Chapter

10 Network Infrastructure

Contents

Linux sports the whole Unix heritage for networking, and Debian provides a full set of tools to create and manage them. This chapter reviews these tools.

10.1. Gateway

A gateway is a system linking several networks. This term often refers to a local network's "exit point" on the mandatory path to all external IP addresses. The gateway is connected to each of the networks it links together, and acts as a router to convey IP packets between its various interfaces.

BACK TO BASICS

IP packet

Most networks nowadays use the IP protocol (*Internet Protocol*). This protocol segments the transmitted data into limited-size packets. Each packet contains, in addition to its payload data, a number of details required for its proper routing.

BACK TO BASICS

TCP/UDP

Many programs do not handle the individual packets themselves, even though the data they transmit does travel over IP; they often use TCP (*Transmission Control Protocol*). TCP is a layer over IP allowing the establishment of connections dedicated to data streams between two points. The programs then only see an entry point into which data can be fed with the guarantee that the same data exits without loss (and in the same sequence) at the exit point at the other end of the connection. Although many kinds of errors can happen in the lower layers, they are compensated by TCP: lost packets are retransmitted, and packets arriving out of order (for example, if they used different paths) are re-ordered appropriately.

Another protocol relying on IP is UDP (*User Datagram Protocol*). In contrast to TCP, it is packet-oriented. Its goals are different: the purpose of UDP is only to transmit one packet from an application to another. The protocol does not try to compensate for possible packet loss on the way, nor does it ensure that packets are received in the same order as were sent. The main advantage to this protocol is that the latency is greatly improved, since the loss of a single packet does not delay the receiving of all following packets until the lost one is retransmitted.

TCP and UDP both involve ports, which are "extension numbers" for establishing communication with a given application on a machine. This concept allows keeping several different communications in parallel with the same correspondent, since these communications can be distinguished by the port number.

Some of these port numbers — standardized by the IANA (*Internet Assigned Numbers Authority*) — are "well-known" for being associated with network services. For instance, TCP port 25 is generally used by the email server.

➨ `http://www.iana.org/assignments/port-numbers`

When a local network uses a private address range (not routable on the Internet), the gateway needs to implement *address masquerading* so that the machines on the network can communicate with the outside world. The masquerading operation is a kind of proxy operating on the network level: each outgoing connection from an internal machine is replaced with a connection from the gateway itself (since the gateway does have an external, routable address), the data going through the masqueraded connection is sent to the new one, and the data coming back in reply is sent through to the masqueraded connection to the internal machine. The gateway uses a range of dedicated TCP ports for this purpose, usually with very high numbers (over 60000). Each connection coming from an internal machine then appears to the outside world as a connection coming from one of these reserved ports.

CULTURE

Private address range

RFC 1918 defines three ranges of IPv4 addresses not meant to be routed on the Internet but only used in local networks. The first one, `10.0.0.0/8` (see sidebar "Essential network concepts (Ethernet, IP address, subnet, broadcast)" page 149), is a class-A range (with 2^{24} IP addresses). The second one, `172.16.0.0/12`, gathers 16 class-B ranges (`172.16.0.0/16` to `172.31.0.0/16`), each containing 2^{16} IP addresses. Finally, `192.168.0.0/16` is a class-B range (grouping 256 class-C ranges, `192.168.0.0/24` to `192.168.255.0/24`, with 256 IP addresses each).

➨ `http://www.faqs.org/rfcs/rfc1918.html`

The gateway can also perform two kinds of *network address translation* (or NAT for short). The first kind, *Destination NAT* (DNAT) is a technique to alter the destination IP address (and/or the TCP or UDP port) for a (generally) incoming connection. The connection tracking mechanism also alters the following packets in the same connection to ensure continuity in the communication. The second kind of NAT is *Source NAT* (SNAT), of which *masquerading* is a particular case; SNAT alters the source IP address (and/or the TCP or UDP port) of a (generally) outgoing connection. As for DNAT, all the packets in the connection are appropriately handled by the connection tracking mechanism. Note that NAT is only relevant for IPv4 and its limited address space; in IPv6, the wide availability of addresses greatly reduces the usefulness of NAT by allowing all "internal" addresses to be directly routable on the Internet (this does not imply that internal machines are accessible, since intermediary firewalls can filter traffic).

BACK TO BASICS

Port forwarding

A concrete application of DNAT is *port forwarding*. Incoming connections to a given port of a machine are forwarded to a port on another machine. Other solutions may exist for achieving a similar effect, though, especially at the application level with `ssh` (see section 9.2.1.3, "Creating Encrypted Tunnels with Port Forwarding" page 194) or `redir`.

Enough theory, let's get practical. Turning a Debian system into a gateway is a simple matter of enabling the appropriate option in the Linux kernel, by way of the `/proc/` virtual filesystem:

```
# echo 1 > /proc/sys/net/ipv4/conf/default/forwarding
```

This option can also be automatically enabled on boot if `/etc/sysctl.conf` sets the net.ipv4.conf.default.forwarding option to 1.

Example 10.1 *The `/etc/sysctl.conf` file*

```
net.ipv4.conf.default.forwarding = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.tcp_syncookies = 1
```

The same effect can be obtained for IPv6 by simply replacing ipv4 with ipv6 in the manual command and using the net.ipv6.conf.all.forwarding line in `/etc/sysctl.conf`.

Enabling IPv4 masquerading is a slightly more complex operation that involves configuring the *netfilter* firewall.

Similarly, using NAT (for IPv4) requires configuring *netfilter*. Since the primary purpose of this component is packet filtering, the details are listed in Chapter 14: "Security" (see section 14.2, "Firewall or Packet Filtering" page 377).

10.2. Virtual Private Network

A *Virtual Private Network* (VPN for short) is a way to link two different local networks through the Internet by way of a tunnel; the tunnel is usually encrypted for confidentiality. VPNs are often used to integrate a remote machine within a company's local network.

Several tools provide this. OpenVPN is an efficient solution, easy to deploy and maintain, based on SSL/TLS. Another possibility is using IPsec to encrypt IP traffic between two machines; this encryption is transparent, which means that applications running on these hosts need not be modified to take the VPN into account. SSH can also be used to provide a VPN, in addition to its more conventional features. Finally, a VPN can be established using Microsoft's PPTP protocol. Other solutions exist, but are beyond the focus of this book.

10.2.1. OpenVPN

OpenVPN is a piece of software dedicated to creating virtual private networks. Its setup involves creating virtual network interfaces on the VPN server and on the client(s); both `tun` (for IP-level tunnels) and `tap` (for Ethernet-level tunnels) interfaces are supported. In practice, `tun` interfaces will most often be used except when the VPN clients are meant to be integrated into the server's local network by way of an Ethernet bridge.

OpenVPN relies on OpenSSL for all the SSL/TLS cryptography and associated features (confidentiality, authentication, integrity, non-repudiation). It can be configured either with a shared private key or using X.509 certificates based on a public key infrastructure. The latter configuration is strongly preferred since it allows greater flexibility when faced with a growing number of roaming users accessing the VPN.

