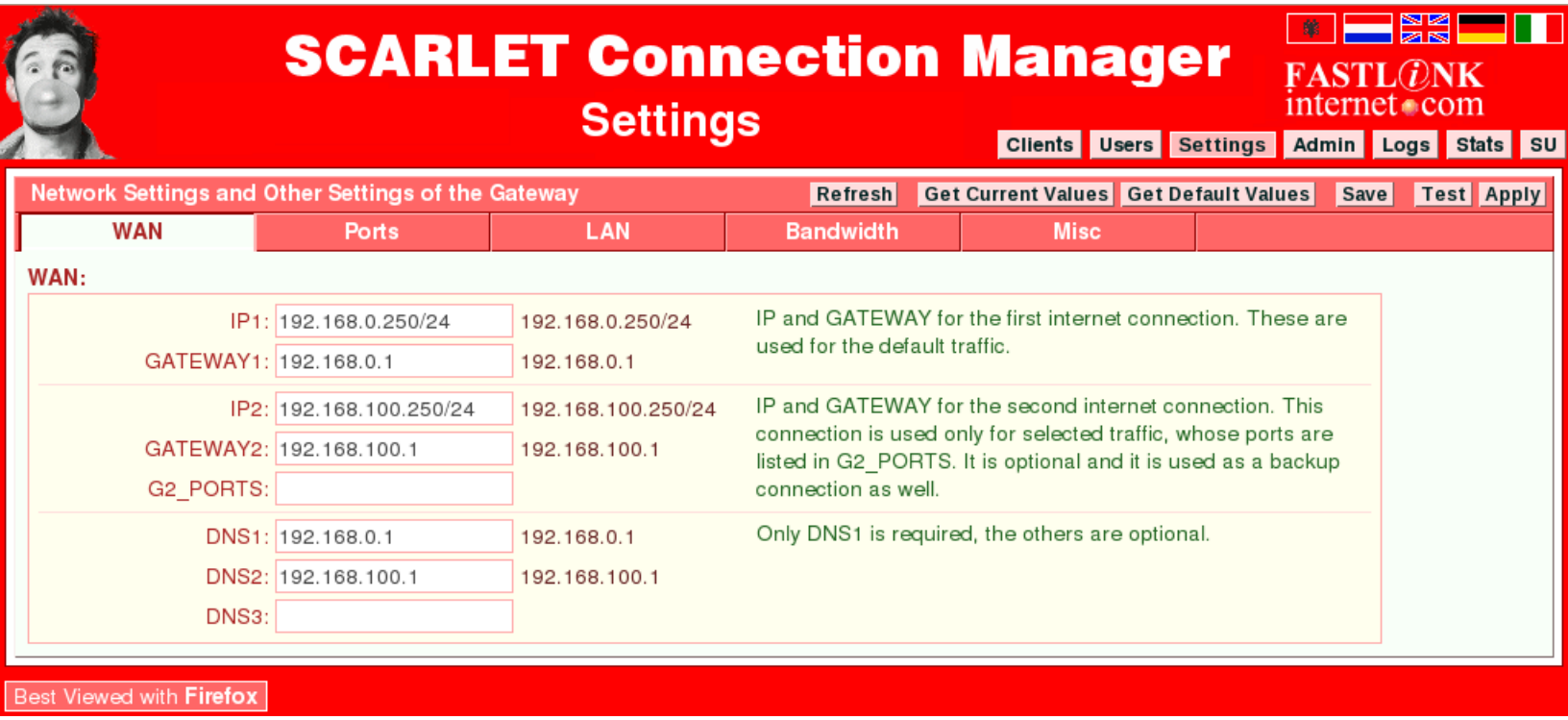
This is the screen that gets an internet client when he tries to access a web page but he is not registered yet (or his registration is expired).



