The LINUX Network Administrators' Guide

For Britta

Legal Notice

Contents

List of Figures

List of Abbreviations

The main task in Networking is to remember what all the abbreviations one encounters really mean. Here's a list of those used frequently throughout the guide:

ACU	Automatic Call Unit. A modem.[1]
ARP	Address Resolution Protocol. Used to map IP addresses to Ethernet addresses.
ARPA	Advanced Research Project Agency.
BBS	Bulletin Board System. Dial-up mailbox system.
BGP	Border Gateway Protocol. A protocol for exchanging routing information between autonomous systems.
BIND	Berkeley Interned Name Domain. An implementation of a DNS server.
BNU	Basic Networking Utilities. A popular UUCP version (also called HoneyDanBer UUCP).
BSD	Berkeley Source Distribution. A Un$\star$x flavor.
CCITT	International organization of postal services.
CSLIP	Compressed Serial Line IP. An protocol for exchanging IP packets over a serial line, using header compression.
DNS	Domain name system. Internet standard for mapping of host names to IP addresses.
EGP	External Gateway Protocol. A protocol for exchanging routing information between autonomous systems.

[1] Alternatively: A teenager with a telephone.

FQDN	Fully Qualified Domain Name. The hostname with the full domain name.
FTP	File Transfer Protocol.
FYI	"For Your Information." Series of documents with informal information on Internet topics.
GMU	Groucho Marx University. Fictitious University used throughout this book.
ICMP	Internet Control Message Protocol. A networking protocol.
IETF	Internet Engineering Task Force.
IP	Internet Protocol. A networking protocol.
ISO	International Standards Organization.
ISDN	Integrated Services Digital Network. New telecommunications technology using digital circuitry instead of analog.
MX	Mail Exchanger. A DNS resource record type.
NFS	Network File System. Standard for accessing data on remote disks transparently.
NIS	Network Information System. An RPC-based software for network-wide file-sharing.
NNTP	Network News Transfer Protocol.
OSI	Open Systems Interface. An ISO standard on network software.
PLIP	Parallel Line IP. A protocol for exchanging IP packets over a parallel line or a printer port.
TCP	Transfer Control Protocol. A networking protocol.
TCP/IP	Sloppy description of the Internet protocol suite as a whole.
RARP	Reverse Address Resolution Protocol. Permits hosts to find out their IP address at boot time.
RFC	Request For Comments. Series of documents describing Internet standards.
RPC	Remote Procedure Call. Protocol for executing procdures inside a process on a remote host.

RIP	Routing Information Protocol. Routing protocol used inside LANs (Autonomous Systems).
SLIP	Serial Line IP. A protocol for exchanging IP packets over a serial line.
SMTP	Simple Mail Transfer Protocol.
SOA	Start of Authority. A DNS resource record type.
UDP	User Datagram Protocol. A networking protocol.
UUCP	Unix to Unix Copy. A suite of network transport commands for dial-up networks.
YP	Yellow Pages. An older name for NIS.

Foreword

When I first heard rumors of LINUX, I was struggling with Andrew Tanenbaum's Minix on my old Atari. Running a multitasking system on a M 68000 is not wholly satisfactory, which was not the fault of Minix but rather that of said architecture. I finally figured out that it was the time to upgrade. I had already worked on Un⋆ces and was looking for a machine which was not too expensive, but would allow me to run a Un⋆xoid operating system which was not too expensive either. Although hedging a deep distrust of PC's (after all, aren't they all DOS machines inside?), I decided to try LINUX, and have not regretted it since.

After going through a number of ups and downs, like file system crashes, occasional core dumps, overwritten password files, etc, my LINUX setup has settled in the meanwhile and has become a stable environment. The kernel and C library have become that good that most standard software may be compiled with no more effort than is required on any other mainstream Un⋆x system, and a broad assortment of packaged LINUX distributions allows you to almost drop it onto your hard disk and start playing.

The only thing that occasinally gets in the way of LINUX enthusiasts is what may be called the Great Information Void — a phenomenon that drags the knowledge-thirsty user into a vortex `README`'s, `FAQ`'s and other files spelt in capital letters which help you solve many problems, but rarely give you the whole picture. What is lacking is some sort of documentation that allows users to understand the mechanism behind everything, and enables them to work things out themselves.

To this end, a couple of people formed the LINUX Documentation Project (LDP) in late 1992, which aims at putting together a coherent set of manuals. Stopping short of answering questions like "How?", or "Why?', or "What's the meaning of life, universe, and all the rest?", these manuals attempt to cover most aspects of running and using a LINUX system. This book is part of the LDP series and deals with network management and administration.

Without networking, there wouldn't be any LINUX today. One of the main factors in the LINUX experience — apart from the determination of its contributors — is that everything can be made available quickly, and that developers and users may communicate with ease.

All this is made possible through the cooperation of unnumbered sites in networks, and the cooperation of these networks among each other. (It also bears well remembering that these services are often administered by people in their spare time.)

Thus, there has always been a special interest among the LINUX community to bring networking to LINUX. From an early stage, UUCP-based software was available, which allowed to connect your LINUX machine to one of the networks using a dialup-connection. Later came the drive to provide LINUX with the necessary kernel functionality and user software that allows participation in local TCP/IP-based networks, using Ethernet, etc, which even allows to get your machine on the Internet. This effort is still under way, but is rapidly evolving.

This manual tries to cover both types of networking, introducing you to the basics, guiding you through the setup stages, and finally describes how to set up and run a number of applications, including NFS, electronic mail, and Usenet news.

Organization of this Book

This book is organized in the following way:

Chapter 1 gives you an introduction to networking in general, working out the concepts and terminology. If you know what the OSI model is, you might want to skip this.

Chapter 2 discusses the issues involved in TCP/IP-based networking. This works out the concepts introduced in Chapter 1.

Chapter 3 gets down to the very basics of TCP/IP-networking in LINUX. It includes a tour of the kernel's networking interfaces, and describes their configuration. It also covers setup of the resolving library, dealing with both simple hosts tables and setup of a name server.

Chapter 4 introduces you to various network applications that can be found on most Un⋆xish systems, like `inetd` and the "`r`" command suite (`rlogin`, `rsh`, etc.).

Chapter 5 deals with configuring NFS, the Networking File System.

Chapter 6 describes how to set up your serial hardware for dialling out and in.

Chapter 7 shows you how to manage Taylor UUCP. For those who have never used UUCP before, and know next to nothing on how it works, there is a short tour of its major principles.

Chapter 8 gives an overview of electronic mail, and the ideas involved in it: address formats, routing, and the whole lot of it. Part of the chapter is devoted to setting up `elm`, a standard mail user interface included in almost every Un⋆x nowadays.

Chapter 9 guides you through the setup of `smail` on your system. If you want to echange mail with other sites, you either have to run this, or `sendmail`.

Chapter 10 is a twin of chapter 9, in that it describes the setup of the `sendmail` mail transport agent.

Chapter 11 covers the basic ideas of netnews, and the setup of Cnews. Also has a short section on using NNTP.

Chapter 12 is a tour de force on newsreader configuration. It tries to cover the basic tasks needed in administering `tin`, `trn`, and `nn`.

Appendix A describes how to build a so-called "null printer cable" for use with the PLIP parallel interface IP protocol decsribed in chapter 3.

Appendix B contains a copy of the GNU General Public License.

The book ends with a annotated bibliography that contains a list of books recommended for further reading.

The Linux Documentation Project

The Linux Documentation Project, or LDP, is a loose team of writers, proofreaders, and editors who are working together to provide complete documentation for the Linux operating system. The overall coordinator of the project is Matt Welsh, who is heavily aided by Lars Wirzenius and Michael K. Johnson.

This manual is one in a set of several being distributed by the LDP, including a Linux Users' Guide, System Administrators' Guide, Network Administrators' Guide, and Kernel Hackers' Guide. These manuals are all available in LaTeX source format, `.dvi` format, and postscript output by anonymous FTP from `nic.funet.fi`, in the directory `/pub/OS/Linux/doc/doc-project`, and from `tsx-11.mit.edu`, in the directory `/pub/linux/docs/guides`.

We encourage anyone with a penchant for writing or editing to join us in improving Linux documentation. If you have Internet e-mail access, you can join the `DOC` channel of the `Linux-Activists` mailing list by sending mail to

```
linux-activists-request@niksula.hut.fi
```

with the line

```
X-Mn-Admin: join DOC
```

in the header or as the first line of the message body. To leave the channel, send a message to the same address, including the line

```
X-Mn-Admin: leave DOC
```

Typographical Conventions

In writing this book, a number of typographical conventions were employed to mark shell commands, variable arguments, etc. They are explained below.

Bold Font Used to mark **new concepts**, **WARNINGS**, and **keywords** in a language.

Italics Font Used for *emphasis* in text, and occasionally for quotes or introductions at the beginning of a section.

`Typewriter Font`

Used to represent screen interaction, as in

```
# ls -l /bin/cp
-rwxr-xr-x  1 root    wheel    12104 Sep 25 15:53 /bin/cp
```

Also used for code examples, whether it is "C" code, a shell script, or something else, and to display general files, such as configuration files. When necessary for clarity's sake, these examples or figures will be enclosed in thin boxes.

Typewriter Slanted Font

Used to mark **meta-variables** in the text, especially in representations of the command line. For example,

`ls -l` *foo*

where *foo* would "stand for" a filename, such as `/bin/cp`.

Key Represents a key to press. You will often see it in this form:

Press return to continue.

◇ A diamond in the margin, like a black diamond on a ski hill, marks "danger" or "caution." Read paragraphs marked this way carefully.

`$` and `#` When preceding a shell command to be typed, these denote the shell prompt. The '`$`' symbol is used as when the command may be executed as a nomral user; '#' means that the command requires super user privilieges.

Thanks

This book owes very much to the numerous people who took the time to proofread it and helped iron out many mistakes, both grammatical and technical. The most vigorous among them are Michael K. Johnson, Iain Lea, and Wolfgang Michaelis. I also wish to thank Ian Taylor for sparing the time to read through the UUCP chapter and fill in some of the gaps.

I would also like to thank Matt Welsh, who was the first to encourage me when I uttered the idea "someone" might write a Networking Guide for LINUX.

Extra praise goes to all the people who have contributed so much time and energy through their work behind the scenes, especially to Ari Lemke, maintainer of the mailing list at `niskula`, and of the FTP archive at `funet`; to Ted T'so, who is maintaining the FTP archive at `tsx-11`; and the numerous FTP administrators at `sunsite`.

Although they are not directly involved in the writing of this guide, I would like to thank all those who started the whole LINUX business, and have kept the ball rolling for so long: Linus Torvalds, Lars Wirzenius, Theodore T'so, H.J. Lu, Ross Biro, Peter MacDonald, Fred van Kempen, Donald Becker,...

Of course, there are many more people involved in the shaping of LINUX than this short list can name; you know who you are!

Chapter 1

Introduction to Networking

1.1 History

The idea of networking is probably as old as telecommunications itself. Consider people living in the stone age, where drums may have been used to transmit messages between individuals. Suppose caveman A wants to invite caveman B for hurling rocks at each other, but they live too far apart for B to hear A banging his drum. So what are A's options? He could 1) walk over to B's place,[1] 2) get a bigger drum, or 3) ask C, who lives halfway between them, to forward the message. The last is called networking.

Of course, we have come a long way from the primitive pursuits and devices of our forebears. Nowadays, we have computers talk to each other over vast assemblages of wires, satellites, and the like, to make an appointment for saturday's soccer match.[2] However, in the following, we will deal with the means and ways by which this is accomplished, and leave out the wires, as well as the soccer part.

We will encounter two types of networks in this guide, those based on UUCP, and those based on TCP/IP. These are protocol suites and software packages that supply means to transport data between two computers. In a networking environment, anything that is able to react reasonably to communication requests from the network is called a *host*. This is very often a computer, but need not be; one can also think of X terminals or printers as hosts. Small agglomerations of hosts are also called *sites*.[3]

[1] This option is in fact so rarely chosen, that any occurence thereof inevitably leaves traces in history books, like, for example, the Marathon run.

[2] The original spirit of which (see above) still shows on some occasions in Europe.

[3] In the UUCP world, most sites consist of one host, but it might also be a LAN with one computer handling the UUCP link to the outside world. Such sites are most often managed in a way to appear as a monolithic system to the outside world.

UUCP started out as a package of programs to transfer files over serial lines, schedule those transfers, and initiate execution of programs on remote sites. It has undergone major changes since its first implementation in the late seventies, but is still rather spartan about the service it offers. UUCP-based networks are of the store-and-forward variety, which means that the physical links between forwarding hosts are activated at certain intervals only (usually several times a day), so that any data has to be stored temporarily. The first of these networks evolved soon after the initial release, with over 80 Unix-developing sites already connected in mid-1978. They were running email as an application, as well as remote printing. However, the system's central use was in distributing new software and bugfixes.[4,5] One year later, in 1979, after the release of UUCP with the new Unix V7, three graduate students had the idea of a general information exchange within the Unix community. They put together some scripts, which became the first netnews system. In 1980, this network connected `duke`, `unc`, and `phs`, at two Universities in North Carolina. Out of this, Usenet finally grew, which nowadays stretches across several continents, comprising networks of all sorts.

TCP/IP traces its descent to a research project funded by the DARPA (Defense Advanced Research Projects Agency) in 1969. This was an experimental network, the ARPANET, which was converted into an operational one in 1975, after it had proven to be a success. Its main difference from UUCP-based networks is that it is not a store-and-forward network, but is based on packet switching. This means that permanent or semi-permanent circuits are used to connect various sites, with data not being transferred in files, but in smaller chunks (packets) which are forwarded instantaneously. This type of network is called a *packet-switched* network.

In 1983, the new protocol suite TCP/IP was adopted as a new standard, and all hosts on the network were required to use it. When ARPANET finally grew into the Internet (with ARPANET itself passing out of existence in 1990), the use of TCP/IP had spread to networks beyond the Internet itself. Most notable are local area networks used in Un⋆x networks.

The networking software included in the various distributions of Linux comes from many places. Many authors have contributed software packages to the net community, often maintaining them for years and investing a lot of their work. Before you curse them because something doesn't work the way you expect it to, imagine where you would be without their effort.

Where single authors have written an application program, they are credited, sometimes

[4] Not that the times had changed that much...

[5] This is detailed in a report by D. A. Novitz and Mike E. Lesk, named "A simple dial-up network in a Unix™ environment", published in August 1978. It is available at some FTP servers.

giving their addresses when they still maintain the particular package. Other software, such as the majority of TCP/IP clients and daemons, comes from the BSD Networking Release 2. Finally, we credit all those people who relentlessly keep hacking the LINUX kernel, especially Linus Torvalds, Ross Biro (who wrote much of the original kernel TCP/IP code), Fred N. van Kempen (who has rewritten much of the network code and taken over further development), and Theodore Ts'o (who maintains the serial drivers). There are many, many more of them. We thank you!

1.2 Two Examples

To have realistic material for demonstration at hand during the following, we introduce two examples:

Consider a University, complete with many, many departments, wars over funding, etc. We call it Groucho Marx University (GMU), situated somewhere in Netland. Most departments run their own LANs, while some share one, and others run several of them. They are all interconnected, and are hooked to the Internet through a single high-speed link.

The other example is more like what you may encounter when setting up your LINUX box: Let's assume we have a little company that's brewing, say, virtual beer.[6] For this purpose, they have some PCs in their offices, all running a bright and shiny LINUX 1.0. They are connected by an Ethernet. Name those machines `vlager`, `vstout`, and `vale` (`v` standing for "virtual", of course). One day they acquire the virtual winery which is located one floor above. The winery people have their own network, with machines named `vbeaujolais`, `vbardolino` and `vchianti`. So they decide to join their respective networks, and connect `vlager` to the winery's Ethernet. We will follow the brewers through their plight in setting up an operational local network.

Someday they decide to get on the net to get into closer touch with their customers, so they look for a UUCP site that might feed them mail and news. They find one, its name being `moria.orcnet.org`. Not a pleasant bunch of neighbors, altogether, but you can't help it, can you?

Now, our virtual brewers try to pick a name for their site. Their first idea is to call themselves `long-live-linus`, but that's too long a name for UUCP transports, so they settle for `linus`. In a later chapter we will peek over their shoulders while configuring UUCP, email, and netnews.

"Now, why two examples?" you might ask. Well, I thought I had to give one example of a site with Internet access, and one of a site with a UUCP link. I hope this isn't getting

[6] Hasn't that joke been beaten to death yet? Well, it hasn't `:-)`

too confusing. I will use the first example mainly to explain TCP/IP networking features in this chapter; later chapters dwelling on LINUX configuration will only refer to the virtual brewers. However, consider the large-scale examples for GMU with a sense of caution: I am in no way an Internet "old hand"; my knowledge comes from books and RFCs only.

1.3 The OSI Model

One paradigm that has proven to be most useful is to view networking as consisting of a number of tasks, organized in layers. They are thought to be built on top of each other, with the hardware being the lowest, and the application program being the highest. Each layer has a well defined interface through which it offers one or more services to the layer above it; it executes requests by using the interface offered to it by the layer immediately below it. The functions offered by an interface are also called *service access points*. Often, layers are split into sublayers to offer a clearer view of their functionality and requirements.

One description of the network software's layering is given by a model drawn up by the *International Standards Organization*, ISO for short, and is called the *Open Sytems Interface (OSI) Reference Model*. It consists of seven layers, which will be explained below as far as they apply to the networks we discuss. Not all of the OSI layers are present in all networks, as we will see. It is not very enlightning to apply the OSI model to UUCP-based networks, since most of its features are absent. In this section, we will therefore refer to UUCP networks only shortly.

At each level, the software performing the task is thought of as exchanging data with the corresponding level on the destination machine. The drivers are therefore referred to as *peer processes*. Peer processes are usually free to exchange data by whatever means they choose, as long as they agree on it. For example, they may split up the data into chunks, or choose from different transport protocols and routes. Now, the service two peer processes provide to the layer above them can be distinguished by whether it is reliable, and whether a connection is thought to exist or not.

Reliable service requires thorough checks on the integrity of thee data transmitted[7] and acknowledgements sent back to the sender, containing positive or negative confirmation on the state of the data received.[8] The usefullness of reliable service is obvious. Unreliable service, on the other hand, might be desirable if the network user performs his own error checking, or simply because it is only important a significant portion of the data comes through.

[7] You all know those fizzles and crackles on long distance calls. You can image what they do to an electric pulse less than a millisecond wide.

[8] Well, it's actually not that simple. Ask Prof. Murphy.

The second criterion is whether actual connections are desired, or connectionless service will suffice. In real life, telephone conversations versus letters sent by postal mail give a pretty good example of the difference between the two types of service. In the networking world, UUCP definitely is connectionless, altough not necessarily unreliable (don't laugh!). The other kind of service is, for example, a dialup session at your favorite BBS — where the "application" connection coincides with the physical conection — or an FTP session to an overseas server — providing for the illusion of a direct connection. The latter is therefore also called a *virtual connection*.