CULTURE

SSL and TLS

The SSL protocol (*Secure Socket Layer*) was invented by Netscape to secure connections to web servers. It was later standardized by IETF under the acronym TLS (*Transport Layer Security*). Since then TLS continued to evolve and nowadays SSL is deprecated due to multiple design flaws that have been discovered.

Public Key Infrastructure: easy-rsa

The RSA algorithm is widely used in public-key cryptography. It involves a "key pair", comprised of a private and a public key. The two keys are closely linked to each other, and their mathematical properties are such that a message encrypted with the public key can only be

decrypted by someone knowing the private key, which ensures confidentiality. In the opposite direction, a message encrypted with the private key can be decrypted by anyone knowing the public key, which allows authenticating the origin of a message since only someone with access to the private key could generate it. When associated with a digital hash function (MD5, SHA1, or a more recent variant), this leads to a signature mechanism that can be applied to any message.

However, anyone can create a key pair, store any identity on it, and pretend to be the identity of their choice. One solution involves the concept of a *Certification Authority* (CA), formalized by the X.509 standard. This term covers an entity that holds a trusted key pair known as a *root certificate*. This certificate is only used to sign other certificates (key pairs), after proper steps have been undertaken to check the identity stored on the key pair. Applications using X.509 can then check the certificates presented to them, if they know about the trusted root certificates.

OpenVPN follows this rule. Since public CAs only emit certificates in exchange for a (hefty) fee, it is also possible to create a private certification authority within the company. The *easy-rsa* package provides tools to serve as an X.509 certification infrastructure, implemented as a set of scripts using the `openssl` command.

NOTE

easy-rsa **before** *Jessie*

In versions of Debian up to *Wheezy*, *easy-rsa* was distributed as part of the *openvpn* package, and its scripts were to be found under `/usr/share/doc/openvpn/examples/easy-rsa/2.0/`. Setting up a CA involved copying that directory, instead of using the `make-cadir` command as documented here.

The Falcot Corp administrators use this tool to create the required certificates, both for the server and the clients. This allows the configuration of all clients to be similar since they will only have to be set up so as to trust certificates coming from Falcot's local CA. This CA is the first certificate to create; to this end, the administrators set up a directory with the files required for the CA in an appropriate location, preferably on a machine not connected to the network in order to mitigate the risk of the CA's private key being stolen.

```
$ make-cadir pki-falcot
$ cd pki-falcot
```

They then store the required parameters into the `vars` file, especially those named with a KEY_ prefix; these variables are then integrated into the environment:

```
$ vim vars
$ grep KEY_ vars
export KEY_CONFIG=‘$EASY_RSA/whichopensslcnf $EASY_RSA‘
export KEY_DIR=”$EASY_RSA/keys”
echo NOTE: If you run ./clean-all, I will be doing a rm -rf on $KEY_DIR
export KEY_SIZE=2048
export KEY_EXPIRE=3650
export KEY_COUNTRY=”FR”
export KEY_PROVINCE=”Loire”
export KEY_CITY=”Saint-Étienne”
export KEY_ORG=”Falcot Corp”
```

```
export KEY_EMAIL="admin@falcot.com"
export KEY_OU="Certificate authority"
export KEY_NAME="Certificate authority for Falcot Corp"
# If you'd like to sign all keys with the same Common Name, uncomment the KEY_CN
       export below
# export KEY_CN="CommonName"
$ . ./vars
NOTE: If you run ./clean-all, I will be doing a rm -rf on /home/roland/pki-falcot/
       keys
$ ./clean-all
```

The next step is the creation of the CA's key pair itself (the two parts of the key pair will be stored under `keys/ca.crt` and `keys/ca.key` during this step):

```
$ ./build-ca
Generating a 2048 bit RSA private key
.....................................................................+++
...+++
writing new private key to 'ca.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:
State or Province Name (full name) [Loire]:
Locality Name (eg, city) [Saint-Étienne]:
Organization Name (eg, company) [Falcot Corp]:
Organizational Unit Name (eg, section) [Certificate authority]:
Common Name (eg, your name or your server's hostname) [Falcot Corp CA]:
Name [Certificate authority for Falcot Corp]:
Email Address [admin@falcot.com]:
```

The certificate for the VPN server can now be created, as well as the Diffie-Hellman parameters required for the server side of an SSL/TLS connection. The VPN server is identified by its DNS name vpn.falcot.com; this name is re-used for the generated key files (`keys/vpn.falcot.com.crt` for the public certificate, `keys/vpn.falcot.com.key` for the private key):

```
$ ./build-key-server vpn.falcot.com
Generating a 2048 bit RSA private key
.....................................................................................
.....................................................................................

...........+++
writing new private key to 'vpn.falcot.com.key'
-----
You are about to be asked to enter information that will be incorporated
```

```
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:
State or Province Name (full name) [Loire]:
Locality Name (eg, city) [Saint-Étienne]:
Organization Name (eg, company) [Falcot Corp]:
Organizational Unit Name (eg, section) [Certificate authority]:
Common Name (eg, your name or your server's hostname) [vpn.falcot.com]:
Name [Certificate authority for Falcot Corp]:
Email Address [admin@falcot.com]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from /home/roland/pki-falcot/openssl-1.0.0.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :PRINTABLE:'Loire'
localityName          :T61STRING:'Saint-\0xFFFFFFC3\0xFFFFFF89tienne'
organizationName      :PRINTABLE:'Falcot Corp'
organizationalUnitName:PRINTABLE:'Certificate authority'
commonName            :PRINTABLE:'vpn.falcot.com'
name                  :PRINTABLE:'Certificate authority for Falcot Corp'
emailAddress          :IA5STRING:'admin@falcot.com'
Certificate is to be certified until Mar  6 14:54:56 2025 GMT (3650 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
$ ./build-dh
Generating DH parameters, 2048 bit long safe prime, generator 2
This is going to take a long time
[…]
```

The following step creates certificates for the VPN clients; one certificate is required for each computer or person allowed to use the VPN:

```
$ ./build-key JoeSmith
Generating a 2048 bit RSA private key
..............................+++
..........................................+++
```

```
writing new private key to 'JoeSmith.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:
State or Province Name (full name) [Loire]:
Locality Name (eg, city) [Saint-Étienne]:
Organization Name (eg, company) [Falcot Corp]:
Organizational Unit Name (eg, section) [Certificate authority]:Development unit
Common Name (eg, your name or your server's hostname) [JoeSmith]:Joe Smith
[…]
```

Now all certificates have been created, they need to be copied where appropriate: the root certificate's public key (`keys/ca.crt`) will be stored on all machines (both server and clients) as `/etc/ssl/certs/Falcot_CA.crt`. The server's certificate is installed only on the server (`keys/vpn.falcot.com.crt` goes to `/etc/ssl/vpn.falcot.com.crt`, and `keys/vpn.falcot.com.key` goes to `/etc/ssl/private/vpn.falcot.com.key` with restricted permissions so that only the administrator can read it), with the corresponding Diffie-Hellman parameters (`keys/dh2048.pem`) installed to `/etc/openvpn/dh2048.pem`. Client certificates are installed on the corresponding VPN client in a similar fashion.