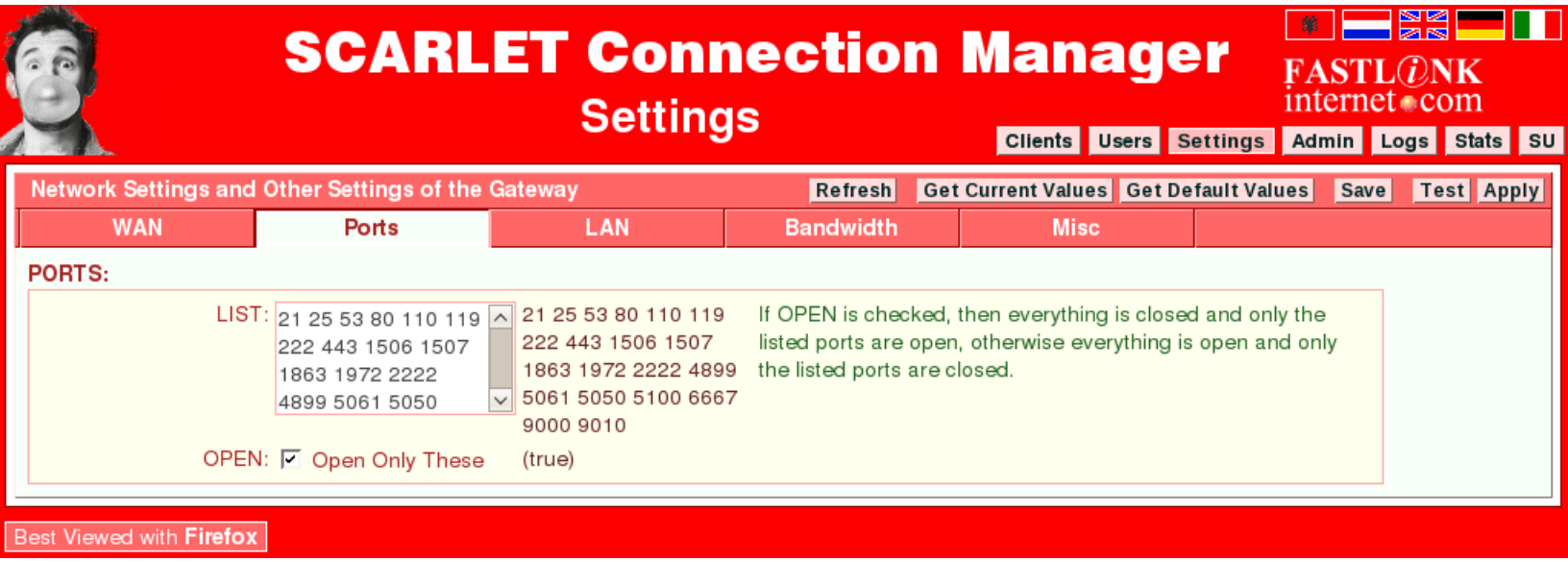
A client can have many connections. He registers them using his username and access code. His connections have a time expiration and an upload/download limit.



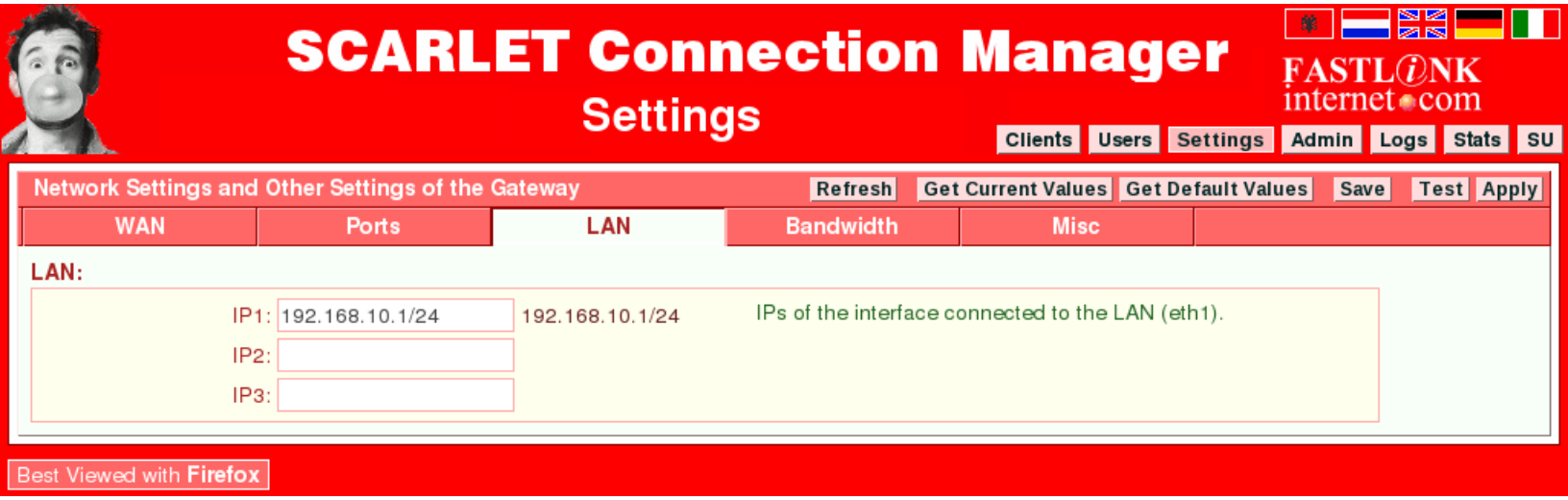
Scarlet generates upload/download traffic statistics for each connection of the client and for all of them in total. It also generates usage graphs for different periods of time (hourly, daily, monthly, etc).