Lower layers may well provide unreliable service, with a higher layer adding error checking and thus providing a reliable service. Equally, a connection-oriented service may be built upon a connectionless service by some layer making sure that any data sent arrives at the destination host and is delivered to the upper layer in the order sent.

To perform its duty, peer processes may need to exchange administrative information related to the data being passed, for example checksums, acknowledgements, or requests to build up or release a connection. Therefore, each layer generally wraps up the data it is handed from above, adding any information it wants to pass along to its peer. Generally, this is done by adding a *header* (and possibly a trailer). This may be as little as a few bits to separate packets from each other, and as much as a few dozen bytes. This combined data is then passed to the software of the layer below, which processes the data according to the service requested but is completely ignorant of the packet's contents. It may then add its own header, and so on. When the packet arrives at the remote host, each layer in turn unwraps the data it is handed from below, processing it according to the information contained in the header, and passes it on to the layer above.

1.3.1 The Phyiscal Layer

The basic layer is the **physical layer**, which deals with the physical aspects of the equipment used, like wiring, timing logic, etc. We leave that to the people in the soldering iron department.

The most widely used equipment is what is commonly known as *Ethernet*. It consists ofa single coaxial cable with hosts being attached to it through connectors, taps or transceivers. Ethernets come in two flavors, called *thick* and *thin*, respectively. Most people prefer thin over thick Ethernet, because it is much cheaper: PC cards come for as little as US$ 80. Together with a net transfer rate of 10 Megabit per second this accounts for much of its popularity.

One of its drawbacks is that the cable length is usually limited to 200 meters, which precludes any use of Ethernet technology other than for LANs. Several Ethernet segments may be linked to each other using repeaters, bridges or routers. Repeaters simply copy the signals between two or more segments, so that all segments together will act as if it was one Ethernet. Due to timing requirements, there may not be more than four repeaters between any two hosts on the network. Bridges and routers are more sophisticated. They analyze incoming data and only forward it when the recipient host is not on the local Ethernet. They will be covered in section 1.3.6.

1.3.2 The Data Link Layer

The next layer is the **data link layer** which utilizes the physical layer to exchange data with hosts it is physically connected to. It is there to hide any idiosyncrasies of the hardware and offer a uniform interface to the upper layers, so that the network software need not worry about different types of hardware. For each type of equipment used, a different driver has to be implemented. For example, there are kernel drivers for handling I/O with your Ethernet card, as well as a SLIP driver for sending network traffic over a serial line. The low-level protocol drivers UUCP uses to safely transfer data over telephone lines also belong here.

The protocols involved in using an Ethernet are described in standard no. 802.3 issued by the IEEE.[9] The standard allows for an arbitrary number of hosts on an Ethernet. To distinguish them, each host is assigned a unique 6-byte address, usually written as `aa:bb:cc:dd:ee:ff`. Data is transported in chunks (so-called *frames*) of up to 1500 bytes long, with a header and trailer being added that contain source and destination address along with a checksum.

To send a frame, a host has to wait until the Ethernet is idle (no other traffic) and then may start sending its frame. All stations will receive this and compare the destination address to their own address. All but the proper recipient will discard the frame, while the recipient will hand the frame's data to the upper layers.

If two stations try to send a frame at the same time, there is a *collision*. Both will stop transmitting and restart after a random time interval.

On serial lines a "de facto" standard is frequently used, which is SLIP, or *serial line IP*. A modification of this is known as CSLIP, or *compressed SLIP*, and performs compression of IP headers to make better use of the relatively low bandwidth provided by serial links.[10] An entirely different serial protocol is PPP, or *point-to-point protocol*. It has many more

[9] To be exact, IEEE 802.3 covers both the physical layer and the medium acess sublayer (MAC) specifications of an Ethernet. The medium access sublayer is the lower part of the data link layer. The upper part is governed by the Logical Link Control (LLC) protocol, which is the same for token bus (IEEE 802.4) and token ring (IEEE 802.5). LLC itself is described in IEEE 802.2.

[10] SLIP and CSLIP are described in RFC 1055 and RFC 1144, respectively.

features than SLIP, including a link negotiation phase. Its main advantage over SLIP, however, is that it isn't limited to IP datagrams, but was designed to allow for any type of datagrams to be transmitted. It is assumed that it will one day replace SLIP.[11] PPP is an example of a data link protocol that may be used for more than one type of hardware. Although its main use today is for serial lines, where are debates going on if it should one day become the standard for ISDN-links.

In TCP/IP networks, the data link layer is also responsible for resolving IP addresses to the hardware-dependent address. On Ethernets, for example, the *Address Resolution Protocol*, or ARP, is used to find the Ethernet address corresponding to an IP address.[12,13] ARP uses the broadcast feature of Ethernet to send a query with the unresolved IP address to all stations on the local Ethernet. The host recognizing the address as its own returns a message to the sender, containing its Ethernet address.

ARP, however, is not limited to Ethernets. It may be used over any hardware that supports broadcasting, for example AX.25 (used for transmitting IP packets over packet radios).

The results of ARP queries are stored in a kernel table from which the driver may look them up. Entries in this table are discarded after a certain time to keep this table from overflowing.

1.3.3 The Networking Layer

On top of the data link layer lies the **networking layer**. It is responsible for transporting data given to it to the destination host using the services provided by the data link layer.

In TCP/IP networking, this operation is generally performed using the *Internet Protocol*, abbreviated IP. The destination host is given to it as a number 32 bits wide, called the IP addresses. We will later see how they are made up (see section 2.1).

One of the networking layer's duties is interconnecting physically dissimilar networks (like Ethernets, token rings, etc.) into one apparently homogeneous network. This is called internetworking, and the resulting "meta-network" is called an internet. Note the subtle difference between *an* internet and *the* Internet here. The latter is the official name of one particular internet.

[11] The original description of PPP can be found in RFC 1134, however there are a number of ancillary RFCs.

[12] ARP uses the broadcast feature of Ethernet to probe for any host that recognizes the IP address as his. See RFC 826 for a specification of ARP.

[13] To allow booting a diskless client via the network, there's a related protocol for mapping Ethernet to IP addresses, named RARP (*reverse ARP*). This, however is not implemented in the data link layer, but is provided by an independent server.

To achieve the illusion of a single (inter-) network, the data link layer hides the diversity of the equipment used by defining an abstract "interface". This interface offers a set of operations which is the same for all types of hardware.

In TCP/IP networking, interfaces are identified by a name. For example, Ethernet interfaces in LINUX are called `eth0`, `eth1`, etc, and SLIP interfaces come as `sl0`, `sl1`, etc.

While the operations of the data link layer only involve hosts that may communicate with each other directly — e.g. all hosts on an Ethernet —, the network layer's task of delivering data to *any* destination requires cooperation between a possibly large number of hosts: when the destination cannot be reached directly, the sender has to rely on one or more other hosts to act as forwarders. The sequence of intermediate hosts the data has to travel is usually referred to as a route.

Most networks implement some algorithm in the network layer to construct these routes. In packet-switched networks, this algorithm often performs "dynamic" routing, which means that routes are adjusted whenever necessary. The opposite, static routing, is often used for mail routing in UUCP-based networks, where new routes are computed once every few days or weeks.

Another, equally important duty of the network layer is congestion control. When network traffic increases, some links tend to congest quickly. To relieve these, some of the sending hosts might be told to decrease the rate at which they emit data. Networks that use a dynamic routing scheme also have the option to adapt their routing decisions to the actual network load.

In TCP/IP networking, routing and congestion control are handled by IP, and a companion protocol, the *Internet Control Message Protocol* (ICMP). Routing information in the networking layer is usually static, but may be adapted dynamically from router daemons (e.g. `routed`). These daemons regularly exchange local routing information, and update the kernel's routing tables according to the information received.

The IP layer of remote hosts may also send an "ICMP Redirect Message" when it detects that a sending host has chosen a bad route. This enables the sending host to augment its routing table by the hint returned in the redirect message.

Another responsibility of ICMP is to return an indication to the sender in case an error occurred. For example, if a host receives an IP packet for an address that it is unable to reach, it returns an error message to the sender. The most widely known ICMP message, however, is the "Echo" message, which simply causes the destination host to acknowledge its receipt. This is used by the Un⋆x `ping` tool (see 3.8.1) to test the reachability of foreign hosts and compute the delay in getting there.

The networking layer in UUCP is not as obvious as with IP. UUCP does not perform any routing of its own, but expects to be handed the destination's address in form of a route. UUCP's network layer only pushes around files between hosts, establishing point-point-connections to transfer them at certain times. Generation of the route has to be performed by higher levels.

The entirety of the lower three layers is also often referred to as the *subnet*, because this is where actual networking — in the terms of handing information from host to host — takes place. The layers above only deal with the destination host, the service desired on the remote host, and possibly a route to it.

The service IP offers to the upper layers is connectionless and unreliable. One reason for this is that experience has shown that the subnet is inherently unreliable, even in the presence of a reliable networking protocol: Gateways may fail, or packets might be routed to the wrong destination, or take a long route only to pop up at the destination much later, and so on. But then, connection-oriented service which is not reliable is not very useful. Another reason is that one also wants to utilize the routing services provided by the IP level for transports that don't need a connection to be established, so that this is best added in a higher layer. This is done in the next layer, the so-called **transport layer**.

1.3.4 The Transport Layer

Its task is to provide the upper layers with access to a remote host's services. In TCP/IP networks, different types of transport may be provided. One is implemented using TCP, or *Transport Control Protocol*. It builds a connection-oriented reliable service on top of IP. There are many gory details to this which we will not discuss here.

Another protocol implemented nearly everywhere is UDP, or *User Datagram Protocol*, which offers an unreliable connectionless service. It takes packets of data and hands them to the IP layer for transport, caring little if they arrive at all, and even less if they do so in the order sent. However, if a host finds it cannot deliver a UDP packet properly, it returns an error indication, so that the sender has some notification of whether the datagram got through.[14]

Finally, there's a protocol that doesn't perform any actions by itself, but simply hands data on to the IP level. This is the RAW protocol.

Of course, networking requires a destination that the data should be delivered to once it has arrived on the remote host. For example, you wouldn't want your love letters to be

[14]This feature is used by a debugging tool, `traceroute`, to detect the route IP datagrams take to a given destination. It sets the datagram's ttl (time-to-live) value such that it is discarded by the host that is exactly n hops away from the sender, gradually increasing the ttl until it reaches the remote host. From the error messages it can deduce the sequence of hosts the datagrams pass.

passed to the news subsystem, or all of `comp.os.linux` to the printer spooler. Thus, the destination address needs to contain a description of the service to hand the message to.

On UUCP-based networks, this is solved by specifying a program to be invoked which will process your data. For example, to print this book on `hostA`, you would issue the following command at the prompt:

```
$ uux - hostA!lpr -Plaserjet < netguide.dvi
```

With TCP/IP networking, a more general scheme is used. Each service a site offers defines a so-called *service access point* through which it may be contacted. These service access points are assigned *port numbers*. Now, when a process wants to contact a service on a remote host, it passes the data to be transferred to the networking software, alongside with the remote host's address and the service's port.

One certainly wants the transport layer to support several concurrent connections to the same service, like several remotely logged-in users. To distinguish these, the client always has to specify a sender port number, which has been generated uniquely by the originating host. Thus, a connection is determined uniquely by a host/port pair for each of the two hosts involved.

In order for the client to know the proper port number to use for accessing a particular service, an agreement has to be reached between the administrators of both systems on the assignment of these numbers. For services that are only used within an organization, this can be done privately. However, service numbers widely used (such as for the network filesystem, `telnet`, electronic mail, etc), have to be administered centrally. This is done by the IETF (or *Internet Engineering Task Force*), which regularly releases a RFC titled "*Assigned Numbers*".[15] It describes, among many other things, the port numbers assigned to *well-known services*. There is also generally a file translating service names to numbers, it is called `/etc/services`. It is described below in 4.1.

However, this is not yet the entire truth. Consider a datagram arriving on a host. Who decides what transport protocol it is handed to? Of course, you may say, the transport layer might decide from the port numbers of sender and destination which protocol is involved. But, for some protocols, like RAW, there is no such thing as a connection established beforehand, so there is no information on the protocol involved. Therefore, there is an IP header field containing a protocol identification which allows it to dispatch the datagram to the appropriate driver in the transport layer. This identification is a 16 bit number. The correspondence between protocols and numbers is, of course, also a matter of standardization, and is also laid down in the "Assigned Numbers" RFC. On your host, this information is kept in the file `/etc/protocols`.

[15] Its most recent release is RFC 1060.

All layers up to the transport are generally located in the kernel, and so they are in LINUX. The user interface most common in the Un⋆x world is the *Berkeley Socket Library*. Its name derives from a popular analogy for service access points that views ports as sockets, and connecting to a port as plugging in. It provides calls to specify a remote host, a transport protocol, and a service (`bind(2)`) to which to connect or to listen to (using `connect(2)`, `listen(2)` and `accept(2)`). The socket library is however somewhat more general, in that it provides not only a class of network-based sockets (the `AF_INET` sockets), but also another class that handles connections local to a machine (the `AF_UNIX` class). Some versions can also handle other classes as well, for instance those that have a class of sockets for the XNS (Xerox Networking System(?)) protocol, or for X.25.

1.3.5 The Upper Layers

The **session layer** usually deals with the way two communicating processes exchange status information and service requests. For example, a FTP application somehow has to tell the FTP service on the remote host which file to retrieve, or which directory to change to.

In TCP/IP networking, no standard session layer is provided. Every application has to define and implement it separately. However, a common practice for applications using TCP connections is to define a set of messages transferred as lines of ASCII text, terminated by carriage return and linefeed. A protocol defines the commands that may be issued by the client, and a set of responses to be returned by the server. Responses most often consist of a three-digit octal number that may be followed by a human-readable interpretation of the message. Transmission of text is usually terminated by a line containing a single dot. An example is SMTP, the *Simple Mail Transfer Protocol*.

Figure 1.1 shows a sample SMTP session. Lines beginning with a three-digit number are messages from the server, all other lines are commands issued by the client.

The layer immediately above is the **presentation layer**, which provides for a system-independent representation of data to be transferred. Again, in TCP/IP networks, every application has to define this for itself. For example, FTP defines a representation to transfer text files between any two machines, and TELNET has a *virtual terminal* to represent terminal control sequences in a system-independent fashion.

Both of these layers are absent in UUCP networks.

The upmost layer is the **application layer**. This is the user program performing a particular task, like `ftp` or the mail transport software.

```
220 monad.swb.de Smail 3.1.28.1 #6 ready at Tue, 29 Jun 93 10:45 MET DST
HELO fubar.swb.de
250 monad.swb.de Hello fubar.swb.de
MAIL FROM:okir@fubar.swb.de
250 <okir@fubar.swb.de> ... Sender Okay
RCPT TO:torvalds@niksula.hut.fi
250 <torvalds@niksula.hut.fi> ... Recipient Okay
DATA
354 Enter mail, end with "." on a line by itself
  --- (sending mail message) ---
.
250 Mail accepted
quit
221 monad.swb.de closing connection
```

Figure 1.1: A sample SMTP session.

1.3.6 Internetworking

Above we stated that one of the networking layer's duties was to link physically dissimilar networks to form an internet. But, of course, it would be rather limiting to allow linking of networks only at this level of the protocol stack. For example, assume a LAN that consists of several Ethernets. Linking them by just inserting a long piece of cable between the segments does not work because of the electrical specifications. On the other hand, inserting a gateway at each junction is forbidding because of the delay introduced. Imagine some data of yours had to pass a dozen or so gateways just to travel a few hundred yards distance, being constantly wrapped and unwrapped by Ethernet drivers!

A simple solution to this problem is the use of *repeaters*. These are electrical devices used to connect two Ethernets, who do not perform any analysis of data passing through it, but only copy bits to and fro. Now traffic may float freely between all Ethernets, and there are no limitations.

Sadly, this is not so. For one, timing specifications require that a signal sent by one station on an Ethernet reaches any other station within a certain amount of time. This puts an upper boundary of four on the number of repeaters two stations may be apart.

Another drawback of repeaters is that instead of at most a few dozen machines on one local segment, all machines from the linked segments compete for the right to transmit a frame. When segments A and B are linked by a repeater, all stations on segment A have to shut up when a station on segment B talks.

Another solution is the use of *bridges*. These are still Ethernet-specific devices, but instead of simple bit-shuffling, a bridge receives entire frames, analyzes the recipient, and only copies it to one of the adjacent segments when the recipient is known to be on that segment (or known to be reachable through this segment). A bridge does not suffer from the limitations of a repeater, because the single Ethernets remain separate entities, and frames only get copied to another segment if really needed. The drawback of bridges is that they need to be configured properly.

Now, let's have a look at where these techniques fit in our protocol stack. A gateway, as we said, is located in the networking layer. Repeaters are on the other end of the scale, they operate entirely at the physical layer, while bridges are in the data link layer.

Of course, one can also think of internetworking schemes that reside in still higher levels. For example, one could think of a program that accepts TCP connections for remote hosts, multiplexes them onto a serial line, with a similar program demultiplexing the whole on another machine and opening a separate TCP connection to the target host.[16] Such a program would definitely operate at the transport layer.

[16] In fact, there is a program for Linux that does roughly this. This is the `term(1)` program written by Micheal O'Reilly, `oreillym@tartarus.uwa.edu.au`.

Chapter 2

Issues of TCP/IP Networking

2.1 IP Addresses

As mentioned in the previous chapter, the addresses understood by the IP network layer are 32-bit numbers. Every machine must be assigned a number unique to the networking environment. If you are running a local network that does not have TCP/IP traffic with other networks, you may assign these numbers according to your personal preferences. However, for sites on the Internet, numbers are assigned by a central authority, the Network Information Center, or NIC.[1]

For easier reading, IP addresses are split up into four 8 bit numbers called *octets*.[2] For example, `uunet.uu.net` has an IP address of `0xC0306002`, which is written as `192.48.96.2`. This format is often referred to as the *dotted quad notation*.

Another reason for this notation is that IP addresses are split into a *network* number, which is contained in the leading octets, and a *host* number, being the remainder. Depending on the size of the network, the host part may need to be smaller or larger. To accomodate different needs, there are several classes of networks, defining different splits of IP addresses.

A
: Class A networks have network numbers of `1.0.0.0` through `127.0.0.0`. This provides for a 24 bit host part, allowing roughly 1.6 million hosts.

B
: Class B networks have network numbers `128.0.0.0` through `191.255.0.0`. This allows for 16065 nets with 65534 hosts each.

[1]To apply for an IP address for your network, send a mail to `hostmaster@nic.ddn.mil`. It is recommended to obtain one even if you don't intend to get on the Internet yet; you may want to in the future.