Configuring the OpenVPN Server

By default, the OpenVPN initialization script tries starting all virtual private networks defined in `/etc/openvpn/*.conf`. Setting up a VPN server is therefore a matter of storing a corresponding configuration file in this directory. A good starting point is `/usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz`, which leads to a rather standard server. Of course, some parameters need to be adapted: ca, cert, key and dh need to describe the selected locations (respectively, /etc/ssl/certs/Falcot_CA.crt, /etc/ssl/vpn.falcot.com.crt, /etc/ssl/private/vpn.falcot.com.key and /etc/openvpn/dh2048.pem). The server 10.8.0.0 255.255.255.0 directive defines the subnet to be used by the VPN; the server uses the first IP address in that range (10.8.0.1) and the rest of the addresses are allocated to clients.

With this configuration, starting OpenVPN creates the virtual network interface, usually under the tun0 name. However, firewalls are often configured at the same time as the real network interfaces, which happens before OpenVPN starts. Good practice therefore recommends creating a persistent virtual network interface, and configuring OpenVPN to use this pre-existing interface. This further allows choosing the name for this interface. To this end, `openvpn --mktun --dev vpn --dev-type tun` creates a virtual network interface named vpn with type tun; this command can easily be integrated in the firewall configuration script, or in an up di-

rective of the `/etc/network/interfaces` file. The OpenVPN configuration file must also be updated accordingly, with the dev vpn and dev-type tun directives.

Barring further action, VPN clients can only access the VPN server itself by way of the 10.8.0.1 address. Granting the clients access to the local network (192.168.0.0/24), requires adding a push route 192.168.0.0 255.255.255.0 directive to the OpenVPN configuration so that VPN clients automatically get a network route telling them that this network is reachable by way of the VPN. Furthermore, machines on the local network also need to be informed that the route to the VPN goes through the VPN server (this automatically works when the VPN server is installed on the gateway). Alternatively, the VPN server can be configured to perform IP masquerading so that connections coming from VPN clients appear as if they are coming from the VPN server instead (see section 10.1, "Gateway" page 222).

Configuring the OpenVPN Client

Setting up an OpenVPN client also requires creating a configuration file in `/etc/openvpn/`. A standard configuration can be obtained by using `/usr/share/doc/openvpn/examples/sample-config-files/client.conf` as a starting point. The remote vpn.falcot.com 1194 directive describes the address and port of the OpenVPN server; the ca, cert and key also need to be adapted to describe the locations of the key files.

If the VPN should not be started automatically on boot, set the AUTOSTART directive to none in the `/etc/default/openvpn` file. Starting or stopping a given VPN connection is always possible with the commands `service openvpn@`*`name`* `start` and `service openvpn@`*`name`* `stop` (where the connection *name* matches the one defined in `/etc/openvpn/`*`name`*`.conf`).

The *network-manager-openvpn-gnome* package contains an extension to Network Manager (see section 8.2.4, "Automatic Network Configuration for Roaming Users" page 153) that allows managing OpenVPN virtual private networks. This allows every user to configure OpenVPN connections graphically and to control them from the network management icon.

10.2.2. Virtual Private Network with SSH

There are actually two ways of creating a virtual private network with SSH. The historic one involves establishing a PPP layer over the SSH link. This method is described in a HOWTO document:

➨ `http://www.tldp.org/HOWTO/ppp-ssh/`

The second method is more recent, and was introduced with OpenSSH 4.3; it is now possible for OpenSSH to create virtual network interfaces (tun*) on both sides of an SSH connection, and these virtual interfaces can be configured exactly as if they were physical interfaces. The tunneling system must first be enabled by setting PermitTunnel to "yes" in the SSH server configuration file (`/etc/ssh/sshd_config`). When establishing the SSH connection, the creation of a tunnel must be explicitly requested with the -w any:any option (any can be replaced with the desired tun device number). This requires the user to have administrator privilege on both

sides, so as to be able to create the network device (in other words, the connection must be established as root).

Both methods for creating a virtual private network over SSH are quite straightforward. However, the VPN they provide is not the most efficient available; in particular, it does not handle high levels of traffic very well.

The explanation is that when a TCP/IP stack is encapsulated within a TCP/IP connection (for SSH), the TCP protocol is used twice, once for the SSH connection and once within the tunnel. This leads to problems, especially due to the way TCP adapts to network conditions by altering timeout delays. The following site describes the problem in more detail:

➨ `http://sites.inka.de/sites/bigred/devel/tcp-tcp.html`

VPNs over SSH should therefore be restricted to one-off tunnels with no performance constraints.

10.2.3. IPsec

IPsec, despite being the standard in IP VPNs, is rather more involved in its implementation. The IPsec engine itself is integrated in the Linux kernel; the required user-space parts, the control and configuration tools, are provided by the *ipsec-tools* package. In concrete terms, each host's `/etc/ipsec-tools.conf` contains the parameters for *IPsec tunnels* (or *Security Associations*, in the IPsec terminology) that the host is concerned with; the `/etc/init.d/setkey` script provides a way to start and stop a tunnel (each tunnel is a secure link to another host connected to the virtual private network). This file can be built by hand from the documentation provided by the `setkey(8)` manual page. However, explicitly writing the parameters for all hosts in a non-trivial set of machines quickly becomes an arduous task, since the number of tunnels grows fast. Installing an IKE daemon (for *IPsec Key Exchange*) such as *racoon* or *strongswan* makes the process much simpler by bringing administration together at a central point, and more secure by rotating the keys periodically.

In spite of its status as the reference, the complexity of setting up IPsec restricts its usage in practice. OpenVPN-based solutions will generally be preferred when the required tunnels are neither too many nor too dynamic.

CAUTION

IPsec and NAT

NATing firewalls and IPsec do not work well together: since IPsec signs the packets, any change on these packets that the firewall might perform will void the signature, and the packets will be rejected at their destination. Various IPsec implementations now include the *NAT-T* technique (for *NAT Traversal*), which basically encapsulates the IPsec packet within a standard UDP packet.

SECURITY

IPsec and firewalls

The standard mode of operation of IPsec involves data exchanges on UDP port 500 for key exchanges (also on UDP port 4500 in the case that NAT-T is in use). Moreover, IPsec packets use two dedicated IP protocols that the firewall must let through; reception of these packets is based on their protocol numbers, 50 (ESP) and 51 (AH).

10.2.4. PPTP

PPTP (for *Point-to-Point Tunneling Protocol*) uses two communication channels, one for control data and one for payload data; the latter uses the GRE protocol (*Generic Routing Encapsulation*). A standard PPP link is then set up over the data exchange channel.

Configuring the Client

The *pptp-linux* package contains an easily-configured PPTP client for Linux. The following instructions take their inspiration from the official documentation:

➡ http://pptpclient.sourceforge.net/howto-debian.phtml

The Falcot administrators created several files: `/etc/ppp/options.pptp`, `/etc/ppp/peers/falcot`, `/etc/ppp/ip-up.d/falcot`, and `/etc/ppp/ip-down.d/falcot`.