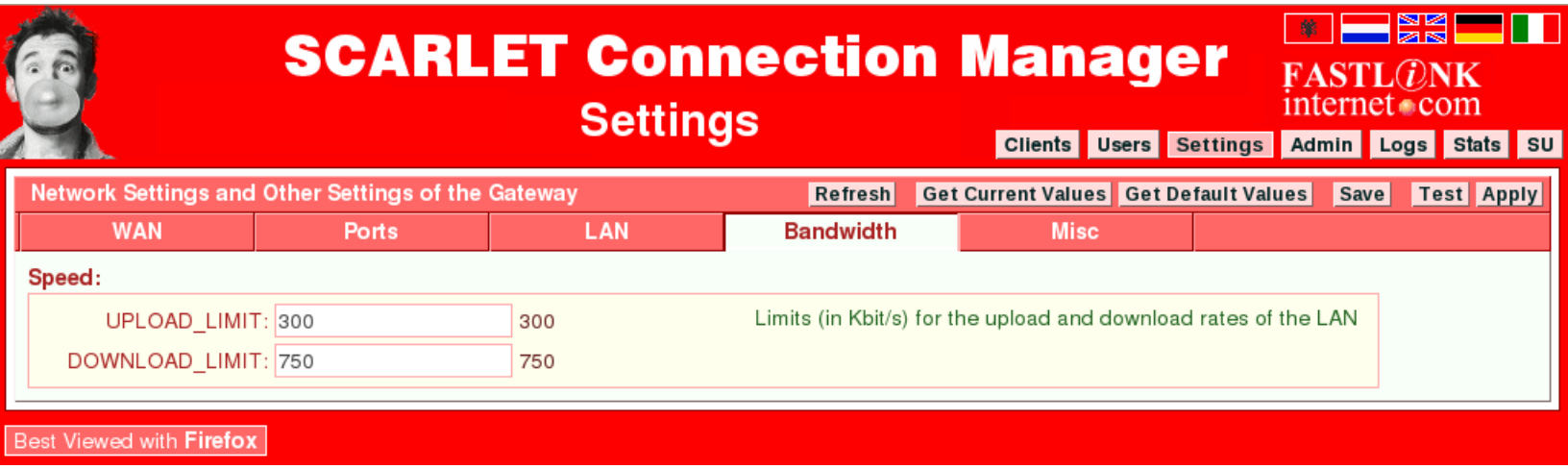
The configuration of the WAN side of Scarlet. It can use two different gateways to the Internet.



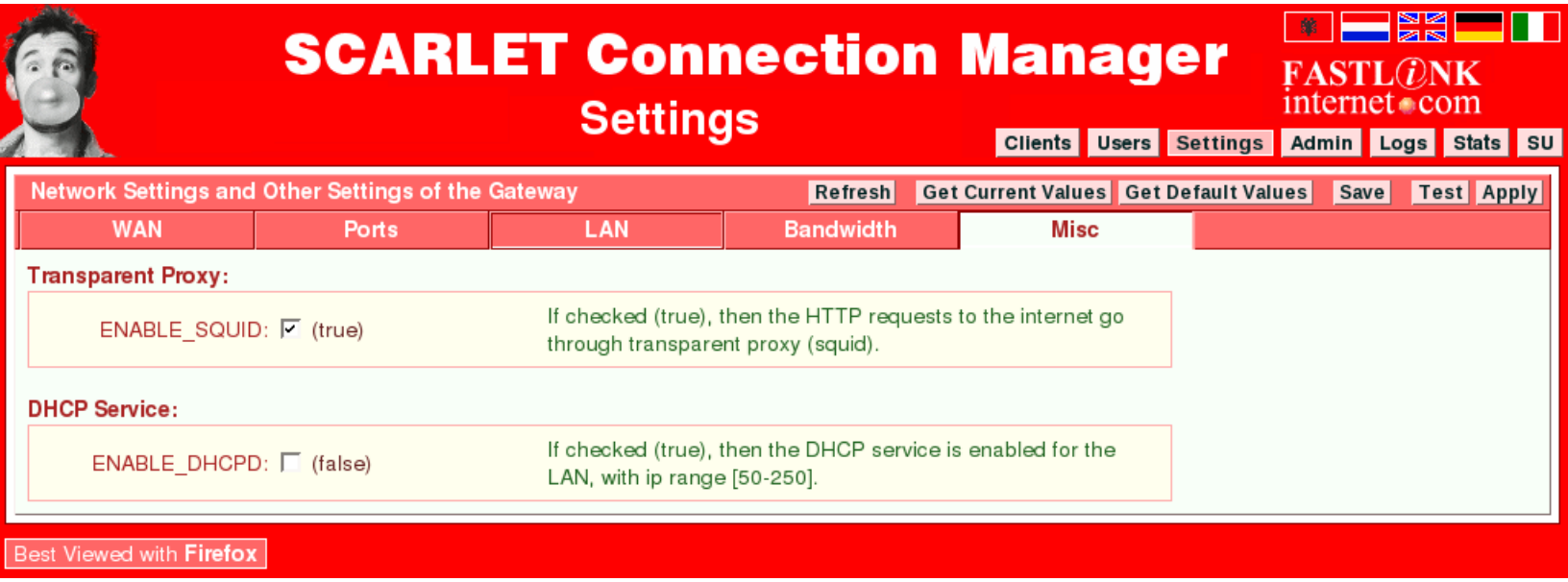
Scarlet can block certain network ports.