[2]They aren't called bytes because there are machines on the Internet with byte sizes other than eight bits.

C Class C networks have network numbers `192.0.0.0` through `223.255.255.0`. This allows for nearly 2 million networks with up to 254 hosts.

D Addresses falling into the range `224` through `254` are reserved for future use and don't specifiy any network.

This shows that `uunet`'s address refers to host `2` on the class C network `192.48.96.0`.

When applying to the NIC for IP addresses, you do not do so for each separate host you plan to use. Instead, you are assigned a network number, and are allowed to assign all valid IP addresses within this range to hosts on your network according to your preferences. The network class you are assigned depends on the size of the network you are running.

You may have noticed that in the above list only 254 possible values were allowed for octets in the host part. This is because host numbers with octets all `0` or all `255` are reserved for special purpose. An address with all host part octets zero refers to the network, and one with all host part octets `255` is called a broadcast address. This refers to all hosts on the specified network simultaneously. Thus, `192.48.96.255` is not a valid host address, but refers to all hosts on network `192.48.96.0`.

There are also two network addresses that are reserved, being `0.0.0.0` and `127.0.0.0`. The first is called the *default route*, the latter the *loopback address*. The default route has something to do with the way IP routes datagrams.

The loopback network is a class A network reserved for testing and debugging purposes. Usually, address `127.0.0.1` will be assigned to a special interface of your host. This is called the *loopback interface* and acts like a closed circuit. Any packet handed to it from TCP or UDP will be returned to them as if they had just arrived from some network. This allows you to develop and test networking software without ever using a "real" network.

2.2 IP Routing

◇ When you write a letter to someone, you will usually put a complete address on the envelope, specifying the country, state, zip code, etc. After putting it into the letter box, the postal services will deliver it to its destination: they will send it to the country indicated, whose national postal services will dispatch it to the proper state and region, etc. The advantage of this hierarchical scheme is rather obvious: Wherever you send the letter from, the local postmaster will know roughly the direction to forward the letter to.

IP networks are structured in a similar way. The whole Internet consists of a number of proper networks, called *autonomous systems*. Each such system performs any routing between its member networks internally, so that the task of delivering a datagram is reduced to finding a path to the destination host's system. This means, as soon as the datagram is handed to *any* host that is on that particular network, further processing is done exclusively by the network itself.

This structure is reflected by splitting IP addresses into a host and network part, as explained above. By default, the destination network is derived from the network part of the IP address. Thus, hosts with identical IP network numbers should be found on the same network, and vice versa.[3]

Now, of course, one would prefer a similar scheme *inside* the network, too, since it may consist of a collection of hundreds of smaller networks itself, with the smallest units being physical networks like Ethernets. Therefore, IP allows you to subdivide an IP network into several *subnets*.

Note that this is different from what "subnet" means in the OSI model: The OSI term refers to the entirety of equipment and software that is responsible for getting data from one host to another:[4] The transport layer hands a packet to the subnet, which does lots of "don't ask" things, and finally it pops up at the destination host — if you're lucky, that is.

However, in IP-speak, a subnet is something very concrete. In fact, it's just one of those things the OSI term wants to hide: it is a subordinate part of an IP network that takes over responsibility for delivering datagrams to a certain range of IP addresses. Like an autonomous system, a subnet is identified by the network part of the IP addresses it represents. However, the network part is now extended to include some bits from the host part. The number of bits that are interpreted as the subnet number is given by the so-called *subnet mask*. This is a 32 bit number, too, which specifies the bit mask for the network part of the IP address.

A class B network, for example `149.76.0.0`, is identified by the first two octets of its IP address. Thus, its subnet mask is `255.255.0.0`. For delivery inside this network, it might be split up into 254 subnets, being `149.76.1.0` through `149.76.254.0`. They all share the same IP network number, while the third octet is used to distinguish between them. Thus they will use a subnet mask of `255.255.255.0`.[5]

It is worth noting that subnetting (as the technique of generating subnets is called) is only an *internal division* of the network. Subnets are generated by the network owner (or

[3] Autonomous systems are slightly more general, however. They may comprise more than one IP network.

[4] Probably the prefix "sub" was chosen to suggest that it denotes some low-level functionality.

[5] These subnets may now be subdivided again by further extending the network part of the IP address. Of course, different "sub-subnets" may use different subnet masks. However, hosts on the *same* subnet must use the same subnet mask for this subnet.

the administrators). Frequently, subnets are created to reflect existing boundaries, be they physical (between two Ethernets), administrative (between two departments), or geographical, and authority over these subnets is delegated to some contact person. However, this structure only affects the network's internal behavior, and is completely invisible to the outside world.

At the lowest level, subnet boundaries usually coincide with hardware boundaries. The viewpoint of a host on a given subnet is a very limited one: The only hosts it may talk to directly are those of the subnet it is on. All other hosts belong to the Great Blue Yonder which is accessed through so-called *gateways*. A gateway is a host that is connected to two or more subnets or networks simultaneously. This may be a bridge (simply copying datagrams from one Ethernet to another), a router (a special device that runs some sort of IP routing software), or a computer. Therefore, hosts which are connected to more than one subnet need one IP address per subnet. Consequently, IP addresses and subnet masks are not considered host parameters, but interface parameters.

Frequently, traffic between two subnets has to traverse more than one gateway. The sequence of hosts to travel is called a *route*. The scheme by which datagrams are directed to the proper host is called *routing*. Consider a datagram being handed to the IP layer of a host on some Ethernet. It will check if the network portion of the IP address is the same has the Ethernet's subnet number (as determined by the subnet mask), and if so, will deliver it directly. Otherwise, it is passed to a gateway which is supposed to deliver the datagram to the destination host. The gateway, having received a datagram, basically faces the same options as the sender, namely local delivery to one of the adjacent Ethernets, or forwarding to another gateway — but it is presumably one step closer to the destination. Thus, in the presence of several gateways, the cunning in IP routing lies in guessing the "right" one to hand the datagram to. The "right" one — apart from any quality critera one might wish to consider — is a gateway that is able to forward the datagram to the destination network.

Thus, the routing information used by IP is basically an (incomplete) table linking networks to gateways to reach them through. A catch-all entry (the "default gateway") must generally be supplied, too; this is the gateway associated with network `0.0.0.0`.[6]

Well, before you start collapsing or begin to wonder why you thought networking seemed such a neat thing in the first place, we will provide you with an example. In figure 2.1, a part of the network topology at Groucho Marx University (GMU) is shown.

GMU has a class B network address of `149.76.0.0`. The administrators at the Groucho Computing Centre (GCC) have decided to use the third octet as a subnet number to address

[6]Routing data through a default gateway is inherently inefficient. To straighten out bad default routes, a gateway may notify the sending gateway of a better alternative by sending it an "ICMP redirect" message. ICMP is the Internet Control Message Protocol. It is described in RFC 792.

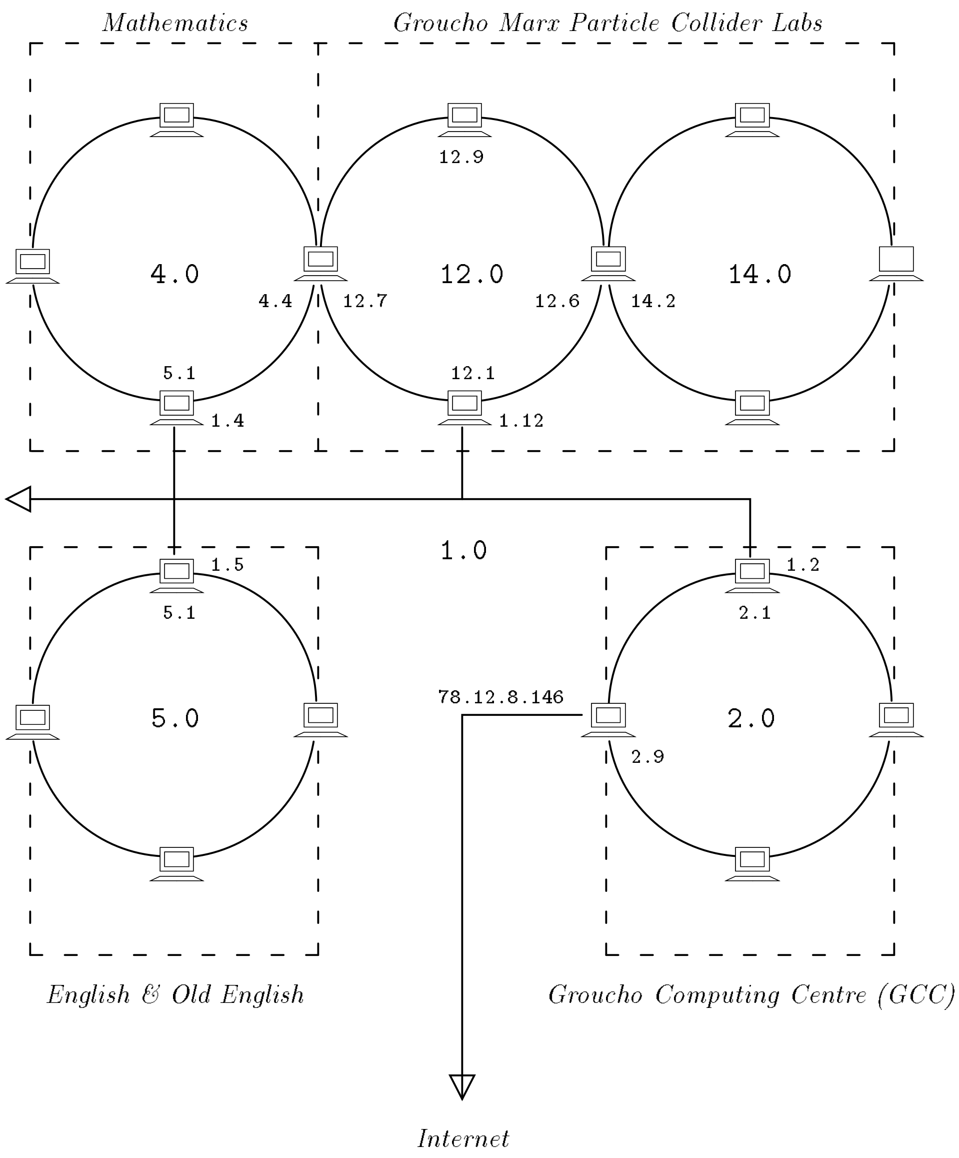

Figure 2.1: A part of the net topology at Groucho Marx Univ.

LANs. Thus, the subnet mask is `255.255.255.0`. The subnet numbers are shown at the center of the circles, which represent LANs. For example, the Mathematics Department runs a single LAN, which has been assigned `149.76.4.0`, while the Collider Lab has two LANs, having network addresses of `149.76.12.0` and `149.76.14.0`, respectively. Most LANs are connected to a Fiber Optics cable that runs across the campus. This is a network by its own right, and has been given number `149.76.1.0`.

In the above graphic, hosts which are on two networks at the same time are shown with both addresses. For example, the machine that is the gateway between `12.0` and the Fiber Optics cable has address `149.76.12.1` when talking to the Collider Lab LAN, and `149.76.1.12` when talking to network `149.76.1.0`.

An example routing table for `149.76.12.1` might look like this:

Network	Gateway
149.76.2.0	149.76.1.2
149.76.4.0	149.76.12.7
149.76.5.0	149.76.1.5
149.76.14.0	149.76.12.6
...	
0.0.0.0.0	149.76.1.2

Routing tables may be built by various means. For LANs, it is usually most efficient to construct them by hand and feed them to IP using the `route` command at boot time (see section 3.6). For larger internets (see section refintro.osi.subnet), they are built and adjusted at run-time by *routing daemons*; these run on central hosts of the network and exchange routing information to compute "optimal" routes between the member networks.

The intended scope of their arbitration determines the protocol employed. For routing inside autonomous systems (such as Groucho Marx campus), the *internal routing protocols* are used. The most prominent one is RIP, or Routing Information Protocol, which is, for example, supported by the BSD `routed` daemon. For routing between autonomous systems, *external routing protocols* like EGP (External Gateway Protocol), or BGP (Border Gateway Protocol) have to be used; these (as well as RIP) have been implemented in the University of Cornell's `gated`. We will not describe them in this document.[7]

[7] For further information, please refer to RFC 1058 (RIP), RFC 827, RFC 904 (EGP), RFC 1163, and RFC 1266 (BGP).

2.3 The Domain Name System

◇ As described above, addressing in TCP/IP networking revolves around 32 bit numbers. However, you will have a hard time remembering more than a few of these. Therefore, "ordinary" names are generally used to name hosts on a network. It is then the application's duty to find the IP address corresponding to this name. This process is called *address resolution*.

Now, on a small network like an Ethernet, or even a cluster of them, it is not very difficult to maintain tables mapping host names to addresses. On Un⋆x systems, this information is usually kept in a file named `/etc/hosts`. This is also the way address resolution was initially handled on the Internet. However, since the beginning of the eighties, the number of sites and computer networks has exploded, so that it is virtually impossible to keep routing information up to date.

This is why a new address resolution scheme has been adopted, the Domain Name System, or DNS for short. Host addresses have been organized in a hierarchy of domains. A domain is a collection of sites that are related in some sense — be it because they form a proper network (e.g. all machines on a campus, or all hosts on BITNET), because they all belong to a certain organization (like the U.S. government), or because they're simply geographically close. Such a domain may itself be part of a larger domain, which is in turn part of a still larger domain. This relationship is described as being a *subdomain* of the larger domain. To produce a unique address for each machine,[8] names inside a domain must be unique. Assume a machine in the mathematics department of Groucho Marx College is named `gauss`. The Ethernet run by the maths department will then be called something like `maths`, and the whole campus network might be named `groucho`. Thus, the machine's address would be `gauss.maths.groucho.edu`. Such a name determines the site's place in the domain hierarchy and is called its *fully qualified domain name(FQDN)*.

There is also one pseudo-domain which encompasses all hostnames and domains. It is called the *root domain* and denoted by a single dot. Therefore, fully qualified domain names are sometimes written with a trailing dot to indicate that they are relative to the root domain.

There are a number of major domains, which are split up into subdomains, which may again be split up, etc. The US top-level domains are

`.edu` Educational institutions like universities, etc.

`.com` Commercial organizations, companies.

[8]In fact, what's needed is only a scheme that maps a name to a host. It is perfectly legal for a machine to have several names (i.e. *aliases*).

`.org` Non-commercial organizations. Often private UUCP networks are in this domain.

`.mil` US military institutions.

`.gov` US Government agencies.

`.net` Gateways and other administrative host on a network.

`.uucp` Officially, all site names formerly used as UUCP names without domain, have been moved to this domain.

Outside the US, each country generally defines its own top domain, like `.uk` for the United Kingdom, `.fr` for France, `.de` for Germany, or `.au` for Australia etc.

Now the idea behind DNS is that address data may be retrieved from so-called *name servers* that do not hold the entire database, but only a fraction of the overall data in their cache. In the beginning, a name server doesn't have much information than on the local domain. The storage in which this data is kept is called the *cache*. This information may be retrieved by anyone by simply sending a query to the name server, which returns the desired information.

Usually, there will be only one name server (or at most a few) for any given domain. Clients within this domain will have to query one of them to obtain addresses for a given hostname. So, how can a client obtain the IP address from a host in a different domain?

Assume the client queries the local name server for `tsx-11.mit.edu`, who fails to find it in its cache. However, instead of returning an error, it checks if it finds the address of the `mit.edu` domain name server in its cache. If this still fails, it looks for `edu`, and finally for a name server for the root domain. Only if this fails, too, will it return an error. However, if any name server address is found, a query for `tsx-11.mit.edu` will be sent to this server, which will most probably produce the corresponding address. The local name server will return the resulting address to the client, and store it in its cache for further use.

Now, DNS wouldn't buy us much if the root name server still had to know all addresses. Therefore, the name space has been split up into *zones of authority* that maintain their part of the name space independently. A zone of authority is roughly the same as a domain for which an organization assumes responsibility. It has a topmost node, which is the (domain) name, and comprises all (domain and host) names below that node unto the beginning of a new zone. For these, it has to provide name service mapping hostnames in this zone to IP addresses. Responsibility may be further delegated to subdomains, by creating new zones at these domains, which then provide independent name service for themselves.

Now, when in the above example, the local name server queries the root name server, this will look if it finds any authoritative name server for the `edu` zone of authority. There sure are name servers for `edu`, one of them being `a.isi.edu`. The root name server will then return its IP address to the local name server, along with the indication that this is a name server's address. The local name server will then continue and query `a.isi.edu` for `tsx-11.mit.edu`, which will probably return a pointer to one of MIT's name servers, etc.

This looks like a lot of traffic being generated for looking up one simple address, but is really only miniscule compared to the amounts of data that would have to be transferred with the old method.

We have seen in the example above that DNS does not only deal with IP addresses of hosts, but also exchanges information on name servers. There are in fact a number of types an entry in the DNS database may have.

Entries in the DNS database are called *resource records*, They have a certain type associated with them, describing the sort of data contained, and a class, describing the sort of network address associated. The latter accomodates the needs of different addressing schemes, like IP addresses (the `IN` class), or addresses of Hesiod networks[9] (the `HS` class), and many more. The prototypical resource record type is the `A` record which associates an IP address with a fully qualified domain name. Another type of record, linking aliases for a domain or host to its *canonical name*, is the `CNAME` resource record.

Now, each zone maintains at least two *master* name servers which are authoritative for this particular zone. Authoritative information is stored in the `SOA` resource record (which stands for "Start of Authority"). It is loaded into the master name server at boot time, together with a list of `A` records describing the names and IP addresses of all hosts in the zone.

Any query for an address in that zone will eventually be passed to one of them (unless, of course, an intermediate server has cached the information). However, as we read above, the root of a zone is part of the zone itself, whence the dilemma arises how to query a name server whose address is the subject of the query itself. Therefore, authoritative data has to be accompanied by information on the master name servers of the zones directly below. It is kept in `NS` and `A` resource records, which give the nameserver's hostname and IP address for the subordinate zone. Since this information is what holds the name space together, they are also called *glue records*. Since nameservers belong to their respective zone, these are never authoritative.

The master name servers for a zone are kept in sync by making one the *primary* name server, and the others *secondary*. The primary server loads its zone information from so-called *zone files* at startup, while secondary servers transfer this information from the pri-

[9] Anybody explain to me what they are?

mary server. To keep themselves fairly well synchronized, secondary servers expire this data after a certain time, which means they query the primary server if the zone's authoritative information has changed, and if so, reload it.