Example 10.2 *The `/etc/ppp/options.pptp` file*

```
# PPP options used for a PPTP connection
lock
noauth
nobsdcomp
nodeflate
```

Example 10.3 *The `/etc/ppp/peers/falcot` file*

```
# vpn.falcot.com is the PPTP server
pty "pptp vpn.falcot.com --nolaunchpppd"
# the connection will identify as the "vpn" user
user vpn
remotename pptp
# encryption is needed
require-mppe-128
file /etc/ppp/options.pptp
ipparam falcot
```

Example 10.4 *The `/etc/ppp/ip-up.d/falcot` file*

```
# Create the route to the Falcot network
if [ "$6" = "falcot" ]; then
  # 192.168.0.0/24 is the (remote) Falcot network
  route add -net 192.168.0.0 netmask 255.255.255.0 dev $1
fi
```

Example 10.5 *The /etc/ppp/ip-down.d/falcot file*

```
# Delete the route to the Falcot network
if [ "$6" = "falcot" ]; then
  # 192.168.0.0/24 is the (remote) Falcot network
  route del -net 192.168.0.0 netmask 255.255.255.0 dev $1
fi
```

SECURITY **MPPE**

Securing PPTP involves using the MPPE feature (*Microsoft Point-to-Point Encryption*), which is available in official Debian kernels as a module.

Configuring the Server

CAUTION **PPTP and firewalls**

Intermediate firewalls need to be configured to let through IP packets using protocol 47 (GRE). Moreover, the PPTP server's port 1723 needs to be open so that the communication channel can happen.

`pptpd` is the PPTP server for Linux. Its main configuration file, `/etc/pptpd.conf`, requires very few changes: *localip* (local IP address) and *remoteip* (remote IP address). In the example below, the PPTP server always uses the 192.168.0.199 address, and PPTP clients receive IP addresses from 192.168.0.200 to 192.168.0.250.

Example 10.6 *The /etc/pptpd.conf file*

```
# TAG: speed
#
#       Specifies the speed for the PPP daemon to talk at.
#
speed 115200

# TAG: option
#
#       Specifies the location of the PPP options file.
#       By default PPP looks in '/etc/ppp/options'
#
option /etc/ppp/pptpd-options

# TAG: debug
#
#       Turns on (more) debugging to syslog
#
# debug
```

```
# TAG: localip
# TAG: remoteip
#
#       Specifies the local and remote IP address ranges.
#
#       You can specify single IP addresses separated by commas or you can
#       specify ranges, or both. For example:
#
#               192.168.0.234,192.168.0.245-249,192.168.0.254
#
#       IMPORTANT RESTRICTIONS:
#
#       1. No spaces are permitted between commas or within addresses.
#
#       2. If you give more IP addresses than MAX_CONNECTIONS, it will
#          start at the beginning of the list and go until it gets
#          MAX_CONNECTIONS IPs. Others will be ignored.
#
#       3. No shortcuts in ranges! ie. 234-8 does not mean 234 to 238,
#          you must type 234-238 if you mean this.
#
#       4. If you give a single localIP, that's ok - all local IPs will
#          be set to the given one. You MUST still give at least one remote
#          IP for each simultaneous client.
#
#localip 192.168.0.234-238,192.168.0.245
#remoteip 192.168.1.234-238,192.168.1.245
#localip 10.0.1.1
#remoteip 10.0.1.2-100
localip 192.168.0.199
remoteip 192.168.0.200-250
```

The PPP configuration used by the PPTP server also requires a few changes in `/etc/ppp/pptpd-options`. The important parameters are the server name (pptp), the domain name (falcot.com), and the IP addresses for DNS and WINS servers.

Example 10.7 *The `/etc/ppp/pptpd-options` file*

```
## turn pppd syslog debugging on
#debug

## change 'servername' to whatever you specify as your server name in chap-secrets
name pptp
## change the domainname to your local domain
domain falcot.com

## these are reasonable defaults for WinXXXX clients
```

```
## for the security related settings
# The Debian pppd package now supports both MSCHAP and MPPE, so enable them
# here. Please note that the kernel support for MPPE must also be present!
auth
require-chap
require-mschap
require-mschap-v2
require-mppe-128

## Fill in your addresses
ms-dns 192.168.0.1
ms-wins 192.168.0.1

## Fill in your netmask
netmask 255.255.255.0

## some defaults
nodefaultroute
proxyarp
lock
```

The last step involves registering the vpn user (and the associated password) in the `/etc/ppp/chap-secrets` file. Contrary to other instances where an asterisk (*) would work, the server name must be filled explicitly here. Furthermore, Windows PPTP clients identify themselves under the *DOMAIN**USER* form, instead of only providing a user name. This explains why the file also mentions the FALCOT\\vpn user. It is also possible to specify individual IP addresses for users; an asterisk in this field specifies that dynamic addressing should be used.

Example 10.8 *The `/etc/ppp/chap-secrets` file*

```
# Secrets for authentication using CHAP
# client        server  secret      IP addresses
vpn             pptp    f@Lc3au     *
FALCOT\\vpn     pptp    f@Lc3au     *
```

SECURITY

PPTP vulnerabilities

Microsoft's first PPTP implementation drew severe criticism because it had many security vulnerabilities; most have since then been fixed in more recent versions. The configuration documented in this section uses the latest version of the protocol. Be aware though that removing some options (such as `require-mppe-128` and `require-mschap-v2`) would make the service vulnerable again.

10.3. Quality of Service

10.3.1. Principle and Mechanism

Quality of Service (or *QoS* for short) refers to a set of techniques that guarantee or improve the quality of the service provided to applications. The most popular such technique involves classifying the network traffic into categories, and differentiating the handling of traffic according to which category it belongs to. The main application of this differentiated services concept is *traffic shaping*, which limits the data transmission rates for connections related to some services and/or hosts so as not to saturate the available bandwidth and starve important other services. Traffic shaping is a particularly good fit for TCP traffic, since this protocol automatically adapts to available bandwidth.

It is also possible to alter the priorities on traffic, which allows prioritizing packets related to interactive services (such as `ssh` and `telnet`) or to services that only deal with small blocks of data.

The Debian kernels include the features required for QoS along with their associated modules. These modules are many, and each of them provides a different service, most notably by way of special schedulers for the queues of IP packets; the wide range of available scheduler behaviors spans the whole range of possible requirements.

CULTURE

LARTC — *Linux Advanced Routing & Traffic Control*

The *Linux Advanced Routing & Traffic Control* HOWTO is the reference document covering everything there is to know about network quality of service.

➨ `http://www.lartc.org/howto/`

10.3.2. Configuring and Implementing

QoS parameters are set through the `tc` command (provided by the *iproute* package). Since its interface is quite complex, using higher-level tools is recommended.

Reducing Latencies: `wondershaper`

The main purpose of `wondershaper` (in the similarly-named package) is to minimize latencies independent of network load. This is achieved by limiting total traffic to a value that falls just short of the link saturation value.