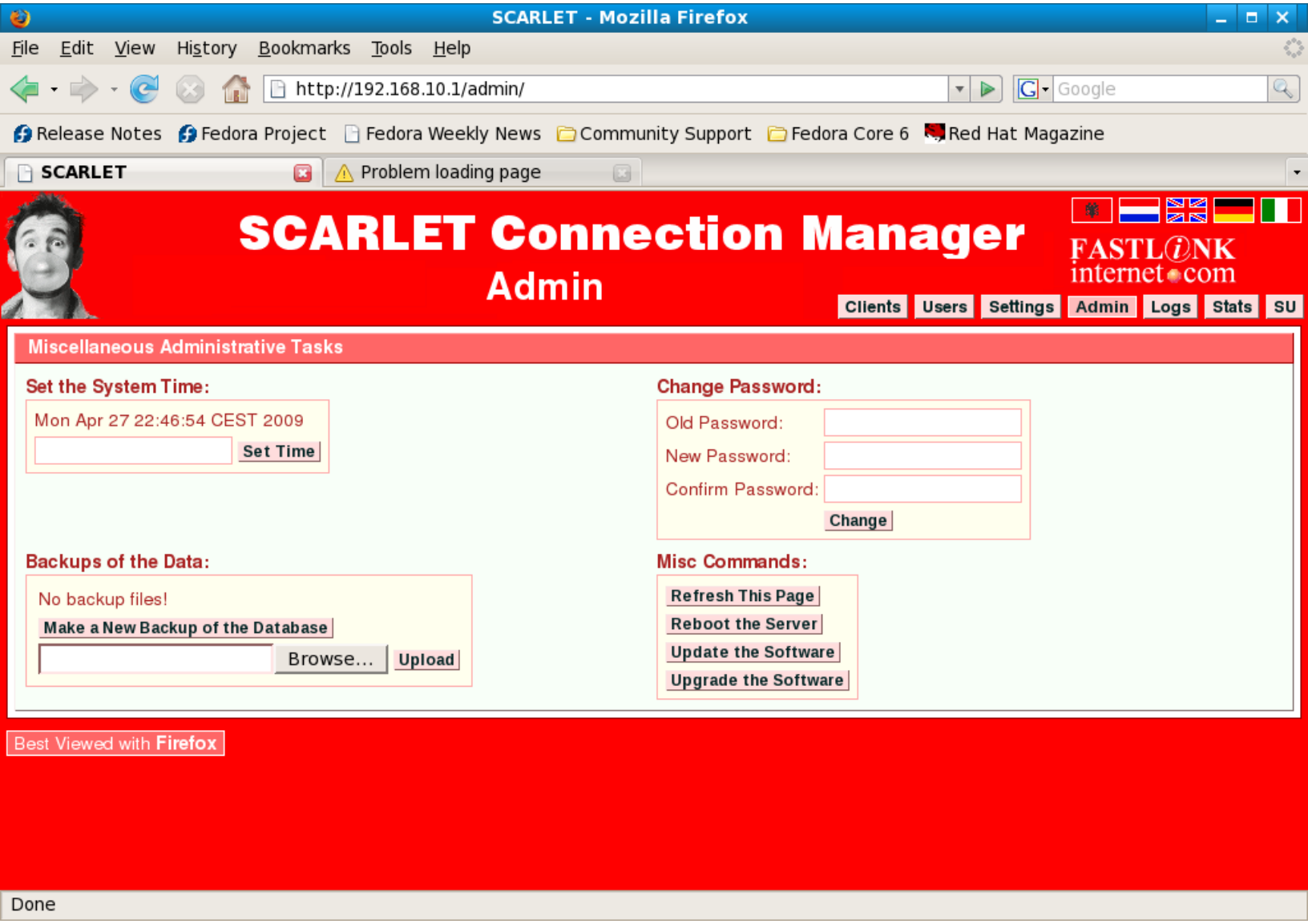
Scarlet can have more than one LAN interface.