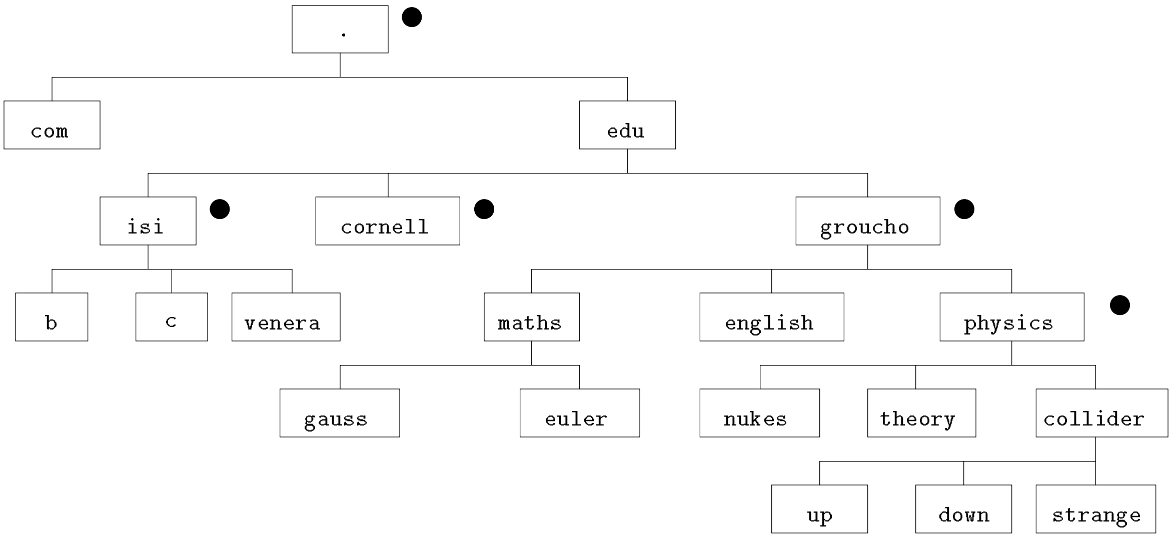

Figure 2.2: A part of the domain name space

Maybe an example to visualize this might be in order here. Consider the section of the name space given in Fig. 2.2. Those domains marked by a fat dot define the root of a zone. You can see that the physics department of our fictitious Groucho Marx University have been given authority over their network. Thus, the zone of authority starting at the node labelled `groucho.edu.` comprises all of the maths and English department, while the physics department is a *separate* zone.

Assume that the master name server for the physics department is located at `149.76.12.1`. Then the configuration file loaded into the nameserver might contain data as shown in figure 2.3, while the configuration file for the master name server for GMU will have a line in it as shown in figure 2.4.

In the two figures, you may note that some names end with a dot, while others don't. That is because the latter are only given relative to the domain `groucho.edu`, and the others are given absolute. DNS software distinguishes this by a trailing dot.

Finally, there's one special domain that is used to find hostnames corresponding to IP addresses and networks. This is called *reverse mapping*, and may be used by some applications to find out the caller's hostname. For example, some FTP servers only allow you to log in when they are able to find your host name. The domain used for

```
;
; Authoritative Information on physics.groucho.edu
@                 IN       SOA          {
                      up.collider.physics.groucho.edu.
                      hostmaster.up.collider.physics.groucho.edu.
                      1034            ; serial no
                      360000          ; refresh
                      3600            ; retry
                      3600000         ; expire
                      3600            ; default ttl
                    }
;
; Our secondary name server
                  IN     NS           gauss.maths.groucho.edu.
;
; Hosts in the physics Dept.
;
; Collider Lab
up.collider       IN     A            149.76.12.1
down.collider     IN     A            149.76.12.4
strange.collider IN      A            149.76.12.6
boson.collider    IN     A            149.76.14.1
muon.collider     IN     A            149.76.14.7
...
;
; Nuclear Physics Dept.
bohr.nukes        IN     A            149.76.8.3
hahn.nukes        IN     A            149.76.8.77
...
```

Figure 2.3: An excerpt from the `named.hosts` file for the Physics Department.

```
@                IN       SOA          {
                    vax12.gcc.groucho.edu.
                    hostmaster.vax12.gcc.groucho.edu.
                    233             ; serial no
                    360000          ; refresh
                    3600            ; retry
                    3600000         ; expire
                    3600            ; default ttl
                  }
....
;
; The zone of authority for the physics dept.
physics          IN       NS       up.collider.physics.groucho.edu.
                 IN       NS       gauss.maths.groucho.edu.
;
; Glue record
up.collider.physics IN    A        149.76.12.1
...
```

Figure 2.4: An excerpt from the `named.hosts` file for GMU.

this is `in-addr.arpa`. To find out the name corresponding to, say `149.76.12.20`, you may query DNS for `20.12.76.149.in-addr.arpa`. Equivalently, to find the name for network `149.76.0.0`, query DNS for `76.149.in-addr.arpa`. Resource records linking `in-addr.arpa` names to hostnames may be given using the `PTR` record type.

Now, it would be sort of unwieldy for user programs to query the name server directly, putting together UDP datagrams manually, and waiting for responses to come in. Therefore, a library is used, the so-called *resolver*. Its configuration is explained below in section 3.9.[10]

[10]For more information on DNS and its implementation, refer to RFC 1034 and RFC 1035, as well as RFC 1033, which describes the `named` configuration file format.

Chapter 3

Configuring TCP/IP Networking

3.1 General Remarks

Due to the nature of TCP/IP development in LINUX, there has not been a single standard for networking applications and configuration in the past. For example, until recently, all configuration files as well as most daemons resided under `/usr/etc/inet`. Many moved this to `/etc/inet` to have these files on their root partition, making `/usr/etc/inet` a symbolic link to `/etc/inet`. In release 4.1 of the C library, H.J. Lu changed the location of all network configuration files to `/etc`, which is the common place for such things on other Un$\star$x systems. Fred van Kempen, in his distribution of networking binaries, reintroduces `/usr/etc` and creates `/etc/net`. Until things settle a bit more, I have found it convenient to move everything to `/etc`, and make the other directories point to `/etc`:

```
# mv /usr/etc/* /etc
# mv /etc/inet/* /etc
# rmdir /usr/etc /etc
# ln -s /etc /usr/etc
# ln -s /etc /etc/net
```

Other differences in network software distributions include the client and server programs distributed, which sometimes expect files in different places, etc. If you can't get something to work, don't yell; get the latest networking software and kernel release. Something that's wrong will probably have been discovered by others as well, so that a fix (or at least a proposed workaround) is likely to be known. Or it may simply be that some applications won't work very well together because they make false assumptions about the location of some file. After all, LINUX is "work in progress".

A good place to look before starting to configure LINUX networking is the NET-2-FAQ.

It is maintained by Matt Welsh[1] and is co-authored by a number of other people. It contains the most up-to-date information. You can get it (among others) from `tsx-11.mit.edu`, as `/pub/linux/doc/NET-FAQ`.

3.2 Software Installation

3.2.1 Installing the Binaries

For those who have been running network applications on LINUX kernels prior to release 0.99.10, note that the new kernel networking code obsoletes some commands from the NET-1 code (most notably the `config` command), and requires a number of new commands. Most application software, however, should continue to work.

Fred van Kempen has put together a complete binary and source distribution of administration tools and application programs, together with a coherent set of configuration files (not included in release 10 of the binary distribution, although the in the source package).

The TCP/IP networking software to be used with the NET-2 release is available by anonymous FTP from `tsx-11.mit.edu` in `/pub/linux/packages/net/net-2`. There are three packages below `binaries/net`:

`net-base.tar.z`
: This contains the administration tools specific to the NET-2 release, such as `ifconfig`, `route`, etc, together with configuration files and boot scripts. Note that the paths used in these files are not consistent with any binary releases prior to NET-2.

`net-std.tar.z`
: This has all basic network applications. It includes `telnet`, `rlogin`, `ftp` and NFS clients and servers.

`net-ext.tar.z`
: This currently contains nothing but a version of the `tin` newreader capable of reading and posting news via NNTP.

You may unpack all of these from the root directory, issuing

```
# zcat net-xxx.tar.z | tar xvvpoof -
```

at the shell prompt.

[1] Matt can be reached at `mdw@tc.cornell.edu`.

Alternatively, if you are a cautious person,[2] you can unpack them in some safe location and move them to the appropriate places by hand.

Sources to all programs are available from the same location, to be found below the `source` directory. The source of tools included in `net-base.tar.z` is in `source/net/net-020.tar.z` (of course, the release number will change).

3.2.2 Setting up the `proc` filesystem

Some of the configuration tools of the NET-2 release rely on the `proc` filesystem for communicating with the kernel. This is a kernel interface implemented by Michael K. Johnson that permits access to kernel run-time information through a filesystem-like mechanism. When mounted, you can list its files like any other filesystem, too, or display their contents. Typical items include the `loadavg` file that contains the system load average, or `meminfo`, which shows current core memory and swap usage.

To this, the NET-2 code adds a `net` directory. It contains a number of files that show things like the kernel ARP tables, the state of TCP connections, and the routing tables. Most network administration tools get their information from these files.

The `proc` filesystem (or `procfs` as it is also known) is not in the kernel by default, you have to make sure it has been configured in. The best way to find out if you have it, try to mount it as described below. If the `procfs` is not in your kernel, you will get a message like "`mount: fs type procfs not supported by kernel`". You will then have to recompile the kernel and answer "yes" when asked for `procfs` support.

The `procfs` is usually mounted on `/proc` at system boot time. The best method is to add the following line to `/etc/fstab`:

```
# procfs mont point:
none /proc proc defaults
```

and execute "`mount /proc`" from your `/etc/rc` script.

[2] Ever had a binary distribution overwrite your `/etc/passwd` file? `:-)`.

3.3 Hardware Configuration

3.3.1 A Tour of Linux IP interfaces

The Linux networking software comes with a number of hardware drivers. This section attempts to provide you with basic information on configuring your hardware components for TCP/IP networking.

> **Please note that due to the current pace of development, the information of this section will be rapidly outdated, so you should always check the kernel source `README`s, too.**

As described in section 1.3.2, the device drivers for the network hardware are located in the data link layer. For each type of hardware, a special driver must be provided, while the functionality they offer to the layer above (the IP networking layer) is uniform across all hardware. There are a number of attributes pertaining to an interface, for example its IP address, its broadcast address, and its MTU. The MTU is the Maximum Transfer Unit and denotes the maximum number of bytes the device is capable of transferring in one transaction.

These attributes are set when configuring the system at boot time, effectively making the hardware available to the IP networking layer.

Any hardware-specific configuration options must be set by different means. At the moment, this usually involves changing kernel compile options, and making a new kernel image. An alternative is to provide a number of `ioctl(2)` calls for each device driver that allows to set the hardware parameters. An example of this is the driver for Western Digital/SMC Elite cards that allows to configure a board from software.

Of course, you somehow have to be able to specify an interface to the IP layer. This happens by means of names assigned to the interfaces, consisting of a part describing the interface type, and a unit number. These are names defined internally in the kernel, and are not proper device files in the `/dev` directory. However, the NET-2 code supports device files for most drivers which may be used for configuring the driver, but the names of the device files and the interface need not agree.

There are a number of standard names for IP interfaces in Linux.

`eth`*n* The *n*-th Ethernet card. The current kernel supports one Ethernet board known as `eth0`.

`lo` The local loopback interface. This is used for testing purposes. It simply sends IP datagrams to your host. There's always one loopback device configured into the kernel, and there's little sense in having more or less.

`sl`*n* The *n*-th SLIP interface. SLIP interfaces are associated with serial lines in the order in which they are allocated; i.e., the first serial line being configured for SLIP becomes `sl0`, etc. The kernel supports up to four SLIP interfaces. This number may be increased when recompiling the kernel (change the `SL_NRUNIT` macro in `net/inet/slip.h` to suit your needs.

`plip`*n* The *n*-th PLIP interface. PLIP transports IP datagrams over parallel lines. Up to three PLIP interfaces are supported. They are allocated by the PLIP driver at system boot time, and are mapped onto parallel ports as described in section 3.3.4. PLIP interfaces are associated with parallel lines in the order in which they are allocated.

For other interface drivers that may be added in the future, like PPP, ISDN, or X.25, other names will be introduced.

3.3.2 The Ethernet Driver

The current Linux network code supports various brands of Ethernet cards, as well as a driver of the D-Link pocket adapter. This adapter allows to access an Ethernet through a parallel port. All drivers were written by Donald Becker (`becker@super.org`), except for the D-Link driver which was written by Bjorn Ekwall.

Currently, the drivers are only able to handle one Ethernet board, but the code will be extended to support several cards.

The following brands of boards are supported:

3Com Boards 3c503 EtherLink II and 3c503/16 EtherLink II.

Novell Eagle NE1000, NE2000. A wide variety of clones is reported to work with it.

Western Digital/SMC
WD8003 and WD8013.

Hewlett Packard
HP27245, HP27247, and HP27250.

"Supported" here means, that it should work according to the specs. Read `comp.os.linux` or the NET channel of the mailing list to find out if particular cards are troublesome. You might also want to check the README files in the `net/inet` subdirectory of the kernel source directory; they usually contain the latest list of working hardware.

To use one of these cards with LINUX, you may use a precompiled kernel that contains drivers for all of them. At boot time, the code will try to locate the board and determine its type. Cards are probed for at the following addresses and in the following order:

`WD80x3`	`0x300, 0x280, 0x380, 0x240`[3]
`3c503`	`0x300, 0x310, 0x330, 0x350, 0x250, 0x280, 0x2a0, 0x2e0`
`NEx000`	`0x300, 0x280, 0x320, 0x340, 0x360`
`HP`	`0x300, 0x320, 0x340, 0x280, 0x2C0, 0x200, 0x240`

Alternatively, you may compile the kernel yourself, including the drivers you need. To do so, go to your kernel source directory and edit `net/inet/CONFIG`, which contains extensive comments on the compile-time options available.

Sometimes the autoprobing code may not detect your card. In this case you have to 'hard-wire' your card's configuration into the kernel. You do this by re-compiling the kernel and specifying the board's address and IRQ in the file `net/inet/CONFIG` in the kernel source directory. Especially WD80x3 cards are said to exhibit this behavior. The current net-2 release (as of LINUX kernel 0.99.10) therefore disables autoprobing for WD80x3 cards and only looks for a board at `0x280`, IRQ 15.

If you're using 3Com's Etherlink card, the driver uses the AUI connector by default. The connector type is not yet settable from software, so if you want the BNC connector, you have to recompile the kernel, too.

3.3.3 The SLIP Driver

SLIP, or *Serial Line IP*, is a widely used protocol for sending IP packets over a serial link. A number of institutions provide dialup SLIP access to machines that are on the Internet, thus providing IP connectivity to private persons (something that's otherwise hardly affordable).

To use a serial port for SLIP, no hardware modifications are necessary. After connecting to the SLIP server via, for example, `kermit`, the *line discipline* is changed to `SLIPDISC`. With this setting, the serial interface cannot be accesses from user processes anymore, but only by the SLIP driver.

Changing to `SLIPDISC` is usually performed using a tool named `dip`.[4] In fact, `dip` is

[3] This is disabled at the moment; the driver only probes for a board at `0x280` on IRQ 15.

[4] `dip` apparently means *Dialup IP*. `dip` was written by Fred van Kempen.

much more versatile than this. It provides a simple script interpreter that can dial and login for you, convert the line to SLIP, and configure the network interface. See the sample script that comes with the `dip` sources, or the manual page for more information.

`dip` may as well be used set up *your* machine as a SLIP server.[5] For this, you have to set up an account, say `slip`, giving it `dip -i` as login shell.[6] When invoked with the `-i` switch, `dip` determines the user who invoked it and looks up the client host's IP address and related information in `/etc/diphosts`. This is used to set up the SLIP interface after the serial line has been converted to `SLIPDISC`. After the line is dropped, `dip` frees the interface and exits.

3.3.4 The PLIP Driver

PLIP stands for Parallel Line IP and implements a protocol for exchanging packets between two machines via their printer ports: the so-called Crynwr protocol.[7] It uses five of the port's outgoing data lines and all five of the port's incoming status lines. Four lines are used to transport the low and high nibbles of the data bytes, respectively, while the fifth is used as data strobe. This somewhat peculiar design enables it to even work with unidirectional interfaces often used for printers. A packet driver for bidirectional ports which uses the full eight bit path seems to be in preparation, but is not yet available.

For connecting two machines using PLIP, you need a special cable sold at some shops as "Null Printer" or "Turbo Laplink" cable. You can, however, make one yourself fairly easy. Appendix A shows you how.

The PLIP driver for Linux was developed by Donald Becker. If compiled into the kernel, it sets up a network interface for each of the possible printer ports. The mapping is as follows:

Interface	IRQ	Address
`plip0`	5	`0x3BC`
`plip1`	7	`0x378`
`plip2`	2	`0x278`

This mapping does however not mean that you cannot use these parallel ports as usual. They are only accessed by the PLIP driver when the corresponding interface is configured

[5] Just for kicks: how about a dialup FTP server instead of anonymous UUCP?

[6] Currently, this fails because `login` does not support arguments to the shell command. A way around this is to write a small C program that simply calls `system("dip -i");`.

[7] Although Crynwr sounds pretty much like a tiny village in Wales, it is the name of a company.

up.

3.3.5 Using `wdsetup`

If you own an Ethernet card by Western Digital or SMC, you are lucky, because there is now a tool that allows you to configure your board's hardware at run-time. It is called **`wdsetup`** and was written by Gregg Weber.[8] If your board is sophisticated enough, it will allow you to view and change the board's configuration.

There are three ways to configure an Ethernet board. Most cards come with a number of jumpers on-board that allow to set parameters like the I/O base address. With the dumbest ones, that's all there is. Smarter cards have a number of registers, whose values may be changed at any time, taking effect immediately. There may also be an EEPROM with one or more configuration pages, with page 0 usually accessible for "soft" configuration. This means it contains values that are copied into the card's registers after a hard reset. Other pages contain factory settings that may not be changed. Thus, in order for soft configuration to take effect, you have to make sure that your card's jumpers are set so that page 0 is used after a reset, instead of one of the other pages.

`wdsetup` checks if your card is soft-configurable. If it is not, it displays a warning and exits. Otherwise, it allows you to display and alter your card's registers both from the command line and interactively. For `wdsetup` to locate the card, you should provide it with its base address using the `-a` option (or `--baseaddr`, alternatively):

```
# wdsetup -a baseaddr
```

This command runs `wdsetup` in interactive mode, accessing the card at *baseaddr*. This address must be specified as a 16 bit hex number, with the first (most significant) and third digit even, the second digit between `0` and `3`, and the last one `0`. Thus, `280` and `E1C0` are valid base addresses, while `14B2` is not.

When you invoke `wdsetup` to run in interactive mode, and you do not give it the card's base address, it will try to locate it all by itself. If you have more than one card in your computer, it will show all of them, and let you choose one. The author, however, cautions that some cards (not necessarily Ethernet cards) may lock up when their registers are read at the wrong time. If this happens to you, you will have to use the `-a` option to skip auto-detection.