Once a network interface is configured, setting up this traffic limitation is achieved by running `wondershaper` *`interface download_rate upload_rate`*. The interface can be `eth0` or `ppp0` for example, and both rates are expressed in kilobits per second. The `wondershaper remove` *`interface`* command disables traffic control on the specified interface.

For an Ethernet connection, this script is best called right after the interface is configured. This is done by adding `up` and `down` directives to the `/etc/network/interfaces` file allowing de-

clared commands to be run, respectively, after the interface is configured and before it is deconfigured. For example:

Example 10.9 *Changes in the `/etc/network/interfaces` file*

```
iface eth0 inet dhcp
    up /sbin/wondershaper eth0 500 100
    down /sbin/wondershaper remove eth0
```

In the PPP case, creating a script that calls `wondershaper` in `/etc/ppp/ip-up.d/` will enable traffic control as soon as the connection is up.

GOING FURTHER
Optimal configuration

The `/usr/share/doc/wondershaper/README.Debian.gz` file describes, in some detail, the configuration method recommended by the package maintainer. In particular, it advises measuring the download and upload speeds so as to best evaluate real limits.

Standard Configuration

Barring a specific QoS configuration, the Linux kernel uses the pfifo_fast queue scheduler, which provides a few interesting features by itself. The priority of each processed IP packet is based on the ToS field (*Type of Service*) of this packet; modifying this field is enough to take advantage of the scheduling features. There are five possible values:

- Normal-Service (0);
- Minimize-Cost (2);
- Maximize-Reliability (4);
- Maximize-Throughput (8);
- Minimize-Delay (16).

The ToS field can be set by applications that generate IP packets, or modified on the fly by *netfilter*. The following rules are sufficient to increase responsiveness for a server's SSH service:

```
iptables -t mangle -A PREROUTING -p tcp --sport ssh -j TOS --set-tos Minimize-Delay
iptables -t mangle -A PREROUTING -p tcp --dport ssh -j TOS --set-tos Minimize-Delay
```

10.4. Dynamic Routing

The reference tool for dynamic routing is currently `quagga`, from the similarly-named package; it used to be `zebra` until development of the latter stopped. However, `quagga` kept the names of the programs for compatibility reasons which explains the `zebra` commands below.

BACK TO BASICS

Dynamic routing

Dynamic routing allows routers to adjust, in real time, the paths used for transmitting IP packets. Each protocol involves its own method of defining routes (shortest path, use routes advertised by peers, and so on).

In the Linux kernel, a route links a network device to a set of machines that can be reached through this device. The `route` command defines new routes and displays existing ones.

Quagga is a set of daemons cooperating to define the routing tables to be used by the Linux kernel; each routing protocol (most notably BGP, OSPF and RIP) provides its own daemon. The `zebra` daemon collects information from other daemons and handles static routing tables accordingly. The other daemons are known as `bgpd`, `ospfd`, `ospf6d`, `ripd`, `ripngd`, `isisd`, and `babeld`.

Daemons are enabled by editing the `/etc/quagga/daemons` file and creating the appropriate configuration file in `/etc/quagga/`; this configuration file must be named after the daemon, with a `.conf` extension, and belong to the quagga user and the quaggavty group, in order for the `/etc/init.d/quagga` script to invoke the daemon.

The configuration of each of these daemons requires knowledge of the routing protocol in question. These protocols cannot be described in detail here, but the *quagga-doc* provides ample explanation in the form of an `info` file. The same contents may be more easily browsed as HTML on the Quagga website:

➡ `http://www.nongnu.org/quagga/docs/docs-info.html`

In addition, the syntax is very close to a standard router's configuration interface, and network administrators will adapt quickly to `quagga`.

IN PRACTICE

OSPF, BGP or RIP?

OSPF is generally the best protocol to use for dynamic routing on private networks, but BGP is more common for Internet-wide routing. RIP is rather ancient, and hardly used anymore.

10.5. IPv6

IPv6, successor to IPv4, is a new version of the IP protocol designed to fix its flaws, most notably the scarcity of available IP addresses. This protocol handles the network layer; its purpose is to provide a way to address machines, to convey data to their intended destination, and to handle data fragmentation if needed (in other words, to split packets into chunks with a size that depends on the network links to be used on the path and to reassemble the chunks in their proper order on arrival).

Debian kernels include IPv6 handling in the core kernel (with the exception of some architectures that have it compiled as a module named ipv6). Basic tools such as `ping` and `traceroute` have their IPv6 equivalents in `ping6` and `traceroute6`, available respectively in the *iputils-ping* and *iputils-tracepath* packages.

The IPv6 network is configured similarly to IPv4, in `/etc/network/interfaces`. But if you want that network to be globally available, you must ensure that you have an IPv6-capable router relaying traffic to the global IPv6 network.

Example 10.10 *Example of IPv6 configuration*

```
iface eth0 inet6 static
    address 2001:db8:1234:5::1:1
    netmask 64
    # Disabling auto-configuration
    # autoconf 0
    # The router is auto-configured and has no fixed address
    # (accept_ra 1). If it had:
    # gateway 2001:db8:1234:5::1
```

IPv6 subnets usually have a netmask of 64 bits. This means that 2^{64} distinct addresses exist within the subnet. This allows Stateless Address Autoconfiguration (SLAAC) to pick an address based on the network interface's MAC address. By default, if SLAAC is activated in your network and IPv6 on your computer, the kernel will automatically find IPv6 routers and configure the network interfaces.

This behavior may have privacy implications. If you switch networks frequently, e.g. with a laptop, you might not want your MAC address being a part of your public IPv6 address. This makes it easy to identify the same device across networks. A solution to this are IPv6 privacy extensions (which Debian enables by default if IPv6 connectivity is detected during initial installation), which will assign an additional randomly generated address to the interface, periodically change them and prefer them for outgoing connections. Incoming connections can still use the address generated by SLAAC. The following example, for use in `/etc/network/interfaces`, activates these privacy extensions.

Example 10.11 *IPv6 privacy extensions*

```
iface eth0 inet6 auto
    # Prefer the randomly assigned addresses for outgoing connections.
    privext 2
```

TIP

Programs built with IPv6

Many pieces of software need to be adapted to handle IPv6. Most of the packages in Debian have been adapted already, but not all. If your favorite package does not work with IPv6 yet, you can ask for help on the *debian-ipv6* mailing-list. They might know about an IPv6-aware replacement and could file a bug to get the issue properly tracked.

➨ http://lists.debian.org/debian-ipv6/

IPv6 connections can be restricted, in the same fashion as for IPv4: the standard Debian kernels include an adaptation of *netfilter* for IPv6. This IPv6-enabled *netfilter* is configured in a similar fashion to its IPv4 counterpart, except the program to use is `ip6tables` instead of `iptables`.

10.5.1. Tunneling

CAUTION

IPv6 tunneling and firewalls

IPv6 tunneling over IPv4 (as opposed to native IPv6) requires the firewall to accept the traffic, which uses IPv4 protocol number 41.