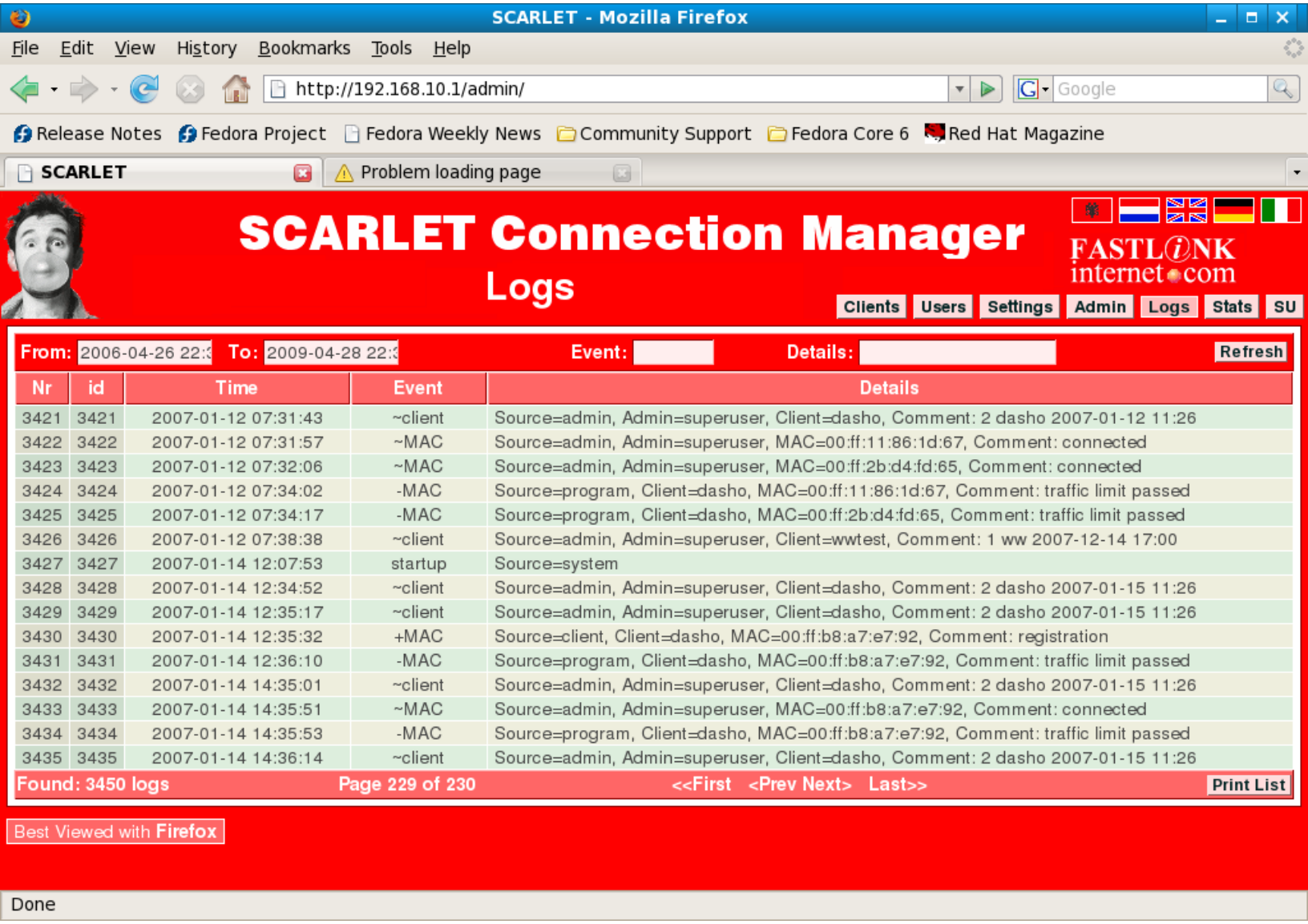
Scarlet can limit the bandwidth of the network traffic.



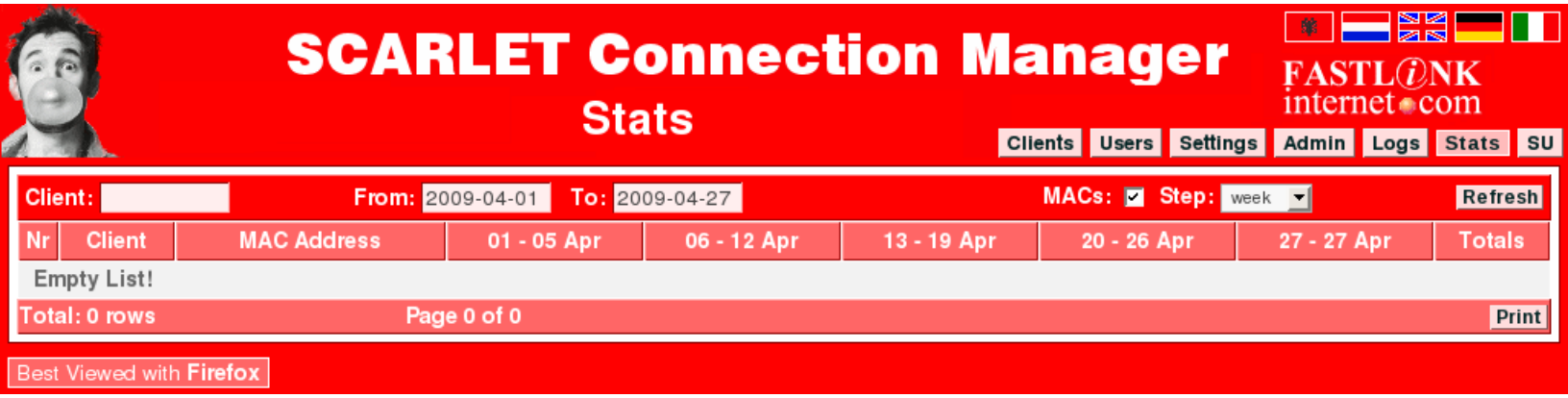
Scarlet can also use transparent proxy, can give DHCP service, etc.