You may alter the base address by invoking `wdsetup` with the `-p` option. If, for example, your card is at `380` by default, you move it to `280` by

```
# wdsetup -a 380 -p 280
```

[8] To be reached at `gregg@netcom.com`.

Note that the kernel expects the board on `280` by default. If you want to put it somewhere else, you have to change the configuration file `net/inet/CONFIG` and recompile the kernel (see section 3.3.2 above).

The Ethernet cards produced by Western Digital and SMC have some dual-port RAM to improve performance. The start of this RAM may be mapped into the PC's memory space at different addresses. You may move its start address using the `-b` option. The card's IRQ may be changed using `-i`.

Some cards allow connecting different media to them. This may be set by the `-m` option, which takes one of the symbolic arguments `aui`, `bnc`, and `twp`. These stand for different connectors and media attached. `twp` denotes twisted pair, `bnc` stands for the BNC twist-on connector that is used with 10base2 Ethernet, and AUI is the 15pin Dsub connector that is used to connect to an Ethernet transceiver.

All settings may be viewed and altered in interactive mode, too. If any of the following options is given, `wdssetup` will not enter interactive mode (Where available, alternative option names are given as well):

`-e` Dump EEPROM contents.

`-r` Dump registers.

`-v` Be verbose when dumping registers.

`-p` *newaddr* or `--newaddr` *newaddr*
: Move the card's base address to *newaddr*.

`-b` *addr* or `--ramstart` *addr*
: Alter RAM start address to *addr*.

`-i` *irq* or `--irq` *irq*
: Set new interrupt number.

`-m` *media* or `--media` *media*
: Set new network media type. *media* may be one of `aui`, `bnc`, or `twp`.

These options only set the card's registers. If you want to write to the EEPROM soft configuration page, you have to use interactive mode.

.

3.4 Setting the hostname

The NET-2 binary release also provides a new `hostname` command. To set the hostname, it is invoked as

```
/etc/hostname -S [name]
```

If the *name* argument is given, your machine's hostname is set to this name. You may choose to use an unqualified hostname as well as with the domain name. Usually, one would not use the unqualified name.

If you don't give the *name* argument on the command line, `hostname` looks for the file `/etc/HOSTNAME`, and sets the hostname to what it finds in this file.

3.5 Assigning IP Addresses

Before you can start out hacking away at your keyboard, you will have to make some administrative decisions. One of them is what IP addresses to assign to your hosts. In section 2.1 we saw that hosts within a local network should be part of the same logical IP network. Hence you have to assign an IP network address. If you have several physical networks, you either have to assign them different network numbers, or use subnetting to split your IP address range into several subnetworks.

If your LAN is not connected to the Internet, you're free to choose any (legal) network address. However, you have to make sure to choose one from classes A, B, or C, else you will most probably have severe difficulties in persuading TCP/IP to work. Maybe it would be best to use one class C network for the whole of your LAN, because even if you don't even dream of getting on the Internet today, things may look different in a few years from now. To operate several Ethernets (and other hardware, once a driver is available), you have to use subnetting, of course.

However, if you intend to get on the Internet in the near future, you should obtain an official IP address *now*. The best way to proceed is to ask your network service provider to help you. If you want to obtain a network number just in case you might get on the Internet someday, request a Network Address Application Form from `hostmaster@nic.ddn.mil`.

For subnetting, assign a number to each network you manage. Note that the subnet number may not be zero, hence you start with subnet number one. If you have a "backbone" whose sole purpose is to connect the various subnets, remeber to also assign it a subnet number. Then allocate a fraction of the IP address' host part sufficient to hold the subnet numbers, plus a little room for future expansion. Make sure the remaining host part bits are still sufficient to accomodate the maximum number of hosts you will have on any of the subnets.

Then start assigning numbers to the hosts on each of the subnets, probably from different ranges for gateways and ordinary hosts, respectively. A gateway needs one IP address per network it is on.

As an example, the brewery's network manager applies to the NIC for a class B network number, and is given `192.72.0.0`. To accomodate the two Ethernets, she decides to use eight bits of the host part as additional subnet bits. This leaves another eight bits for the host part, allowing for 254 hosts on each of the subnets. She then assigns subnet number 1 to the brewery, and gives the winery number 2. Their respective network addresses are thus `192.72.1.0` and `192.72.2.0`. The subnet mask is `255.255.255.0`.

Note that in this example we are using a class B network to keep the example simple; a class C network would be more realistic. With the new networking code, subnetting is not limited to byte boundaries, so even a class C network may be split into several subnets.

`vlager`, which is the gateway between the two networks, is assigned a host number of 1 on both of them, which gives it the IP addresses `192.72.1.1` and `192.72.2.1`, respectively.

3.6 Interface Configuration for IP

Your next job is to configure the interfaces so the IP layer can use them. Usually, this is done at system boot time by executing the `rc.inet1` script from `/etc/rc.d`. The commands used in this script will be explained below.

Another important topic being addressed in this chapter is configuration of the hostname resolver software. We will cover both static host tables (using the`/etc/hosts` file) as well as use of the BIND server. If you are only running an isolated Ethernet, you may safely skip the section on `named`.

3.6.1 Interface Configuration with `ifconfig`

After each system boot, the first step in configuring your machine for TCP/IP networking is to make your interfaces known to IP, along with the IP addresses you have assigned to them. Remember from section 2.2 that the IP address is not really assigned to the host itself, but to the network interface. This is because the network part of the address has to agree for all hosts on a single physical network, hence a machine that is on several networks has to have one IP address per network.

Since the early days of LINUX networking, IP addresses have been used to be assigned to interfaces using the `config` command. Fred van Kempen's recent NET-2 release uses a command for that purpose which is more "standard", namely `ifconfig`. Since the `config` command is obsolete, we will only cover the latter.

`ifconfig`'s command line options are:

`/etc/ifconfig` *interface* `[`*address*`]` `[`*parameters*`]`

interface is the interface name as described above. *address* is the IP address to be assigned to the interface. This may either be an IP address in dotted quad notation, or a hostname. Note, however, that `ifconfig` tries to resolve the hostname by looking it up in `/etc/hosts`. Thus, when giving the hostname as the *address* parameter to `ifconfig`, you have to have this file set up on the local host.

By default, the subnet accessible through the interface is deduced from the address parameter, using the standard subnet mask pertaining to the address class. For a class B address, the default subnet mask is `255.255.0.0`, thus the network number of `192.71.1.1` is `192.71.0.0`. If subnetting is used, as is the case with the Virtual Brewery, the network number is however `192.1.1.0`. Therefore, an alternative subnet mask has to be specified for this interface using the `netmask` option (see below). In our example, the correct netmask is `255.255.255.0`.

`ifconfig` recognizes the following parameters:

`up` This marks an interface "up", i.e. accessible to the IP layer. This option is implied when an *address* is given on the command line. It may also be used to re-eenable an interface that has been taken down temporarily using the `down` option.

`down` This marks an interface "down", i.e. inaccessible to the IP layer. This effectively disables any IP traffic through the interface. Note, however, that this does not automatically delete entries in the routing table that use this interface.

`netmask` *mask*
This assigns a subnet mask to be used by the interface. It may be given as either a 32-bit hexadecimal number preceded by `0x`, or as a dotted quad of decimal numbers.

`broadcast` *address*
IP allows for special addresses that may be used to broadcast a datagram to all hosts on a logical network. This is usually the network address with

the host part bits all set to ones. Some IP implementations use a different scheme when this option is supplied, but LINUX adheres to the standard.

`metric` *number*
: This option may be used to assign a metric value to the routing table entry created for the interface. This metric is used by the Routing Information Protocol (RIP) to build routing tables for the network.[9] The default metric used by `ifconfig` is a value of zero. If you don't run a RIP daemon, you don't need this option at all; if you do, you will rarely need to change the metric value.

`mtu`
: This sets the Maximum Transmission Unit, which is the maximum number of octets the interface is able to handle in one transaction. For Ethernets, the MTU defaults to 1500; for SLIP interfaces, this is 296.

`trailers`
: This is an option specific to Ethernet interfaces. It enables so-called *trailer encapsulation*, a technique to minimize the number of memory-to-memory copies the receiving system needs to perform. This is enabled by specifying `trailers` on the `ifconfig` command line. To disable it, precede it with a dash ('`-`').

`arp`
: Like `trailers`, this is an Ethernet-specific option. It enables the use of ARP, the Address Resolution Protocol used to detect the IP addresses of hosts attached to the Ethernet. To enable it, specify `arp` on the command line; to disable it, precede the option with a dash ('`-`'). This is rarely necessary.

`pointopoint`
: This option is used for point-to-point IP links that involve only two hosts. This option is needed to configure, for example, SLIP or PLIP interfaces.

The IP address of the Ethernet board on `vlager` connecting it to the brewery subnet, is `192.72.1.1`. To bring this interface up, one would issue

```
# /etc/ifconfig eth0 192.72.1.1 netmask 255.255.255.0 \
        broadcast 192.72.1.255 arp up
```

To enable the second Ethernet board, which connects vlager to the winery's network, one issues

[9] RIP chooses the optimal route to a given host based on the "length" of the path. It is computed by summing up the individual metric values of each host-to-host link. By default, a hop has length 1, but this may be any positive integer less than 16. (A route length of 16 is equal to infinity. Such routes are considered unusable.) The `metric` parameter sets this hop cost, which is then broadcast by the routing daemon.

```
# /etc/ifconfig eth1 192.72.2.1 netmask 255.255.255.0 \
        broadcast 192.72.2.255 arp up
```

To enable the loopback interface, use

```
# /etc/ifconfig lo 127.0.0.1
```

The commands to issue on `vstout`, which is host number 2 on the brewery net, are

```
# /etc/ifconfig eth0 192.72.1.1 netmask 255.255.255.0 \
        broadcast 192.72.1.255 arp up
```

This command brings up the Ethernet interface, telling IP that it is directly connected to subnet 1. To be able to reach hosts on the winery's Ethernet, IP still needs to be given a gateway to this subnet. This is explained below in section **??**.

`ifconfig` may also be used to configure an interface as point-to-point link, which means that the medium used only connects two hosts. In rare cases, this might be applied to Ethernet links, but usually, point-to-point links are used for SLIP or PLIP connections. To configure an interface as point-to-point link, you invoke it using the `pointopoint` option:

`# ifconfig` *device address* `pointopoint` *remote*

Here, *remote* is the remote host's address given either as dotted quad or as a hostname to be resolved using `/etc/hosts`.

Without an address or any other parameters specified, `ifconfig` will simply display the interface's configuration. This allows to check the setting of an interface:

```
# /etc/ifconfig eth0
eth0      IP ADDR 192.72.1.1  BCAST 192.72.1.255  NETMASK 255.255.255.0
          MTU 1500  METRIC 0  POINT-TO-POINT ADDR 0.0.0.0
          FLAGS: 0x0043 ( UP BROADCAST RUNNING )
```

Without any parameter at all, `ifconfig` displays the above type of information for all interfaces marked `up`.

3.7 Building IP Routing tables

To enable TCP/IP to communicate with remote hosts across physical network boundaries, IP must be provided with information on how to reach these hosts. Section 2.2 describes how this is done: IP keeps a table that maps logical networks to the interface and gateway they may be accessed through. This section describes several alternative ways this may be done.

Routing tables may be created in several ways. The case of an isolated Ethernet is the simplest one: the route to the network is entered into the table at boot time using a command named `route`, and nothing more than this is needed. However, if you have more than one physical network, you need gateways to switch packets between them, and you have to create additonal routing table entries to direct traffic to the proper gateway. These routes have to be entered at boot time, too, and may optionally be monitored and adjusted during service. To build static routing tables, the `route` command must be used. Dynamic routing is performed by *routing daemons* like `routed` or `gated`, which exchange information on network reachability, route lengths, etc. Some routing protocols provided by `gated` also measure network load. Unless your network is really very large, or richly interconnected, you will have no use for running any of these. For this reason, only static routing tables will be discussed in this book.

In the preceding section, you have already seen that IP considers an interface as a special kind of route to a network. Routing table entries, however, may also point to gateways, or single hosts (for point-to-point links). Thus, there are a number of attributes a routing table entry may have:

`G` This flag is set if the route uses a gateway.

`U` This flag is set if the interface to be used is up.

`H` This flag is set if only a single host can be reached through the route. For example, this is the case for the loopback entry `127.0.0.1`.

`N` This is the opposite of the `H` flag. It is set whenever the interfaces accesses a subnet.

`D` This is set if the table entry has been generated by an ICMP redirect message (see 1.3.4).

`M` This is set if the table entry was modified by an ICMP redirect message.

These flags are also displayed by the `netstat` command, which is explained below in section 3.8.

3.7.1 Route to the Subnet

Static routing tables are built by using `route`. A simple way of invoking it is

`route add` *destination*

destination specifies the host or network the routing table entry points to. You may either specify an IP address in dotted quad notation, or a symbolic host or network name. Symbolic names are looked up in `/etc/hosts` and `/etc/networks`. If it is found in either of them, the corresponding IP address is used as a host or network address, respectively.

For the Virtual Brewery, the administrator would put together the following `/etc/networks` file and install it on all hosts:

```
# Virtual Brewery /etc/networks file
# Maps symbolic network names to subnet numbers.
loop-net        127.0.0.0       # loopback network
brew-net        192.72.1.0      # subnet 1
wine-net        192.72.2.0      # subnet 2
```

We are now ready to set up the first entries in the routing tables at the Virtual Brewery. First, we do so on `vlager`, the host that joins the two Ethernets, by executing

```
# route add 127.0.0.1
# route add brew-net
# route add wine-net
```

The first command creates an entry for `127.0.0.1`, while the following enter a route to each of the subnets. At first glance, this looks pretty stupid: we already *did* specify a subnet number when configuring the Ethernet interfaces, the loopback device *has* been assigned number `127.0.0.1`, and anyway, why don't these commands specify the interfaces the networks may be reached by?

The answer is this: After configuring the interfaces, the routing table entry is completely empty. Although the kernel may receive IP packets from these devices, it cannot send any. To be able to, it has to be told exactly *what* subnet is accessible through the interface. For example, the loopback device seems to offer access to a class A network, because `127.0.0.1` is a class A address. IP has to be told explicitly that `lo` is an interface to a single host. Analoguously, the routes to the two subnets have to be enabled explicitly.

Now, why don't we have to specify the interface to `route`? This is because after configuring `eth0` with `ifconfig`, the IP layer knows that *if* there is a route to subnet `192.72.1.0`, it can only use `eth0`!

When giving `route` an IP address in dotted quad notation, it attempts to guess whether it is a network or a hostname by looking at the host part bits. If the address' host part is zero, `route` assumes it denotes a network, otherwise it takes it as a host address.

Of course, this heuristic fails when you use subnetting. To override this check, you may precede the IP address with one of the modifiers `-net` or `-host`, respectively. This forces `route` into treating the address as a network or a host, respectively.

Thus, the Brewery's network administrator might alternatively have skipped putting the network names into `/etc/networks`, and edited `rc.inet1` directly:

```
# route add 127.0.0.1          # local host
# route add -net 192.72.1.0    # brewery subnet
# route add -net 192.72.2.0    # winery subnet
```

3.7.2 Routes through a Gateway

In the presence of several subnets, the Linux IP layer has to be told what gateways to reach them through. This is done using the `route` command, too. You invoke it as

`route add` *destination* `gw` *gateway*

When you are using your Linux box in a larger network environment,

destination is the destination subnet or host, while *gateway* is the gateway to reach it through. They may be either given in dotted quad notation, or as a hostname listed in `/etc/hosts`. The *network* argument may also be the keyword `default`, which creates a routing table entry for the default route `0.0.0.0`. The default route names the gateway all datagrams to unknown networks are forwarded to.

Again, the `route` command does not take a command line option that specifes the interface to use in accessing the gateway. Instead, it tries to determine it from the gateway's address: the address is checked against the routing tables constructed so far, so you have to make sure the address is on a subnet you have already specified a route to.

For example, `vstout` has to use `vlager` as a gateway when communicating with a host on the winery subnet. However, `vlager` has two addresses, being `194.72.1.1` and `194.72.2.1`. Of course, when adding the route to the winery subnet, you have to use the first IP address because this is on the subnet the kernel already knows how to route to.[10] Thus, the sequence for enabling routing on `vstout` is

```
# route add 127.0.0.1
# route add brew-net
# route add wine-net gw 194.72.1.1 metric 1
```

The `metric` argument is used to give some idea of the route's "cost", usually counting the number of gateways. It is used by some routing protocols, like RIP, to choose between different routes if there is more than one way for routing to a given subnet. Although the metric argument is not needed if you use static routing, there is a rule that says that routes whose gateway is the local host should have a metric of zero, while those that use a remote gateway should have a value greater than zero. If you don't run any routing daemon, it's best to provide a *metric* of 1 for all routes to remote gateways. If you omit *metric*, it defaults to zero.

To delete a route from the routing table, you invoke `route` like this:

`route del` *destination*

If invoked without any arguments, `route` displays the routing table, including some interface statistics.[11] On `vstout`, this produces:

```
# route -n
Kernel routing table
Destination net/address   Gateway address          Flags RefCnt    Use Iface
127.0.0.1                 *                        UH         0     50 lo
192.72.1.0                *                        UN         0    478 eth0
192.72.2.0                192.72.1.1               UGH        0    250 eth0
```

The `-n` option makes `route` print host addresses as dotted quad IP addresses. Without this option, it would attempt to resolve them to symbolic hostnames.

3.8 Verifying your IP Setup

There's a number of useful tools for verifying the operation of your TCP/IP network setup. We will describe them below.

[10] Note that the gateway need not be on a subnet you are directly attached to, but you may also specify one that is itself only reached through one or more gateways. Routes with this property are called *indirect*.

[11] In fact, it simply invokes `netstat` with the `-r` switch. `netstat` is explained below in section 3.8.2.

3.8.1 ping

ping is the networking equivalent of a sonar device.[12] It measures the time a datagram requires to travel from your host to a destination machine and back again. For this, it uses a special feature of the ICMP protocol, the “echo” message. When a host’s IP layer receives an echo message, it returns a response bearing a time stamp, plus the sequence number of the original packet.[13] ping sends such ICMP messages to check for the reachability of a host, as well as the delays encountered. A simple way to invoke it is thus:[14]

```
$ /etc/ping nic.funet.fi
PING nic.funet.fi: 64 byte packets
64 bytes from 128.214.6.100: icmp_seq=7. time=8998. ms
64 bytes from 128.214.6.100: icmp_seq=8. time=11530. ms
64 bytes from 128.214.6.100: icmp_seq=9. time=12368. ms
64 bytes from 128.214.6.100: icmp_seq=10. time=11762. ms
64 bytes from 128.214.6.100: icmp_seq=13. time=12172. ms
64 bytes from 128.214.6.100: icmp_seq=17. time=12326. ms
^C

----nic.funet.fi PING Statistics----
30 packets transmitted, 6 packets received, 80% packet loss
round-trip (ms)  min/avg/max = 8998/11526/12368
```

By default, ping will go on emitting packets forever. To stop it, you have to interrupt it by typing Ctrl-C. This causes ping to stop and compute the packet loss rate, which is the ratio of outstanding responses to the total number of packets sent.