If a native IPv6 connection is not available, the fallback method is to use a tunnel over IPv4. Gogo6 is one (free) provider of such tunnels:

➡ `http://www.gogo6.com/freenet6/tunnelbroker`

To use a Freenet6 tunnel, you need to register for a Freenet6 Pro account on the website, then install the *gogoc* package and configure the tunnel. This requires editing the `/etc/gogoc/gogoc.conf` file: userid and password lines received by e-mail should be added, and server should be replaced with authenticated.freenet6.net.

IPv6 connectivity is proposed to all machines on a local network by adding the three following directives to the `/etc/gogoc/gogoc.conf` file (assuming the local network is connected to the eth0 interface):

```
host_type=router
prefixlen=56
if_prefix=eth0
```

The machine then becomes the access router for a subnet with a 56-bit prefix. Once the tunnel is aware of this change, the local network must be told about it; this implies installing the `radvd` daemon (from the similarly-named package). This IPv6 configuration daemon has a role similar to `dhcpd` in the IPv4 world.

The `/etc/radvd.conf` configuration file must then be created (see `/usr/share/doc/radvd/examples/simple-radvd.conf` as a starting point). In our case, the only required change is the prefix, which needs to be replaced with the one provided by Freenet6; it can be found in the output of the `ifconfig` command, in the block concerning the `tun` interface.

Then run `service gogoc restart` and `service radvd start`, and the IPv6 network should work.

10.6. Domain Name Servers (DNS)

10.6.1. Principle and Mechanism

The *Domain Name Service* (DNS) is a fundamental component of the Internet: it maps host names to IP addresses (and vice-versa), which allows the use of www.debian.org instead of 5.153.231.4 or 2001:41c8:1000:21::21:4.

DNS records are organized in zones; each zone matches either a domain (or a subdomain) or an IP address range (since IP addresses are generally allocated in consecutive ranges). A primary server is authoritative on the contents of a zone; secondary servers, usually hosted on separate machines, provide regularly refreshed copies of the primary zone.

Each zone can contain records of various kinds (*Resource Records*):

- A: IPv4 address.
- CNAME: alias (*canonical name*).
- MX: *mail exchange*, an email server. This information is used by other email servers to find where to send email addressed to a given address. Each MX record has a priority. The highest-priority server (with the lowest number) is tried first (see sidebar "SMTP" page 252); other servers are contacted in order of decreasing priority if the first one does not reply.
- PTR: mapping of an IP address to a name. Such a record is stored in a "reverse DNS" zone named after the IP address range. For example, 1.168.192.in-addr.arpa is the zone containing the reverse mapping for all addresses in the 192.168.1.0/24 range.
- AAAA: IPv6 address.
- NS: maps a name to a name server. Each domain must have at least one NS record. These records point at a DNS server that can answer queries concerning this domain; they usually point at the primary and secondary servers for the domain. These records also allow DNS delegation; for instance, the falcot.com zone can include an NS record for internal.falcot.com, which means that the internal.falcot.com zone is handled by another server. Of course, this server must declare an internal.falcot.com zone.

The reference name server, Bind, was developed and is maintained by ISC (*Internet Software Consortium*). It is provided in Debian by the *bind9* package. Version 9 brings two major changes compared to previous versions. First, the DNS server can now run under an unprivileged user, so that a security vulnerability in the server does not grant root privileges to the attacker (as was seen repeatedly with versions 8.x).

Furthermore, Bind supports the DNSSEC standard for signing (and therefore authenticating) DNS records, which allows blocking any spoofing of this data during man-in-the-middle attacks.

CULTURE

DNSSEC

The DNSSEC norm is quite complex; this partly explains why it is not in widespread usage yet (even if it perfectly coexists with DNS servers unaware of DNSSEC). To understand all the ins and outs, you should check the following article.

➨ http://en.wikipedia.org/wiki/Domain_Name_System_Security_Extensions

10.6.2. Configuring

Configuration files for `bind`, irrespective of version, have the same structure.

The Falcot administrators created a primary falcot.com zone to store information related to this domain, and a 168.192.in-addr.arpa zone for reverse mapping of IP addresses in the local networks.

CAUTION

Names of reverse zones

Reverse zones have a particular name. The zone covering the `192.168.0.0/16` network needs to be named `168.192.in-addr.arpa`: the IP address components are reversed, and followed by the `in-addr.arpa` suffix.

For IPv6 networks, the suffix is `ip6.arpa` and the IP address components which are reversed are each character in the full hexadecimal representation of the IP address. As such, the `2001:0bc8:31a0::/48` network would use a zone named `0.a.1.3.8.c.b.0.1.0.0.2.ip6.arpa`.

TIP

Testing the DNS server

The `host` command (in the *bind9-host* package) queries a DNS server, and can be used to test the server configuration. For example, `host machine.falcot.com localhost` checks the local server's reply for the `machine.falcot.com` query. `host` *`ipaddress`* `localhost` tests the reverse resolution.

The following configuration excerpts, taken from the Falcot files, can serve as starting points to configure a DNS server:

Example 10.12 *Excerpt of `/etc/bind/named.conf.local`*

```
zone "falcot.com" {
        type master;
        file "/etc/bind/db.falcot.com";
        allow-query { any; };
        allow-transfer {
                195.20.105.149/32 ; // ns0.xname.org
                193.23.158.13/32 ; // ns1.xname.org
        };
};

zone "internal.falcot.com" {
        type master;
```

```
        file "/etc/bind/db.internal.falcot.com";
        allow-query { 192.168.0.0/16; };
};

zone "168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/db.192.168";
        allow-query { 192.168.0.0/16; };
};
```

Example 10.13 *Excerpt of `/etc/bind/db.falcot.com`*

```
; falcot.com Zone
; admin.falcot.com. => zone contact: admin@falcot.com
$TTL    604800
@       IN      SOA     falcot.com. admin.falcot.com. (
                        20040121        ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
; The @ refers to the zone name ("falcot.com" here)
; or to $ORIGIN if that directive has been used
;
@       IN      NS      ns
@       IN      NS      ns0.xname.org.

internal IN      NS      192.168.0.2

@       IN      A       212.94.201.10
@       IN      MX      5 mail
@       IN      MX      10 mail2

ns      IN      A       212.94.201.10
mail    IN      A       212.94.201.10
mail2   IN      A       212.94.201.11
www     IN      A       212.94.201.11

dns     IN      CNAME   ns
```

CAUTION

Syntax of a name

The syntax of machine names follows strict rules. For instance, `machine` implies `machine.`*`domain`*. If the domain name should not be appended to a name, said name must be written as `machine.` (with a dot as suffix). Indicating a DNS name outside the current domain therefore requires a syntax such as `machine.otherdomain.com.` (with the final dot).