Scarlet has some interesting administrative features, like backup, update, upgrade, etc.



Every important action is logged and the logs can be filtered (searched) and printed.



Scarlet also has a flexible statistics module.

15.2.Slackware Packages Installed in Scarlet

dependencies.txt :

```
apache
cyrus-sasl
diffutils
dnsmasq
gawk
gettext-tools
glibc-i18n
iproute2
iptables
isapnptools
lftp
libart_lgpl
libidn
libxml2
mhash
mysql
openssh
openssl
php
procmail
sendmail
stunnel
subversion
wget
wireless-tools
```

packages.txt :

```
aaa_base-11.0.0-noarch-2
aaa_elflibs-11.0.0-i486-9
acl-2.2.39_1-i486-1
acpid-1.0.4-i486-2
apache-1.3.37-i486-2
apmd-3.0.2-i386-1
apr-1.2.7-i486-1
apr-util-1.2.7-i486-1
at-3.1.10-i486-1
attr-2.4.32_1-i486-1
bash-3.1.017-i486-1
```

```
bin-11.0-i486-3
bind-9.3.2_P1-i486-1
bzip2-1.0.3-i486-3
coreutils-5.97-i486-1
curl-7.15.5-i486-1
cxxlibs-6.0.3-i486-1
cyrus-sasl-2.1.22-i486-1
dcron-2.3.3-i486-5
devs-2.3.1-noarch-25
dhcp-3.0.4-i486-2
dhcpcd-2.0.4-i486-2
diffutils-2.8.1-i486-3
dnsmasq-2.33-i486-1
e2fsprogs-1.38-i486-2
elvis-2.2_0-i486-2
etc-11.0-noarch-2
findutils-4.2.28-i486-1
gawk-3.1.5-i486-3
genpower-1.0.5-i486-1
gettext-0.15-i486-1
gettext-tools-0.15-i486-1
glibc-i18n-2.3.6-noarch-6
glibc-solibs-2.3.6-i486-6
glibc-zoneinfo-2.3.6-noarch-6
gnupg-1.4.5-i486-1
grep-2.5-i486-3
groff-1.19.2-i486-1
gzip-1.3.5-i486-1
hdparm-6.6-i486-1
hotplug-2004_09_23-noarch-11
inetd-1.79s-i486-7
iproute2-2.6.16_060323-i486-1
iptables-1.3.5-i486-2
iptraf-2.7.0-i386-1
isapnptools-1.26-i386-1
joe-3.5-i486-1
jove-4.16.0.61-i386-1
kernel-ide-2.4.33.3-i486-1
kernel-modules-2.4.33.3-i486-1
less-394-i486-1
lftp-3.5.4-i486-1
libart_lgpl-2.3.17-i486-1
libidn-0.6.5-i486-1
libxml2-2.6.26-i486-1
lilo-22.7.1-i486-2
logrotate-3.7.4-i486-1
mailx-12.1-i486-1
man-1.6c-i486-2
```

```
man-pages-2.39-noarch-1
mc-4.6.1-i486-2
mhash-0.9.7-i486-1
minicom-2.1-i486-2
mkinitrd-1.0.1-i486-3
module-init-tools-3.2.2-i486-2
mutt-1.4.2.2i-i486-1
mysql-5.0.24a-i486-1
nc-1.10-i386-1
neon-0.25.5-i486-2
netwatch-1.0a-i386-1
nmap-4.11-i486-1
openldap-client-2.3.27-i486-1
openssh-4.4p1-i486-1
openssl-0.9.8d-i486-1
openssl-solibs-0.9.8d-i486-1
pciutils-2.2.3-i486-2
pcmcia-cs-3.2.8-i486-3
php-4.4.4-i486-3
pidentd-3.0.19-i486-1
pkgtools-11.0.0-i486-4
portmap-5.0-i486-3
procmail-3.22-i486-2
procps-3.2.7-i486-1
reiserfsprogs-3.6.19-i486-1
rrdtool-1.2.8-i486-1jm
scarlet-2.0
screen-4.0.2-i486-1
sed-4.1.5-i486-1
sendmail-8.13.8-i486-4
shadow-4.0.3-i486-13
sharutils-4.6.3-i486-1
slocate-3.1-i486-1
smartmontools-5.36-i486-1
squid-2.6.STABLE1-i386-1
stunnel-4.17-i486-1
subversion-1.4.0-i486-1
sudo-1.6.8p12-i486-1
sysfsutils-2.0.0-i486-2
sysklogd-1.4.1-i486-9
syslinux-2.13-i486-1
sysvinit-2.84-i486-69
tar-1.15.1-i486-2
tcpdump-3.9.4-i486-2
tcpip-0.17-i486-39
tcsh-6.14.00-i486-2
texinfo-4.8-i486-1
traceroute-1.4a12-i386-2
```

```
udev-097-i486-10
umsdos-progs-1.13-i386-1
usbutils-0.72-i486-1
utempter-1.1.3-i486-1
util-linux-2.12r-i486-5
vim-7.0.109-i486-1
wget-1.10.2-i486-2
whois-4.7.15-i486-1
wireless-tools-28-i486-3
```