The above example shows very loaded lines, with many packets getting dropped. On the Internet, it’s not that uncommon for packets to disappear or to arrive unordered. This is not so much due to technical deficiencies as due to temporary excess loads on forwarding hosts, which makes them delay or even drop incoming datagrams.[15] On a LAN, you should never see such a thing. The output of ping should rather be like this:[16]

```
$ /etc/ping vstout
PING vstout: 64 byte packets
```

[12] Anyone remember Pink Floyd’s “Echoes”?

[13] There’s more information in it, but we won’t go into this.

[14] Some ping programs will only say “foo is alive” when being invoked without any option. To obtain the behavior shown in the example, you may have to add the -s switch on other operating systems.

[15] A host has to drop packets, for example, when the buffer space available does not suffice to accomodate another incoming packet. Dropped packets simply disappear, they are not rescheduled by the host immediately preceding the one that drops it. The reason for this is potential deadlock avoidance.

[16] Note that you cannot give the destination host’s name on the command line unless you have properly set up address resolution.

```
64 bytes from 192.72.1.2: icmp_seq=0. time=11. ms
64 bytes from 192.72.1.2: icmp_seq=1. time=7. ms
64 bytes from 192.72.1.2: icmp_seq=2. time=12. ms
64 bytes from 192.72.1.2: icmp_seq=3. time=3. ms
^C

----vstout.linus.lxnet.org PING Statistics----
5 packets transmitted, 4 packets received, 20% packet loss
round-trip (ms)  min/avg/max = 3/8/12
```

The packet loss in this example is due to us interrupting `ping` while waiting for `vstout`'s response to packet number 4.

If you encounter unusual packet loss rates, this hints at a hardware problem, like bad or missing terminators, etc. If you don't receive any packets at all, or even see error messages, this means there is a configuration problem. You may probe for these problems using `wdsetup`, described above in section 3.3.5, and `netstat`, and `arp`, described below.

You may also check for all reachable hosts on your LAN by specifying the broadcast address[17]:

```
$ /etc/ping 192.72.1.255
PING 192.72.1.255: 64 byte packets
64 bytes from 192.71.1.1: icmp_seq=0. time=27. ms
64 bytes from 192.71.1.4: icmp_seq=0. time=34. ms
64 bytes from 192.71.1.2: icmp_seq=0. time=39. ms
64 bytes from 192.71.1.3: icmp_seq=0. time=44. ms
64 bytes from 192.71.1.4: icmp_seq=1. time=39. ms
64 bytes from 192.71.1.2: icmp_seq=1. time=68. ms
64 bytes from 192.71.1.1: icmp_seq=1. time=103. ms
64 bytes from 192.71.1.3: icmp_seq=1. time=129. ms
^C

----192.71.1.255 PING Statistics----
2 packets transmitted, 8 packets received, -400% packet loss
round-trip (ms)  min/avg/max = 27/55/129
```

[17] If you have Internet access, please, *never* do this with any network but your local subnet. This will clog your (and others') networks for a while, and will surely get you visits by friendly people speaking softly and carrying big sticks.

3.8.2 netstat

This is a versatile command for checking various aspects of your network interfaces' configuration, like viewing the routing tables, show interface statistics (this is not yet there), or all connections active.

The `-r` option to `netstat` is used to display the kernel routing tables. When invoking `netstat` on `vstout`, the output may look like this:

```
$ netstat -nr
Routing tables
Destination      Gateway            Flags     Refs      Use  Interface
127.0.0.1        *                  UH        1527    66614  lo
192.72.1.0       *                  UN         556  7917876  eth0
192.72.2.0       192.72.1.1         UGN          0    76735  eth0
```

The first column is the destination network the entry applies to. The second is the gateway to reach it through, and the last one is the interface to be used. The flags field contains a combination of the letters `U`, `G`, `H`, `N` and `D`, which denote a route through an operating ("up") interface, a route through a gateway, a route to a single host (i.e. a point-to-point link), and a route generated by an ICMP redirect message.

Other options are those to show active connections. Using the `-t`, `-u`, `-w` and `-x` limits the display to certain protocols only, namely TCP, UDP, RAW, and Unix domain sockets. Invoking `netstat -t` on `vlager` produces

```
$ netstat -t
Active Internet connections
Proto Recv-Q Send-Q Local Address          Foreign Address        (State)
tcp        0      0 *:domain               *:*                    LISTEN
tcp        0      0 *:time                 *:*                    LISTEN
tcp        0      0 *:smtp                 *:*                    LISTEN
tcp        0      0 vlager:smtp            vstout:1040            ESTABLISHED
tcp        0      0 *:telnet               *:*                    LISTEN
tcp        0      0 localhost:1046         vbardolino:telnet      ESTABLISHED
tcp        0      0 *:chargen              *:*                    LISTEN
tcp        0      0 *:daytime              *:*                    LISTEN
tcp        0      0 *:discard              *:*                    LISTEN
tcp        0      0 *:echo                 *:*                    LISTEN
tcp        0      0 *:shell                *:*                    LISTEN
tcp        0      0 *:login                *:*                    LISTEN
```

This shows most servers simply waiting for an incoming connection. However, the fourth line shows an incoming SMTP connection from `vstout`, and the sixth line tells you there is an outgoing `telnet` connection to `vbardolino`.[18]

Output for TCP and UDP sockets may be combined by invoking `netstat` with the `-a` switch. For more information, please refer to the manual page.

3.8.3 arp

In section 1.3.2 we already mentioned that the Ethernet driver uses the ARP protocol to query for the Ethernet address corresponding to a given IP address. However, there are other types of equipment that also allow to use the ARP protocol; the ham-radio AX.25 hardware is an example of this. Although a driver for this is not yet there, `arp` already provides support for this.

The `arp` tool may be used to view and alter the kernel ARP tables. Its command line options are

/etc/arp [-v] [-t *hwtype*] -a [*hostname*]
/etc/arp [-v] [-t *hwtype*] -s *hostname hwaddr*
/etc/arp [-v] -d *hostname* [*hostname...*]

All *hostname* arguments are expected to be symbolic host names (no IP address), so that using `arp` is only possible after having configured address resolution. It is, however, also possible to specify an IP address.

The first invocation displays the ARP tables related to the specified host, or all hosts known. For example, invoking `arp` on `vlager` may yield

```
# arp -a
IP address      HW type               HW address
192.72.1.3      10Mbps Ethernet       00:00:C0:5A:42:C1
192.72.1.2      10Mbps Ethernet       00:00:C0:90:B3:42
192.72.2.4      10Mbps Ethernet       00:00:C0:04:69:AA
```

When using the `-a` option together with a *hostname* only shows the ARP table entry for that host if there is one. Using the `-t` option limits the display to the hardware type specified. This may be one of `ether`, `ax25`, or `pronet`, standing for 10Mbps Ethernet, AMPR AX.25, and ProNet equipment,[19] respectively.

[18] You can tell wether a connection is outgoing or not from the port numbers involved. The port number shown for the *calling* host will always be a simple number higher than `1024`, while on the host being called, a well-known service port will be in use, for which `netstat` uses the symbolic name found in `/etc/services`.

[19] Can anybody tell me what this is?

The `-s` option is used to permanently add *hostname*'s Ethernet address to the ARP tables. The *hwaddr* argument specifies the hardware address, which is by default expected to be an Ethernet address, specified as six hexadecimal bytes separated by colons (':'). You may also set the hardware address for other types of hardware, too, using the `-t` option. One problem which may require you to use the `-s` option is when for some reasons ARP queries for the remote host fail, whether its ARP driver is buggy or there is another host in the network that erroneously identifies itself with `hostname`'s IP address.[20]

Invoking `arp` using the `-d` switch deletes all ARP entries relating to the given host. This may be used to force the interface to re-attempt to obtain the Ethernet address for the IP address in question.

3.9 Name Service and Resolver Configuraton

As explained in section 2.3 above, TCP/IP networking may rely on different schemes to convert names into addresses. The simplest way, which takes no advantage of the way the name space has been split up into zones, is a host table stored in a single file. This is only useful for small LANs that are run by one single administrator, and otherwise have no IP traffic with the outside world. This information is stored in a file named `/etc/hosts`, whose format is described below in 3.9.1.

Alternatively, one may use BIND for resolving host names to IP addresses. This may be a real chore, but once you've done it, changes in the network topology are easily made. On LINUX, as on many other Un⋆xish systems, name service is provided through a program called `named`. At startup, it loads a set of master files into its cache, and waits for queries from remote or local user processes. There are different ways to set up BIND, and not all require to run a name server on every host.

Whatever way you may choose, you have to configure your software to adapt to the scheme used.

When talking of "the resolver", we do not mean any special application, but rather refer to the *resolver library*, a collection of functions that can be found in the standard C library. The central routines are `gethostbyname(2)` and `gethostbyaddr(2)` which look up all IP addresses belonging to a host, and vice versa. They may be configured to simply look up the information in `/etc/hosts`, query a number of name servers, or use NIS (Network Information Service). Other applications, like `smail`, may include different drivers for any of these, and need special care taken of them.

[20]**Author's Note: I'm wondering if this can also be used for proxy ARPing. Can anybody help me with this?**

3.9.0.1 The `host.conf` File

The central file that controls your resolver setup is `host.conf`. It resides in `/etc` and describes to the resolver which services to use, and in what order.

Options in `host.conf` must occur on separate lines. Fields may be separated by white space (spaces or tabs). A hash sign (‘`#`’) instroduces a comment that extends to the next newline.

The following options are available:

`order` This determines the order in which the resolving services are tried. Valid options are `bind` for queryig the name server, `hosts` for lookups in `/etc/hosts`, and `nis` for NIS lookups. Any or all of them may be specified. The order in which they appear on the line detemines the order in which the respective services are tried.

`multi` Takes `on` or `off` as options. This detemines if a host in `/etc/hosts` is allowed to have several IP addresses. It has no effect on DNS or NIS queries.

`nospoof` As explained above in section 2.3, DNS allows to find the hostname belonging to an IP address by using the `in-addr.arpa` domain. Attempts by name servers to supply a false hostname are called “*spoofing*”. To guard against this, the resolver may be configured to check if the original IP address is in fact associated with the hostname obtained. If not, the name is rejected and an error returned. This behavior is turned on by setting `nospoof on`.

`alert` This option takes `on` or `off` as arguments. If it is turned on, any spoof attempts (see above) will cause the resolver to log a message to the `syslog` facility.

`trim` This option takes a domain name as an argument, which will be removed from hostnames before lookup. This is useful for `/etc/hosts` entries, where you might only want to specify hostnames sans local domain. A lookup of a host with the local domain name appended will have this removed, thus allowing the lookup in `/etc/hosts` to succeed.

`trim` options accumulate, making it possible to consider your host as being local to several domains.

A sample file for `vlager` is shown below:

```
# /etc/host.conf
```

```
# We have named running, but no NIS (yet)
order   bind hosts
# Allow multiple addrs
multi   on
# We have no one to spoof us
nospoof on
# Trim local domain (not really necessary).
trim    linus.lxnet.org.
```

3.9.0.2 Resolver Environment Variables

The settings from `host.conf` may be overridden using a number of environment variables. These are

`RESOLV_HOST_CONF`
: This specifies a file to be read instead of `/etc/host.conf`.

`RESOLV_SERV_ORDER`
: Overrides the `order` option given in `host.conf`. Services are given as `hosts`, `bind`, and `nis`, separated by a space, comma, colon, or semicolon.

`RESOLV_SPOOF_CHECK`
: Determines the measures taken against spoofing. It is turned off using `off`. The values `warn` and `warn off` enable spoof checking, but turn logging on and off, respectively. A value of `*` turns on spoof checks, but leaves the logging facility as defined in `host.conf`.

`RESOLV_MULTI`
: A value of `on` or `off` may be used to override the `multi` options from tt host.conf.

`RESOLV_OVERRIDE_TRIM_DOMAINS`
: This environment specifies a list of trim domains which override the one given in `host.conf`.

`RESOLV_ADD_TRIM_DOMAINS`
: This environment specifies a list of trim domains which are added to those given in `host.conf`.

3.9.1 What /etc/hosts looks like

Originally, all resolving information on Internet hosts was kept in a HOSTS file that was generated regularly from a centrally maintained file named HOSTS.TXT. For small Un⋆x systems that are not on the Internet, using static host tables is still standard procedure. Data linking IP addresses to hostnames is kept in a file called /etc/hosts.

It contains one entry per line which consists of an IP address, a hostname, and an optional list of aliases for the hostname. The hostname field must begin in column one. The fields are separated by spaces or tabs. Anything between a hash sign ('#') and the next newline is regarded as a comment and is ignored. The numbers of the IP address may be given using the standard C format for decimal, octal, or hexadecimal numbers. Octal numbers are denoted by a prefix of 0, hexadecimal numbers have a prefix of 0x.

Hostnames may either be local (i.e. without any domain name qualification at all), or fully qualified by a domain name. To enable the resolver to find local hosts independently of whether their name is given fully qualified, or in its local form, one usually enters both in the hosts file. Alternatively, one might only enter the local name, and include "trim *localdomain*" in host.conf (see section 3.9.3 below), where *localdomain* is the local domain name. Note that the domain statement in resolv.conf has no effect on lookups in /etc/hosts.

This is an example how a hosts file at the virtual Brewery might look:

```
#
# Virtual Brewery Ethernet.
#
# IP            local       fully qualified domain name
192.72.1.1      vlager      vlager.linus.lxnet.org
192.72.1.2      vstout      vstout.linus.lxnet.org
192.72.1.3      vale        vale.linus.lxnet.org
#
# Virtual Winery Ethernet
#
192.72.2.1      vlager       vlager.linus.lxnet.org
192.72.2.2      vbeaujolais  vbeaujolais.linus.lxnet.org
192.72.2.3      vbardolino   vbardolino.linus.lxnet.org
192.72.2.4      vchianti     vchianti.linus.lxnet.org
```

3.9.2 The /etc/networks file

Analoguously to the hosts file, there is a file for mapping network names to IP network addresses. It is called `/etc/networks`, and was already mentioned above.

Since the network names refer to the physical organization of the network, they are generally not domain names, but anything like "`brew-lan`" or "`loopback`". Entries in the file begin in column one, and contain the network name, followed by the network's IP address, and an optional list of aliases. The numbers of the IP address may be given using either of decimal, octal, or hexadecimal numbers in the standard C format. The fields are separated by spaces or tabs. Comments are introduced by a hash sign ('`#`') and extend to the next newline.

A sample file for the Virtual Brewery is given below:

```
#
# /etc/networks for the Virtual Brewery.
#
# The loopback network
loopback        127
#
# The brewery's Ethernet
brew-lan        192.72.1.0     brew-net
wine-lan        192.72.2.0     wine-net
```

3.9.3 Configuring Name Server Lookups — resolv.conf

When configuring the resolver library to use BIND name service for host lookups, you have to tell it which name servers to use, etc. There is a separate file for this, called `resolv.conf`. If this file does not exist or is empty, the resolver assumes the name server is on your local host.

If you run a name server on your local host, you have to set it up separately, as will be explained in section 3.10 below. If your are on a local network and have the opportunity to use an existing nameserver, this shoud always be preferred.

You may set the following options in `resolv.conf`:

`nameserver` This gives the IP address of a name server to the resolver. `nameserver` options accumulate. For BIND lookups, they are tried in the order in which they appear in the configuration file. Currently, up to three name servers are supported.

If no `nameserver` option is given, the resolver attempts to connect to the name server on the local host.

`domain` This specifies a domain name to be tacked onto all hostnames if BIND fails to resolve it with the first query. This is to allow to resolve hostnames specified relative to the local domain.

If no `domain` option is present, the resolver obtains it from the local hostname through `gethostname(2)`. If no domain part is present in the local hostname, the root domain is assumed.

`search` This is an alternative to the `domain` option: they are mutually exclusive. `search` may be used to specify a list of domains to be tacked onto a hostname after a query fails. The items in the list are separated by spaces or tabs.

By default, this list contains the local domain, followed by the list of all parent domains up to the root. For any host at the brewery, this would contain `linus.lxnet.org`, `lxnet.org`, and `org`. Any attempt to resolve `foo` would result in the lookup of `foo`, `foo.linus.lxnet.org`, `foo.lxnet.org`, and `foo.org`.

The following gives a sample `resolv.conf` file for the Virtual Brewery:

```
# /etc/resolv.conf
# Our domain
domain          linux.lxnet.org
#
# If no nameserver is specified, the resolver will query
# the local server.
# To use vlager as central nameserver, uncomment the line below:
#nameserver     192.72.1.1
```

3.9.4 Resolver Robustness

If you are running a LAN inside a larger network, you will most certainly use central name servers if they are available. The advantage of this is that these will develop rich caches, since all queries are forwarded to them. This scheme, however has a drawback: when a fire recently destroyed the backbone cable at our university, no more work was possible on our department's LAN, because the resolver couldn't reach any of the name servers anymore. There was no logging in on X terminals anymore, no printing, etc.

Although it is not very common for campus backbones to go down in flames, one might want to take precautions against cases like these.

Three options deal with setting up a local name server. This may either be a caching-only name server that has little but a `cache` and a `forwarders` statement in its boot file, or a secondary server for the campus zone. The drawback of these schemes are that the local name server still needs the central servers at boot time. This can be circumvented by creating a zone of authority for your domain. The administrative overhead involved with setting up a zone may forbid this option.

A fourth option, which lies somewhere in the middle of all these, is to maintain a backup host table for your domain or LAN in `/etc/hosts`. In `/etc/host.conf` you would then include "`order bind hosts`" to make the resolver fall back to the hosts file if the central name server is down.

3.10 Running named

The program that provides domain name service on Un⋆x machines is usually called `named` (pronounced *name-dee*). This is a server program originally developed for BSD, that provides name service to clients, and possibly to other name servers. It is usually started at system boot time, and runs until the machine goes down again.

It takes its information from a configuration file called `named.boot`, and various files that contain data mapping domain names to addresses and the like. The latter are called *zone files*. The formats and semantics of these files will be explained in the following section.

To run `named`, simply enter

```
# /etc/named
```

at the prompt. `named` will come up, read the `named.boot` file and any zone files specified therein. It writes its process id to `/etc/named.pid` in ASCII, downloads any zone files from primary servers, if necessary, and starts listening on port `53` for DNS queries.