Example 10.14 *Excerpt of `/etc/bind/db.192.168`*

```
; Reverse zone for 192.168.0.0/16
; admin.falcot.com. => zone contact: admin@falcot.com
$TTL    604800
@       IN      SOA     ns.internal.falcot.com. admin.falcot.com. (
                        20040121        ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL

        IN      NS      ns.internal.falcot.com.

; 192.168.0.1 -> arrakis
1.0     IN      PTR     arrakis.internal.falcot.com.
; 192.168.0.2 -> neptune
2.0     IN      PTR     neptune.internal.falcot.com.

; 192.168.3.1 -> pau
1.3     IN      PTR     pau.internal.falcot.com.
```

10.7. DHCP

DHCP (for *Dynamic Host Configuration Protocol*) is a protocol by which a machine can automatically get its network configuration when it boots. This allows centralizing the management of network configurations, and ensuring that all desktop machines get similar settings.

A DHCP server provides many network-related parameters. The most common of these is an IP address and the network where the machine belongs, but it can also provide other information, such as DNS servers, WINS servers, NTP servers, and so on.

The Internet Software Consortium (also involved in developing `bind`) is the main author of the DHCP server. The matching Debian package is *isc-dhcp-server*.

10.7.1. Configuring

The first elements that need to be edited in the DHCP server configuration file (`/etc/dhcp/dhcpd.conf`) are the domain name and the DNS servers. If this server is alone on the local network (as defined by the broadcast propagation), the `authoritative` directive must also be enabled (or uncommented). One also needs to create a `subnet` section describing the local network and the configuration information to be provided. The following example fits a 192.168.0.0/24 local network with a router at 192.168.0.1 serving as the gateway. Available IP addresses are in the range 192.168.0.128 to 192.168.0.254.

Example 10.15 *Excerpt of /etc/dhcp/dhcpd.conf*

```
#
# Sample configuration file for ISC dhcpd for Debian
#

# The ddns-updates-style parameter controls whether or not the server will
# attempt to do a DNS update when a lease is confirmed. We default to the
# behavior of the version 2 packages ('none', since DHCP v2 didn't
# have support for DDNS.)
ddns-update-style interim;

# option definitions common to all supported networks...
option domain-name "internal.falcot.com";
option domain-name-servers ns.internal.falcot.com;

default-lease-time 600;
max-lease-time 7200;

# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
authoritative;

# Use this to send dhcp log messages to a different log file (you also
# have to hack syslog.conf to complete the redirection).
log-facility local7;

# My subnet
subnet 192.168.0.0 netmask 255.255.255.0 {
    option routers 192.168.0.1;
    option broadcast-address 192.168.0.255;
    range 192.168.0.128 192.168.0.254;
    ddns-domainname "internal.falcot.com";
}
```

10.7.2. DHCP and DNS

A nice feature is the automated registering of DHCP clients in the DNS zone, so that each machine gets a significant name (rather than something impersonal such as machine-192-168-0-131.internal.falcot.com). Using this feature requires configuring the DNS server to accept updates to the internal.falcot.com DNS zone from the DHCP server, and configuring the latter to submit updates for each registration.

In the bind case, the allow-update directive needs to be added to each of the zones that the DHCP server is to edit (the one for the internal.falcot.com domain, and the reverse zone). This directive

lists the IP addresses allowed to perform these updates; it should therefore contain the possible addresses of the DHCP server (both the local address and the public address, if appropriate).

```
allow-update { 127.0.0.1 192.168.0.1 212.94.201.10 !any };
```

Beware! A zone that can be modified *will* be changed by `bind`, and the latter will overwrite its configuration files at regular intervals. Since this automated procedure produces files that are less human-readable than manually-written ones, the Falcot administrators handle the internal.falcot.com domain with a delegated DNS server; this means the falcot.com zone file stays firmly under their manual control.

The DHCP server configuration excerpt above already includes the directives required for DNS zone updates: they are the ddns-update-style interim; and ddns-domain-name "internal.falcot.com"; lines in the block describing the subnet.

10.8. Network Diagnosis Tools

When a network application does not run as expected, it is important to be able to look under the hood. Even when everything seems to run smoothly, running a network diagnosis can help ensure everything is working as it should. Several diagnosis tools exists for this purpose; each one operates on a different level.

10.8.1. Local Diagnosis: `netstat`

Let's first mention the `netstat` command (in the *net-tools* package); it displays an instant summary of a machine's network activity. When invoked with no argument, this command lists all open connections; this list can be very verbose since it includes many Unix-domain sockets (widely used by daemons) which do not involve the network at all (for example, dbus communication, X11 traffic, and communications between virtual filesystems and the desktop).

Common invocations therefore use options that alter `netstat`'s behavior. The most frequently used options include:

- -t, which filters the results to only include TCP connections;
- -u, which works similarly for UDP connections; these options are not mutually exclusive, and one of them is enough to stop displaying Unix-domain connections;
- -a, to also list listening sockets (waiting for incoming connections);
- -n, to display the results numerically: IP addresses (no DNS resolution), port numbers (no aliases as defined in `/etc/services`) and user ids (no login names);
- -p, to list the processes involved; this option is only useful when `netstat` is run as root, since normal users will only see their own processes;
- -c, to continuously refresh the list of connections.

Other options, documented in the `netstat(8)` manual page, provide an even finer control over the displayed results. In practice, the first five options are so often used together that systems and network administrators practically acquired `netstat -tupan` as a reflex. Typical results, on a lightly loaded machine, may look like the following:

```
# netstat -tupan
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      397/rpcbind
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      431/sshd
tcp        0      0 0.0.0.0:36568           0.0.0.0:*               LISTEN      407/rpc.statd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      762/exim4
tcp        0    272 192.168.1.242:22        192.168.1.129:44452     ESTABLISHED 1172/sshd: roland [
tcp6       0      0 :::111                  :::*                    LISTEN      397/rpcbind
tcp6       0      0 :::22                   :::*                    LISTEN      431/sshd
tcp6       0      0 ::1:25                  :::*                    LISTEN      762/exim4
tcp6       0      0 :::35210                :::*                    LISTEN      407/rpc.statd
udp        0      0 0.0.0.0:39376           0.0.0.0:*                           916/dhclient
udp        0      0 0.0.0.0:996             0.0.0.0:*                           397/rpcbind
udp        0      0 127.0.0.1:1007          0.0.0.0:*                           407/rpc.statd
udp        0      0 0.0.0.0:68              0.0.0.0:*                           916/dhclient
udp        0      0 0.0.0.0:48720           0.0.0.0:*                           451/avahi-daemon: r
udp        0      0 0.0.0.0:111             0.0.0.0:*                           397/rpcbind
udp        0      0 192.168.1.242:123       0.0.0.0:*                           539/ntpd
udp        0      0 127.0.0.1:123           0.0.0.0:*                           539/ntpd
udp        0      0 0.0.0.0:123             0.0.0.0:*                           539/ntpd
udp        0      0 0.0.0.0:5353            0.0.0.0:*                           451/avahi-daemon: r
udp        0      0 0.0.0.0:39172           0.0.0.0:*                           407/rpc.statd
udp6       0      0 :::996                  :::*                                397/rpcbind
udp6       0      0 :::34277                :::*                                407/rpc.statd
udp6       0      0 :::54852                :::*                                916/dhclient
udp6       0      0 :::111                  :::*                                397/rpcbind
udp6       0      0 :::38007                :::*                                451/avahi-daemon: r
udp6       0      0 fe80::5054:ff:fe99::123 :::*                                539/ntpd
udp6       0      0 2001:bc8:3a7e:210:a:123 :::*                                539/ntpd
udp6       0      0 2001:bc8:3a7e:210:5:123 :::*                                539/ntpd
udp6       0      0 ::1:123                 :::*                                539/ntpd
udp6       0      0 :::123                  :::*                                539/ntpd
udp6       0      0 :::5353                 :::*                                451/avahi-daemon: r
```

As expected, this lists established connections, two SSH connections in this case, and applications waiting for incoming connections (listed as LISTEN), notably the Exim4 email server listening on port 25.