15.3.Modifications of the Configuration Files

sshd_config.diff :

```
--- /etc/ssh/sshd_config.bak    2006-04-25 16:20:16.000000000 +0200
+++ templates/sshd_config       2006-04-25 16:17:32.000000000 +0200
@@ -11,6 +11,8 @@
 # default value.

 #Port 22
+### dasho
+Port 2222
 #Protocol 2,1
 #AddressFamily any
 #ListenAddress 0.0.0.0
```

rc.local.diff :

```
--- rc.local.1  2007-01-11 01:39:51.000000000 +0100
+++ rc.local    2007-01-04 15:06:21.000000000 +0100
@@ -4,3 +4,20 @@
 #
 # Put any local setup commands in here:

+### application path
+app_path=|APP_PATH|
+
+### record the startup event
+$app_path/scripts/record_startup.sh
+
+### configure the server
+$app_path/server-config/reconfig.sh
+
+### check periodicly the status of the gateways
+$app_path/server-config/watch.sh &
+
```

```
+### start the traffic counter
+$app_path/server-config/traffic-counter/start.sh
+
+### start 'cron/check-traffic-limits.php'
+$app_path/cron/check-traffic-limits.php &
```

hosts.diff :

```
--- /etc/hosts.bak      2006-04-25 15:18:00.000000000 +0200
+++ templates/hosts     2006-04-25 14:33:39.000000000 +0200
@@ -13,7 +13,7 @@

 # For loopbacking.
 127.0.0.1       localhost
+|LOCAL_IP|      scarlet.|DOMAIN| scarlet router.|DOMAIN| router

 # End of hosts.
```

dnsmasq.diff :

```
--- dnsmasq.conf.1      2007-01-11 01:35:46.000000000 +0100
+++ dnsmasq.conf        2007-01-11 01:32:05.000000000 +0100
@@ -30,6 +30,7 @@
 # Change this line if you want dns to get its upstream servers
from
 # somewhere other that /etc/resolv.conf
 #resolv-file=
+resolv-file=/etc/resolv.dnsmasq.conf

 # By  default,  dnsmasq  will  send queries to any of the upstream
 # servers it knows about and tries to favour servers to are  known
@@ -54,6 +55,7 @@
 # Add local-only domains here, queries in these domains are
answered
 # from /etc/hosts or DHCP only.
 #local=/localnet/
+local=/|DOMAIN|/

 # Add domains which you want to force to an IP address here.
 # The example below send any host in doubleclick.net to a local
@@ -72,6 +74,7 @@
 #interface=
 # Or you can specify which interface _not_ to listen on
 #except-interface=
+except-interface=eth0
 # Or which to listen on by address (remember to include 127.0.0.1
```

```
if
 # you use this.)
 #listen-address=
@@ -97,6 +100,7 @@
 # Set this (and domain: see below) if you want to have a domain
 # automatically added to simple names in a hosts-file.
 #expand-hosts
+expand-hosts

 # Set the domain for dnsmasq. this is optional, but if it is set,
it
 # does the following things.
@@ -106,6 +110,7 @@
 #    domain of all systems configured by DHCP
 # 3) Provides the domain part for "expand-hosts"
 #domain=thekelleys.org.uk
+domain=|DOMAIN|

 # Uncomment this to enable the integrated DHCP server, you need
 # to supply the range of addresses available for lease and
optionally
@@ -113,6 +118,7 @@
 # repeat this for each network on which you want to supply DHCP
 # service.
 #dhcp-range=192.168.0.50,192.168.0.150,12h
+dhcp-range=|LAN_NETWORK|.50,|LAN_NETWORK|.250,12h

 # This is an example of a DHCP range where the netmask is given.
This
 # is needed for networks we reach the dnsmasq DHCP server via a
relay
```