At startup, `named` reads a configuration file called `/etc/named.boot`. This file is generally very small and contains little else but pointers to master files containing zone information, and pointers to other name servers. Comments in the boot file start with a semicolon ('`;`') and extend to the next newline.

The following options are available:

`directory` This specifies a directory local to which the names of zone files may be given. Different directories may be specified by repeatedly using `directory`.

`primary` This takes a *domain name* and a *file name* as an argument, declaring the local server authoritative for the named domain. As a primary server, `named` loads the zone's information from the given *file*.

Generally, there will always be at least one `primary` entry in every boot file, namely for reverse mapping of network `127.0.0.0`, which is the local loopback network.

`secondary` This statement takes a *domain name*, an *address list*, and a *file name* as an argument. It declares the local server a secondary master server for the domain specified.

A secondary server holds authoritative data on the domain, too, but it doesn't gather it from files, but tries to download it from the primary server. The IP address of at least one primary server must thus be given to `named` in the address list. The local server will contact each of them in turn until it successfully transfers the zone database. The data received is stored in the file given in the last argument and retrieved at the next nameserver boot. This provides the nameserver with a complete (although probably outdated) copy of the master files, even if none of the primary servers could be reached.

`cache` This takes a *domain* and a *file name* as arguments. The local server's cache is initialized from this file. Generally, this is used to load the information on the name servers for the root domain, so the *domain* argument is generally the root domain "`.`". The file is generally called `/etc/named.ca`.

If this statement does not occur in the boot file, `named` will not develop a local cache at all. This will severely degrade performance and increase network load if the next server queried is not on the local net.

`forwarders` This statement takes an *address list* as argument. The IP addresses in this list specify a list of servers to `named` that may be queried if it fails to resolve a query from its local cache. They are tried in order until one of them responds to the query.

Two things have to be considered before adding a forwarder to the list. Above all, you have to be sure all listed servers are willing and able to perform recursive queries for you. Secondly, the more hosts use a name server, the faster its cache grows. You should experiment with the number of forwarders you install on your local network.

`slave` This statement makes the name server a *slave* server. That is, it will never perform recursive queries itself, but only queries the forwarders specified with the `forwarders` statement.

Two more options are available whose use is no longer recommended. These are `sortlist` and `domain`. We will not describe them here.

An example `named.boot` file for `vlager` is given in figure 3.1.

```
;
; /etc/named.boot file for vlager.linus.lxnet.org
;
directory     /etc/domains
;
cache         .                         named.ca
primary       linus.lxnet.org.          named.hosts
primary       0.0.127.in-addr.arpa.     named.local
primary       72.192.in-addr.arpa.      named.rev
```

Figure 3.1: The `named.boot` file for `vlager`.

The commands `cache` and `primary` mentioned above load information into `named`. These files contain textual representations of DNS resource records, which we will look at below. Additionally, there are two directives that may be used inside these files. These are `$INCLUDE` and `$ORIGIN`. Since they are rarely needed, we will not describe them here.

Files included by `named`, like `named.hosts`, always have a domain associated with them, which is called the *origin*. This allows the administrator to specify domain and host names relative to this domain. A name given in a configuration file is considered *absolute* if it ends in a single dot, otherwise it is considered relative to the origin. The origin all by itself may be referred to using "`@`".

The first resource record (RR for short) in most master files is a `SOA` record. An exception from this is the cache file that contains the addresses of the root name servers.

Resource record representations in the master files share a common format, which is

[*domain*] [*ttl*] [*class*] *type* *rdata*

Fields are separated by spaces or tabs. An entry may be continued across several lines if an opening brace occurs before the first newline, and the last field is followed by a closing brace. Anything between a semicolon and a newline is ignored.

domain This is the domain name to which the entry applies. If no domain name is given, the RR is assumed to apply to the domain of the previous RR.

ttl — In order to force resolvers to discard information after a certain time, each RR is associated a "*time to live*", or *ttl* for short. The *ttl* field specifies the time in seconds the information is valid after it has been retrieved from the server. It is a decimal number at most eight digits wide.

If no *ttl* value is given, it defaults to the value of the *minimum* field of the preceding `SOA` record.

class — This is an address class, like `IN` for IP addresses, or `HS` for objects in the Hesiod class. For TCP/IP networking, you have to make this `IN`.

If no *class* field is given, the class of the preceding RR is assumed.

type — This describes the type of the RR. The most common types are `A`, `SOA`, `PTR`, and `NS`. The following sections describe the various types of RR's.

rdata — This holds the data associated with the RR. The format of this fields depends on the type of the RR. Below, it will be described for each RR separately.

The following is an incomplete list of RRs to be used in DNS master files. There are a couple more of them, which we will not explain. They are experimental, and of little use generally.

`SOA` — This describes a zone of authority (`SOA` means "Start of Authority"). It signals that the records following the `SOA` RR contain authoritative information for the domain. The resource data contains the following fields:

origin — This is the canonical hostname of the primary name server for this domain. It is usually given as an absolute name.

contact — This is the email address of the person responsible for maintaining the domain, with the '`@`' sign replaced by a dot.

serial — This is the version number of the zone file, expressed as a a single decimal number. Whenever data is changed in the zone file, this number should be incremented.

The serial number is used by secondary name servers to discern when zone information has changed. To stay up to date, secondary servers request the primary server's `SOA` record at certain intervals, and compare the serial number to that of the cached `SOA` record. If the number has changed, the secondary servers transfers the whole zone database from the primary server.

refresh This specifies the interval in seconds that the secondary servers should wait between checking the `SOA` record of the primary server. Again, this is a decimal number at most eight places wide.

Generally, the network topology doesn't change too often, so that this number should specify an interval of roughly a day for larger networks, and even more for smaller ones.

retry This number determines the intervals at which a secondary server should retry contacting the primary server if a request or a zone refresh fails. It must not be too low, or else a temporary failure of the server or a network problem may cause the secondary server to waste network resources. One hour, or perhaps one half hour, might be a good choice.

expire This specifies the time in seconds after which the server should finally discard all zone data if it hasn't been able to contact the primary server. It should normally be very large. Craig Hunt ([Hunt92]) recommends 42 days.

minimum This is the default ttl value for resource records that do not explicitly specify one. This requires other name servers to discard the RR after a certain amount of time. It has however nothing to do with the time after which a secondary server tries to update the zone information.

minimum should be a large value, especially for LANs where the network topology almost never changes. A value of around a week or a month is probably a good choice. In the case that single RRs may change more frequently, you can still assign them different ttl's.

`A` This associates an IP address with a hostname. The resource data field simply contains the address in dotted quad notation.

`NS` This points to a master name server of a subordinate zone. For an explanation of the need to have `NS` records, see section 2.3. The resource data field contains the hostname of the name server. To resolve the hostname, an additional `A` record is needed, the so-called *glue record* which gives the name server's IP address.

`CNAME` This associates an alias for a hostname with its *canonical hostname*. The canonical hostname is the one the master file provides an `A` record for; aliases are simply linked to that name by a `CNAME` RR, but don't have any other records of their own.

`PTR` This type of record is used to associate names in the `in-addr.arpa` domain with hostnames. This is used for reverse mapping of IP addresses to hostnames.

`MX` This RR announces a *mail exchanger* for a domain. The reasons to have mail exchangers are discussed in section 8.4.

The syntax of an `MX` record is

[*domain*] [*ttl*] [*class*] `MX` *preference host*

host names the mail exchanger for *domain*. Every mail exchanger has a *preference* associated with it. A mail transport agent who desires to deliver mail to *domain* will try all hosts who have an `MX` record for this domain until it succeeds. The one with the lowest preference value is tried first, then the others in order of increasing preference value.

`WKS` This RR advertises *well known services* offered by the host. The syntax is

[*domain*] [*ttl*] [*class*] `WKS` *address protocol services*

address is the host's IP address written in dotted quad notation. The protocol used by the service, which must either be TCP or UDP, is named in the *protocol* field. There may be only one `WKS` record per protocol. Finally, `services` is a list of service names as found in the `/etc/services` file (see section 4.1). The service names are separated by blanks or tabs, and may be continued across newlines using braces.

`HINFO` This record provides information on the system's hardware and software. Its syntax is

[*domain*] [*ttl*] [*class*] `HINFO` *hardware software*

The *hardware* field identifies the hardware used by this host. There are special conventions to specify this, a list of valid names is given in the "Assigned Numbers" RFC1060. If the field contains any blanks, it must be enclosed in quotes ('"'). The *software* field names the operating system software used by the system. Again, a valid name from the "Assigned Numbers" RFC should be chosen.

Figures 3.2, 3.3, 3.4, and 3.5 give sample files for a name server at the brewery, located on `vlager`. Owing to the nature of the network discussed (a single LAN), the example is pretty straightforward. If your requirements are more complex, and you can't get `named` going, get Craig Hunt's "TCP/IP Network Administration" ([Hunt92]). He has an extensive example for setting up a network of a couple of LANs and an Internet link.

```
;
; /etc/domain/named.ca         Cache file for the brewery.
;               We're not on the Internet, so we don't need
;               any root servers.
;
; Example entry (note loooong expiry interval)
; A.ISI.EDU.    99999999   IN    A   26.3.0.103
;                          IN    NS  A.ISI.EDU.
```

Figure 3.2: The `named.ca` file.

3.10.1 Verifying the Resolver Setup

There's a fine tool for checking the operation of your name server setup. It is called `nslookup`, and may be used both interactively and from the command line. In the latter case, you simply invoke it as

`nslookup` *hostname*

and it will query the name server specified in `/etc/resolv.conf`. (If this file names more than one server, it will choose one at random).

The interactive mode, however, is much more exciting. Besides looking up individual hosts, you may query for any type of DNS record, and transfer the entire zone information for a domain.

When invoked without argument, `nslookup` will display the name server it uses, and enter interactive mode. At the '`>`' prompt, you may type any domain name it should query for. By default, it asks for class `A` records, those containing the IP address relating to the domain name. You may change this type by issuing "`set type=`*type*", where *type* is one of the resource record names described above in section 3.10, or `ANY`.

For example, you might have the following dialogue with it:

```
;
; /etc/domain/named.local       Local hosts at the brewery
;                               Origin is linus.lxnet.org
;
@                   IN  SOA   vlager.linus.lxnet.org. (
                              janet.linus.lxnet.org.
                              16         ; serial
                              86400      ; refresh: once per day
                              3600       ; retry:   one hour
                              3600000    ; expire:  42 days
                              604800     ; minimum: 1 week
                              )
                    IN  NS    vlager.linus.lxnet.org.
;
; local mail is distributed on vlager
                    IN  MX    10 vlager
;
; loopback address
localhost.          IN  A     127.0.0.1
; brewery Ethernet
vlager              IN  A     192.72.1.1
; vlager is also news server
news                IN  CNAME vlager
vstout              IN  A     192.72.1.2
vale                IN  A     192.72.1.3
; winery Ethernet
vbeaujolais         IN  A     192.72.2.1
vbardolino          IN  A     192.72.2.2
vchianti            IN  A     192.72.2.3
```

Figure 3.3: The `named.hosts` file.

```
;
; /etc/domain/named.local       Reverse mapping of 127.0.0
;                               Origin is 0.0.127.in-addr.arpa.
;
@                   IN  SOA   vlager.linus.lxnet.org. (
                              joe.linus.lxnet.org.
                              1          ; serial
                              360000     ; refresh: 100 hrs
                              3600       ; retry:   one hour
                              3600000    ; expire:  42 days
                              360000     ; minimum: 100 hrs
                              )
                    IN  NS    vlager.linus.lxnet.org.
1                   IN  PTR   localhost.
```

Figure 3.4: The named.local file.

```
;
; /etc/domain/named.rev         Reverse mapping of our IP addresses
;                               Origin is 72.192.in-addr.arpa.
;
@                   IN  SOA   vlager.linus.lxnet.org. (
                              joe.linus.lxnet.org.
                              16         ; serial
                              86400      ; refresh: once per day
                              3600       ; retry:   one hour
                              3600000    ; expire:  42 days
                              604800     ; minimum: 1 week
                              )
                    IN  NS    vlager.linus.lxnet.org.
; brewery
1.1                 IN  PTR   vlager.linus.lxnet.org.
1.2                 IN  PTR   vstout.linus.lxnet.org.
1.3                 IN  PTR   vale.linus.lxnet.org.
; winery
2.1                 IN  PTR   vbeaujolais.linus.lxnet.org.
2.2                 IN  PTR   vbardolino.linus.lxnet.org.
2.3                 IN  PTR   vchianti.linus.lxnet.org.
```

Figure 3.5: The named.rev file.

```
$ nslookup
Default Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

> sunsite.unc.edu
Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

Non-authoritative answer:
Name:    sunsite.unc.edu
Address:  152.2.22.81

> unc.edu
*** No address (A) records available for unc.edu
Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

> set type=MX
> unc.edu
Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

Non-authoritative answer:
unc.edu preference = 10, mail exchanger = lambada.oit.unc.edu

Authoritative answers can be found from:
UNC.EDU nameserver = SAMBA.ACS.UNC.EDU
UNC.EDU nameserver = NS.BME.UNC.EDU
UNC.EDU nameserver = NCNOC.CONCERT.NET
lambada.oit.unc.edu     internet address = 152.2.22.80
SAMBA.ACS.UNC.EDU       internet address = 128.109.157.30
NS.BME.UNC.EDU          internet address = 128.109.155.1
NS.BME.UNC.EDU          internet address = 152.2.100.1
NCNOC.CONCERT.NET       internet address = 192.101.21.1
NCNOC.CONCERT.NET       internet address = 128.109.193.1
> set type=SOA
> unc.edu
Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

Non-authoritative answer:
unc.edu
        origin = ns.unc.edu
        mail addr = shava.ns.unc.edu
```

```
        serial = 930408
        refresh = 28800 (8 hours)
        retry   = 3600 (1 hour)
        expire  = 1209600 (14 days)
        minimum ttl = 86400 (1 day)

Authoritative answers can be found from:
[same as above]
> exit
$
```

The complete set of commands available with `nslookup` may be obtained by the `help` command.

Chapter 4

Various Network Applications

After successfully setting up IP and the resolver, you have to turn to the services you want to provide over the network. This chapter lists the configuration options and requirements for several services. Some, however, require more space, and are thus explained in a separate chapter. This certainly applies to electronic mail and netnews, which are also not specific to TCP/IP networking.

There are other network applications as well, which only need little attendance when setting them up; please refer to their respective manual pages for details.

Generally, services are performed by *daemons*. A daemon is a program that opens a certain port, and waits for incoming connections. If one occurs, it creates a child process which accepts the connection, while the parent continues to listen for further requests. The daemon providing the service is generally called the *server*, and the program requesting it is called the *client*.

Support of NIS ("Network Information System", a.k.a. "Yellow Pages") is under way. Once it is released, I will add a section on its configuration.

4.1 The `/etc/services` file

As noted before in section 1.3.4, the port numbers on which certain "standard" services are offered are laid down in the "Assigned Numbers" RFC. To enable server and client programs to convert service names to these numbers, at least a part of the list is kept on each host; it is stored in a file called `/etc/services`.

It contains single lines, each describing a service. An entry is made like this:

service port/proto [*aliases*]

Here, *service* specifies the service name, *port* and *proto* define the port the service is offered on, and which transport protocol is used. Commonly, this is either `udp` or `tcp`. It is possible for a service to be offered for more than one protocol, as well as offering different services on the same port, as long as the protocols are different. The *aliases* field allows to specify alternative names for the same service.

Usually, you don't have to change the services file that comes along with the network software on your LINUX system. Nevertheless, we will give a small excerpt from the file in figure 4.1.

Note that, for example, the `echo` service is offered on port 7 for both TCP and UDP, and that port 512 is used for two different services, namely the COMSAT daemon (which notifies users of newly arrived mail, see `xbiff(1x)`), using UDP, and for remote execution (`rexec(1)`), using TCP.

4.2 The `/etc/protocols` file

Similar to the services file, the networking library needs a way to translate protocol names — for example, those used in the services file — to protocol numbers understood by the IP layer on other hosts. This is done by looking up the name in the `/etc/protocols` file. It contains one entry per line, each containing a protocol name, and the associated number. Having to touch this file is even more unlikely than having to muddle with `/etc/services`. A sample file is given in figure 4.2.

4.3 The `inetd` Super-Server

The concept of servers and clients described above has one drawback: for every service offered, a daemon has to run that listens on the port for a connection to occur. Since, however, on many installation not all services are required at all times, this means a waste of system resources (swap space).

Thus, almost all Un⋆x installations run a "super-server" that creates sockets for a number of services, and listens on all of them simultaneously. When a remote host requests one of the services, the super-server notices this and spawns the server specified for this port.

The super-server commonly used is `inetd`, the Internet Daemon. It is started at system boot time, and takes the list of services it is to manage from a startup file named `/etc/inetd.conf`. In addition to those servers invoked, there are a number of trivial services which are performed by `inetd` itself, these are called *internal services*. They include

```
# The services file:
#
# well-known services
echo           7/tcp                  # Echo
echo           7/udp                  #
discard        9/tcp  sink null       # Discard
discard        9/udp  sink null       #
systat        11/tcp  users           # Active Users
daytime       13/tcp                  # Daytime
daytime       13/udp                  #
chargen       19/tcp  ttytst source  # Character Generator
chargen       19/udp  ttytst source  #
ftp-data      20/tcp                  # File Transfer Protocol (Data)
ftp           21/tcp                  # File Transfer Protocol (Control)
telnet        23/tcp                  # Virtual Terminal Protocol
smtp          25/tcp                  # Simple Mail Transfer Protocol
domain        53/tcp  nameserver      # Domain Name Service
domain        53/udp  nameserver      #
nntp         119/tcp  readnews        # Network News Transfer Protocol
#
# UNIX services
biff         512/udp  comsat          # mail notification
exec         512/tcp                  # remote execution, passwd required
login        513/tcp                  # remote login
who          513/udp  whod            # remote who and uptime
shell        514/tcp  cmd             # remote command, no passwd used
syslog       514/udp                  # remote system logging
printer      515/tcp  spooler         # remote print spooling
talk         517/udp                  # conversation
ntalk        518/udp                  # new talk, conversation
route        520/udp  router routed  # routing information protocol
```

Figure 4.1: A sample /etc/services file.

```
#
# Internet (IP) protocols
#
ip      0       IP              # internet protocol, pseudo protocol number
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # internet group multicast protocol
ggp     3       GGP             # gateway-gateway protocol
tcp     6       TCP             # transmission control protocol
egp     8       EGP             # exterior gateway protocol
pup     12      PUP             # PARC universal packet protocol
udp     17      UDP             # user datagram protocol
raw     255     RAW             # RAW IP interface
```

Figure 4.2: A sample `/etc/protocols` file.