10.8.2. Remote Diagnosis: `nmap`

`nmap` (in the similarly-named package) is, in a way, the remote equivalent for `netstat`. It can scan a set of "well-known" ports for one or several remote servers, and list the ports where an application is found to answer to incoming connections. Furthermore, `nmap` is able to identify some of these applications, sometimes even their version number. The counterpart of this tool is that, since it runs remotely, it cannot provide information on processes or users; however, it can operate on several targets at once.

A typical `nmap` invocation only uses the -A option (so that `nmap` attempts to identify the versions of the server software it finds) followed by one or more IP addresses or DNS names of machines to scan. Again, many more options exist to finely control the behavior of `nmap`; please refer to the documentation in the `nmap(1)` manual page.

```
# nmap mirtuel

Starting Nmap 6.47 ( http://nmap.org ) at 2015-03-09 16:46 CET
Nmap scan report for mirtuel (192.168.1.242)
Host is up (0.000013s latency).
rDNS record for 192.168.1.242: mirtuel.internal.placard.fr.eu.org
Not shown: 998 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
111/tcp open  rpcbind

Nmap done: 1 IP address (1 host up) scanned in 2.41 seconds
# nmap -A localhost

Starting Nmap 6.47 ( http://nmap.org ) at 2015-03-09 16:46 CET
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000013s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 997 closed ports
PORT    STATE SERVICE VERSION
22/tcp  open  ssh     OpenSSH 6.7p1 Debian 3 (protocol 2.0)
|_ssh-hostkey: ERROR: Script execution failed (use -d to debug)
25/tcp  open  smtp    Exim smtpd 4.84
| smtp-commands: mirtuel Hello localhost [127.0.0.1], SIZE 52428800, 8BITMIME,
       PIPELINING, HELP,
|_ Commands supported: AUTH HELO EHLO MAIL RCPT DATA NOOP QUIT RSET HELP
111/tcp open  rpcbind 2-4 (RPC #100000)
| rpcinfo:
|   program version   port/proto  service
|   100000  2,3,4        111/tcp  rpcbind
|   100000  2,3,4        111/udp  rpcbind
|   100024  1          36568/tcp  status
|_  100024  1          39172/udp  status
Device type: general purpose
Running: Linux 3.X
OS CPE: cpe:/o:linux:linux_kernel:3
OS details: Linux 3.7 - 3.15
Network Distance: 0 hops
Service Info: Host: mirtuel; OS: Linux; CPE: cpe:/o:linux:linux_kernel

OS and Service detection performed. Please report any incorrect results at http
       ://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.54 seconds
```

As expected, the SSH and Exim4 applications are listed. Note that not all applications listen on all IP addresses; since Exim4 is only accessible on the lo loopback interface, it only appears during an analysis of localhost and not when scanning mirtuel (which maps to the eth0 interface on the same machine).

Sometimes, one needs to look at what actually goes on the wire, packet by packet. These cases call for a “frame analyzer”, more widely known as a *sniffer*. Such a tool observes all the packets that reach a given network interface, and displays them in a user-friendly way.

The venerable tool in this domain is `tcpdump`, available as a standard tool on a wide range of platforms. It allows many kinds of network traffic capture, but the representation of this traffic stays rather obscure. We will therefore not describe it in further detail.

A more recent (and more modern) tool, `wireshark` (in the *wireshark* package), has become the new reference in network traffic analysis due to its many decoding modules that allow for a simplified analysis of the captured packets. The packets are displayed graphically with an organization based on the protocol layers. This allows a user to visualize all protocols involved in a packet. For example, given a packet containing an HTTP request, `wireshark` displays, separately, the information concerning the physical layer, the Ethernet layer, the IP packet information, the TCP connection parameters, and finally the HTTP request itself.

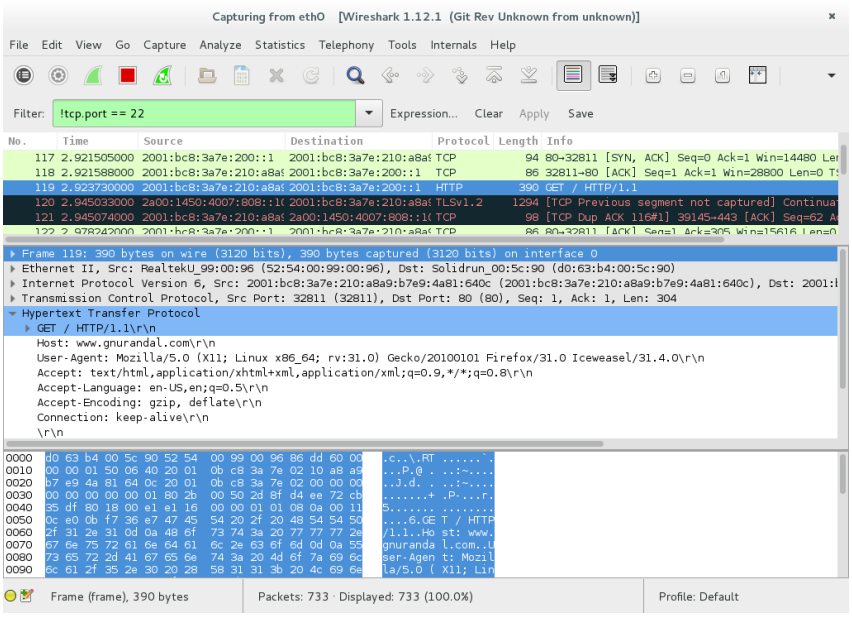

Figure 10.1 *The `wireshark` network traffic analyzer*

In our example, the packets traveling over SSH are filtered out (with the !tcp.port ==22 filter). The packet currently displayed was developed at the HTTP layer.

TIP

`wireshark` with no graphical interface: `tshark`

When one cannot run a graphical interface, or does not wish to do so for whatever reason, a text-only version of `wireshark` also exists under the name `tshark` (in a separate *tshark* package). Most of the capture and decoding features are still available, but the lack of a graphical interface necessarily limits the interactions with the program (filtering packets after they've been captured, tracking of a given TCP connection, and so on). It can still be used as a first approach. If further manipulations are intended and require the graphical interface, the packets can be saved to a file and this file can be loaded into a graphical `wireshark` running on another machine.

Keywords

Postfix
Apache
NFS
Samba
Squid
OpenLDAP
SIP