httpd.conf.diff :

```
--- /etc/apache/httpd.conf.default      2006-08-10
02:50:08.000000000 +0200
+++ httpd.conf  2007-01-11 01:29:28.000000000 +0100
@@ -136,14 +136,17 @@
 # a new spare.  If there are more than MaxSpareServers, some of
the
 # spares die off.  The default values are probably OK for most
sites.
 #
-MinSpareServers 5
-MaxSpareServers 10
+#MinSpareServers 5
+#MaxSpareServers 10
```

```
+MinSpareServers 2
+MaxSpareServers 5

 #
 # Number of servers to start initially --- should be a reasonable
ballpark
 # figure.
 #
-StartServers 5
+#StartServers 5
+StartServers 2
#
 # Limit on total number of servers running, i.e., limit on the
number
@@ -152,7 +155,8 @@
 # It is intended mainly as a brake to keep a runaway server from
taking
 # the system with it as it spirals down...
 #
-MaxClients 150
+#MaxClients 150
+MaxClients 50

 #
 # MaxRequestsPerChild: the number of requests each child process
is
@@ -168,7 +172,8 @@
 #       an initial request and 10 subsequent "keptalive" requests,
it
 #       would only count as 1 request towards this limit.
 #
-MaxRequestsPerChild 0
+#MaxRequestsPerChild 0
+MaxRequestsPerChild 1000

 #
 # Listen: Allows you to bind Apache to specific IP addresses
and/or
@@ -327,7 +332,7 @@
 # e-mailed.  This address appears on some server-generated pages,
such
 # as error documents.
 #
-ServerAdmin root@tree.slackware.lan
+ServerAdmin root@scarlet.localnet.net

 #
 # ServerName allows you to set a host name which is sent back to
```

```
clients for
@@ -346,13 +351,15 @@
 # local testing and development, you may use 127.0.0.1 as the
server name.
 #
 #ServerName www.example.com
+ServerName scarlet.localnet.net

 #
 # DocumentRoot: The directory out of which you will serve your
 # documents. By default, all requests are taken from this
directory, but
 # symbolic links and aliases may be used to point to other
locations.
 #
-DocumentRoot "/var/www/htdocs"
+#DocumentRoot "/var/www/htdocs"
+DocumentRoot "|APP_PATH|"

 #
 # Each directory to which Apache has access, can be configured
with respect
@@ -377,7 +384,8 @@
 #
 # This should be changed to whatever you set DocumentRoot to.
 #
-<Directory "/var/www/htdocs">
+#<Directory "/var/www/htdocs">
+<Directory "|APP_PATH|">

 #
 # This may also be "None", "All", or any combination of "Indexes",
@@ -386,7 +394,8 @@
 # Note that "MultiViews" must be named *explicitly* --- "Options
All"
 # doesn't give it to you.
 #
-    Options Indexes FollowSymLinks MultiViews
+    #Options Indexes FollowSymLinks MultiViews
+    Options FollowSymLinks MultiViews

 #
 # This controls which options the .htaccess files in directories
can
@@ -911,6 +920,8 @@
 #
 #    2) local redirects
 #ErrorDocument 404 /missing.html
```

```
+ErrorDocument 404 /connect.php
+
 #  to redirect to local URL /missing.html
 #ErrorDocument 404 /cgi-bin/missing_handler.pl
 #  N.B.: You can redirect to a script or a document using server-
side-includes.
@@ -1030,7 +1041,7 @@
 #                     mysql (AP series), gmp (L series), mhash (L
series),
 #                     and apache (N series)
 #
-#Include /etc/apache/mod_php.conf
+Include /etc/apache/mod_php.conf

 # ==> mod_ssl configuration settings <==
 #
```

mod_php.conf.diff :

```
--- /etc/apache/mod_php.conf.bak        2006-04-25
15:16:47.000000000 +0200
+++ templates/mod_php.conf      2006-04-25 14:33:39.000000000 +0200
@@ -10,3 +10,10 @@

 # This will display PHP files in colored syntax form.  Use with
caution.
 #AddType application/x-httpd-php-source .phps
+
+#
+# Add index.php to the list of files that will be served as
directory
+# indexes.
+#
+DirectoryIndex index.php
+
```

php.ini.diff :

```
--- /etc/apache/php.ini.bak     2006-04-26 14:33:06.000000000 +0200
+++ /etc/apache/php.ini 2006-04-26 14:27:13.000000000 +0200
@@ -282,14 +282,17 @@
 ;
 ;   - Show all errors
 ;
-error_reporting  =  E_ALL
+;error_reporting  =  E_ALL
```

```
+error_reporting = E_ALL & ~E_NOTICE

 ; Print out errors (as a part of the output).  For production web
sites,
 ; you're strongly encouraged to turn this feature off, and use
error logging
 ; instead (see below).  Keeping display_errors enabled on a
production web site
 ; may reveal security information to end users, such as file paths
on your Web
 ; server, your database schema or other information.
-display_errors = Off
+display_errors = On

 ; Even when display_errors is on, errors that occur during PHP's
startup
 ; sequence are not displayed.  It's strongly recommended to keep
```

squid.conf :

```
```

squid.logrotate :

```
```