`chargen` which simply generates a string of characters, `daytime` which returns the system's idea of the time of day.

An entry in this file consists of a single line in the following format:

service type proto wait user server cmdline

The meaning of the fields is as follows:

service Gives a service name as specified in `/etc/services`.

type Specifies a socket type, either `stream` (for connection-oriented protocols) or `dgram` (for datagram protocols). TCP-based services should therefore always use `stream`, while UDP-based services should always use `dgram`.

proto The transport protocol used by the service. Must be a valid protocol name found in `/etc/protocols`.

wait This option only applies to `dgram` sockets. It may be either `wait` or `nowait`. If `wait` is specified, `inetd` will only execute one server for the specified port at any time. Otherwise, it will immediately continue to listen on the port after executing the server.

This is useful for "single-threaded" servers that read all incoming datagrams until no more arrive, and then exit. Most RPC servers are of this type and should therefore specify `wait`. The opposite type, "multi-threaded" servers, allow an unlimited number of instances to run concurrently; `named` is of this type. These servers should specify `nowait`.

`stream` sockets should always use `nowait`.

user This is the login id of the user the process is executed under. Generally, this will be `root`, but some services may use different accounts. For example, the NNTP news server will run as `news`, while services that may pose a security threat (such as `tftp` or `finger`) are often run as `nobody`.

server This gives the full path name of the server program to be executed. Internal services are marked by the keyword `internal`.

cmdline This is the command line to be passed to the server. This includes argument 0, that is the command name. Usually, this will be the program name of the server, unless the program behaves differently when invoked by a different name.

This field is empty for internal services.

A sample `inetd.conf` file is shown in figure 4.3. We see that `tftp` and `finger` are commented out, so that they are not available. This is often done for security reasons, because these services are prime targets for intruders.

4.4 The `tcpd` access control facility

Since opening a computer to network access involves many security risks, applications are designed to guard against several types of attacks. Some of these, however, may be flawed (most drastically demonstrated by the RTM Internet worm), or do not allow to differentiate between secure hosts from which requests for a particular service will be accepted, and insecure hosts whose requests should be rejected. The `finger` and `tftp` services, for example, pose such a risk because they allow potential intruders to gather login names from users on the system. Thus, one would want to limit access to these services to "trusted hosts" only, which is impossible with the usual setup, where `inetd` either provides this service to all clients, or not at all.

The NET-2 release contains `tcpd`,[1] a so-called daemon wrapper. For TCP services you want to monitor or protect, it is invoked instead of the server program, logs the request to the `syslog` daemon, performs optional host access control, and only if this succeeds, executes the real server program. Note that is does not work with UDP-based services, most notably not RPC.

[1]Written by Wietse Venema, `wietse@wzv.win.tue.nl`.

```
#
# inetd services
ftp          stream tcp nowait root /etc/ftpd     ftpd -l
telnet       stream tcp nowait root /etc/telnetd  telnetd -b/etc/issue
#tftp         dgram  udp wait   root /etc/tftpd    tftpd
#finger       stream tcp nowait bin /etc/fingerd  fingerd
login        stream tcp nowait root /etc/rlogind  rlogind
shell        stream tcp nowait root /etc/remshd   remshd
exec         stream tcp nowait root /etc/rexecd   rexecd
#
#       inetd internal services
#
daytime      stream tcp nowait root internal
daytime      dgram  udp nowait root internal
time         stream tcp nowait root internal
time         dgram  udp nowait root internal
echo         stream tcp nowait root internal
echo         dgram  udp nowait root internal
discard      stream tcp nowait root internal
discard      dgram  udp nowait root internal
chargen      stream tcp nowait root internal
chargen      dgram  udp nowait root internal
```

Figure 4.3: A sample /etc/inetd.conf file.

For example, to wrap the `finger` daemon, you have to change the corresponding line in `inetd.conf` to

```
# wrap finger daemon
finger  stream  tcp    nowait  root    /usr/etc/tcpd   in.fingerd
```

Without adding any access control, this will appear to the client just as a usual `finger` setup, except that any requests are logged to `syslog`.

Access control is implemented by means of two files located in the `/etc` directory; these are `hosts.allow` and `hosts.deny`. They contain entries allowing and denying access, respectively, to certain services and hosts. If `tcpd` has to handle a request for a service, say `finger`, from a client host named `biff.foobar.com`, it scans `hosts.allow` and `hosts.deny` (in this order) for an entry matching both the service and client host. If a matching entry is found in `hosts.allow`, access is granted, regardless of any entry in `hosts.deny`. If a match is found in `hosts.deny`, the request is rejected by closing down the connection. If no match is found at all, the request is accepted, too.

Entries in the access files look like this:

servicelist: *hostlist* [:*shellcmd*]

servicelist is a list of service names from `/etc/services`, or the keyword `ALL`. To match all services except `finger`, use "`ALL EXCEPT finger`".

hostlist is a list of hostnames or IP addresses, or the keywords `ALL`, `LOCAL`, or `UNKNOWN`. `ALL` matches any host, while `LOCAL` matches hostnames not containing a dot.[2] `UNKNOWN` matches any hosts whose name or address lookup failed. A name starting with a dot matches all hosts whose domain is equal to this name. For example, `.foobar.com` matches `biff.foobar.com`. There are also provisions for IP network addresses and subnet numbers. Please refer to the `hosts_access(5)` manual page for details.

To deny access to the `finger` and `tfpd` services to all but the local hosts, put the following in `/etc/hosts.deny`, and leave `/etc/hosts.allow` empty:

```
in.ftpd, in.fingerd: ALL EXCEPT LOCAL, .your.domain
```

The optional *shellcmd* field may contain a shell command to be invoked when the entry is matched. This is useful to set up traps that may expose the attacker:

[2] Usually local hostnames obtained from lookups in `/etc/hosts`.

```
in.ftpd: ALL EXCEPT LOCAL, .your.domain : \
        finger -l @%h | /usr/bin/mail -s %d-%h root
```

The `%h` and `%d` arguments are expanded by `tcpd` to the client host name and service name, respectively. Please refer to the `hosts_access(5)` manual page for details.

4.5 Configuring the r commands

There are a number of commands for executing commands on remote hosts. These are `rlogin`, `rsh`, `rcp` and `rcmd`. They all spawn a shell on the remote host and allow the user to execute commands. Of course, the client needs to have an account on the host where the commmand is to be executed, thus all these commands perform an authorization procedure. Usually, the client will tell the user's login name to the server, which in turn requests a password that is validated in the usual way.

However, for reasons of convenience, you may facilitate access to these services within a network by overriding authorization checks. However, this is only advisable for a small number of hosts whose password databases are synchronized, or for a small number of privileged users who need to access many machines for administrative reasons. Whenever you want to allow people to log into your host without having to specify a login id or password, make sure that you don't accidentally grant access to anybody else.

There are two ways to disable authorization checks for the `r` commands. One is for the super user to allow certain or all users on certain or all hosts (the latter definitely being a bad idea) to log in without being asked for a password. This access is controlled by a file called `/etc/hosts.equiv`, which defines a list of *equivalent* hosts and users. The second option is for a user to grant other users on certain hosts access to her account. These may be listed in the file `.rhosts` in the user's home directory. If this file is not owned by the user or the super user, or if it is a symbolic link, it is ignored.[3]

When a client requests an `r` service, her host and user name are searched in the `/etc/hosts.equiv` file, and then in the home directory of the local user with the same name as the client, if one exists. Thus, if `janet` executes

```
$ rlogin -l joe euler
```

on `gauss` to login as `joe` on `euler`, the server will first search `/etc/hosts.equiv`[4] for

[3] In an NFS environment, you may need to give it a protection of `444`, because the super user is often very restricted in accessing files on disks mounted via NFS.

[4] Note that the `hosts.equiv` file is *not* searched when someone attempts to log in as `root`.

an entry matching `gauss` and `janet`, and if this fails, try to look it up in `.rhosts` in `joe`'s home directory.

Both files contain lines that describe accounts on remote hosts. They may be used to allow or deny access to these users. However, their interpretation of the entries differ slightly.

An entry consists of a hostname, optionally followed by a login name. Each hostname in turn is compared to the client's canonical hostname. If it matches, and if the entry contains a user name, it is compared to the client's user name (on the system she is calling from). If the entry does not give a user name, the client's name is checked against the login name she tries to log in as.

Note that the client's hostname is obtained by reverse mapping the address to a name, so that this feature will fail with hosts unknown to the resolver. The client's hostname is considered to match the name in the hosts files in one of the following cases:

- The client's canonical hostname (not an alias) literally matches the hostname in the file.
- If the client's hostname is a fully qualified domain name (such as returned by the resolver when you have DNS running), and it doesn't literally match the hostname in the hosts file, it is compared to that hostname expanded with the local domain name.

Now, let's look what happens Janet's attempt to log into Joe's account on `euler`. Suppose the `/etc/hosts.equiv` file contains the following line

```
# Allow access to users from gauss
gauss
```

This entry allows users on `gauss` to log into `euler` without specifying a password as long as they log into their *own* account.[5] That is, Janet would be allowed to log in as `janet`, but *not* as `joe`. No luck so far. Now, assume that the `.rhosts` file in `joe`'s home directory contains

```
# Allow janet to log in from gauss
gauss   janet
```

Finally, this entry matches, and `rlogind` lets Janet log in as Joe without asking for a password.

[5] Except for `root`.

4.6 Configuring RPC

RPC, or *remote procedure call*, is a service that allows clients to access a network-based service the same way they call a subroutine. It was originally developed by Sun Mircosystems, Inc., but is now present on most Un⋆x systems.[6] A RPC server registers a number of routines with the so-called *portmapper*, which may be accessed over TCP/IP, with data being passed in a system-independent fashion. The protocol it uses is called the *external data representation* (XDR) protocol.[7]

The transport protocol used by most RPC services is the user datagram protocol, because most requests are simple queries and replies, with only small amounts of data being transferred. However, some services also use TCP when transferring larger amounts of data that do not fit into a single datagram.

Services that use RPC are the network filesystem, NFS, and the network information system, NIS.[8] The naming convention used throughout the NET-2 release has it that RPC-based server programs reside in `/usr/etc`, their name being prefixed with "`rpc.`". In this section, we will use the unadorned name (like `nfsd`) and supply the prefix only when using the full path name (as in `/usr/etc/rpc.nfsd`).

Each server is assigned a program number, and each procedure it offers is assigned a procedure number. The program number is registered with the portmapper, together with the port the server listens to for requests. (Thus the name portmapper, because it maps program numbers to port numbers). Again, the program numbers for a list of standard services may be found in the "Assigned Numbers" RFC. When a client wants to access a procedure, it first inquires the portmapper on the remote host for the port it should direct its request at, and then contacts the server. Alternatively, it may ask the portmapper to forward its request to the appropriate server.

The portmapper is a program started at system boot time, namely `/etc/rpc.portmap`. It must be invoked before the `inetd` super server, because some RPC servers are managed by `inetd`.

The portmapper does not require any configuration files, because any RPC servers will register themselves with `portmap` once they start up. The advantage of this is that you may write distributed applications that use RPC without having to have access to a configuration file. The drawback is that in case the portmapper should crash, all servers have to be restarted.

However, there is an RPC configuration file that may be used by some clients to find

[6] For a specification of RPC, see RFC 1057.

[7] XDR is a good example of a session layer protocol as described in section 1.3.5.

[8] This is a successor to the *yellow pages*, or yp, service. However, yp is often used to refer to NIS now.

the program number corresponding to a certain service. This is the `/etc/rpc` file, which is very similar in function to the `/etc/services` file. Each line has the following information:

name program_number [*aliases*]

where *name* is the service name, *program_number* is the service's program number, and *aliases* is an optional list of alternative names for the service. An example file is shown in figure 4.4.

```
#
# rpc 88/08/01 4.0 RPCSRC; from 1.12   88/02/07 SMI
#
portmapper      100000  portmap sunrpc
rstatd          100001  rstat rstat_svc rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
yppasswdd       100009  yppasswd
bootparam       100026
ypupdated       100028  ypupdate
```

Figure 4.4: A sample `/etc/rpc` file.

4.7 Configuring NIS

Currently (as of July 2, 1993), there is no NIS support publicly available. Someone is working on this, however.

4.8 Configuring FTP

The most widely known use of FTP (File Transfer Protocol) is to provide users from any host with access to publicly available data. This is generally called anonymous FTP. However, it may be used for file transfers from and to local user accounts as well. The FTP server, `ftpd`, performs the usual authorization procedure using the `/etc/passwd` database.[9] Accounts

[9] The `fptd` in SLS was compiled to use the `/etc/shadow` file.

without password are ignored.

After passing this procedure, a client is allowed access to any files and directories accessible to the user she logged in as. Certain users may be denied access to FTP by listing them in the file `/etc/ftpusers`. This file contains one user name per line, without any white space allowed.

I am aware that there is another FTP daemon called `ftpd-diku`, which has enhanced functionality.[10] Since it is not part of the "official" networking release, I do not describe it here. Please refer to the documentation that comes with it.

4.8.1 Anonymous FTP

Anonymous FTP requires tough security measures to prevent clients from accessing any important information on your system.

To set up an anonymous FTP account, you have to create a user named `ftp`. It should be member of a group with very limited access only, for example `guest`, have a login shell of `/bin/false`, and a password of "`*`" to prevent anyone from actually logging in as `ftp`. The home directory should be something like `/usr/ftp`, have proper ownership (namely `ftp` and `guest`) and have a protection mode of 555.

If a client specifies a user name of `ftp` or `anonymous` when asked by `ftpd`, the server performs a `chroot(2)` to the home directory of the `ftp` account. The effect of this is that this home directory appears to the client as the root of the directory hierarchy, thus denying it access to any files outside the home directory.[11] Any files (such as the `ls` command) accessed after this `chroot` must thus be available beneath this directory.

In addition, you may allow sub-logins of users whom you have granted special access to your FTP area. These first log in as an anonymous FTP user, and after that access the sub-login by specifying a user name and password using the `USER` command of `ftp`. For these, you have to set up a home directory, which is owned by the user and has mode 700. Remember that the home directory's name has the `~ftp` directory as its root.

The following files directories need to be present:

`~ftp/bin` This directory should be owned by `root` and have mode 555. It should contain a copy of the `ls` command, which is needed by `ftpd` to list directories.

[10] Some people were wondering what the `/etc/ftpaccess` file in the SLS release is good for. It is used by `ftpd-diku` only.

[11] When you merely change the working directory, you may access files outside of it by either using absolute pathnames or '`..`' to access files not below the working directory. When doing a `chroot`, all absolute pathnames are interpreted as starting from the new directory, and its parent directory is inaccessible.

It should be mode `111`. If `~ftp` is on the same file system as `/bin`, this may also be a hard link to `/bin/ls`; however, you can not make a symbolic link.

`~ftp/lib` This directory should be owned by `root` and have mode `555`. It must contain a copy of the current version of the `libc` jumptable image, `/lib/libc.so.`*X*. This is needed for `ls`, which is linked with the jumptable libraries. As with `~ftp/bin/ls`, this may be a hard link to the jumptable image, if possible, but not a symbolic link. It should be owned by `root` and have mode `111`.

`~ftp/etc` This directory should be owned by `root` and have mode `555`. It should contain versions of the `passwd`, `group`, and `shadow` files from `/etc`. These are needed to map user id's to login names when listing files. They should not be complete copies of the files from `/etc`, nor should they be hard links, since this would provide intruders with a list of all accounts on your system.

Accounts other than sub-logins should be denied access by specifying a password of "`!`" in `~ftp/etc/shadow`.

`~ftp/incoming`
This directory should be owned by `ftp`. It either should have mode `733` to allow users to deposit files, but preventing others to retrieve them before you have moved them to the proper location (or thrown them away, for that matter), or should have mode `777` if you want to allow them to be retrieved.

`~ftp/pub` Under this directory, you keep all files you want to make available to users of your anonymous FTP service. The directory should be owned by `ftp` and have mode `555`. Files below this should have mode `444` to prevent FTP users to remove or modify them.

If you allow sub-logins of people who have taken over responsibility for part of your FTP area, you should make sure to set permissions properly for these directories.

If your `~ftp/` directory is not on the same partition as your shared libraries, you will almost assuredly be better off to make staticly linked copies of your executables, as the shared library is quite large, and would take up more disk space than simply making a few staticly-linked executables. Another possibility is installing the "lite" shared library, which is available in the GCC/library distribution, and is quite a bit smaller.

4.8.2 `ftpd` Options

Usually, `ftpd` is invoked by the `inetd` super-server. On the command line passed to `ftpd`, you may specify the following options:

`-l` Enable logging of FTP sessions.

`-d` *dbglvl* Enable logging of debug messages.

`-t` *timeout* This causes `ftpd` to close down sessions after the client has been inactive for the specified amount of time. By default, this value is set to 15 minutes.

`-u` *umask* Sets the file creation mask used by `ftpd` to *umask*. It defaults to `027`.

When logging is enabled, `ftpd` logs messages to the `daemon` facility of `syslogd`. FTP sessions are logged at the `info` level; debug messages are logged at the `debug` level. To send these messages to a file, enter the following line to `/etc/syslogd.conf`:

```
daemon.info:    /usr/adm/daemon.log
```

Chapter 5

Configuring NFS

NFS, the network filesystem, is one of the most prominent network services using RPC. Unlike conventional file systems that write blocks to and read them from physical media, NFS does not use any special device, or deals with block allocation, file system consistency, and the like. Instead, the basic unit of information with NFS is a file. This file access is completely transparent to the client, and works across a variety of server and host architectures.

This offers a number of advantages:

- Data accessed by all users can be kept on a central host, with clients mounting this directory at boot time. For example, this may be the disk containing `/home`, so that users don't have to remember the host where they keep the most up-to-date version of their current project.

- Data consuming large amounts of disk space may be kept on a single host. For example, all files and program relating to LaTeX and METAFONT could be kept and maintained in one place, reducing administrative overhead.

- Administrative data may be kept on a single host. No need to use `rcp` anymore to install the same stupid file on 20 different machines.

However, before you start to weep with joy, let me tell you that the Linux implementation of NFS is not yet perfect. Although it is now possible to run binaries from NFS-mounted file systems, it is still a bit slow. This is due to the fact that the kernel is currently lacking a generalized form of the `mmap(2)` call, which allows a program to map files to a process' address space.

Let's have a look now at how NFS works: A client may request to mount a directory from a remote host on a local directory, just the same way it does with physical devices.

However, the syntax used to specify the remote directory is different. For example, to mount `/home` from host `vlager` to `/users` on `vale`, the administrator would issue the following command on `vale`:[1]

```
# mount -t nfs vlager:/home /users
```

`mount` will then try to connect to the `mountd` mount daemon on `vlager` via RPC. The server will check if `vale` is permitted to mount the directory in question, and if so, return it a file handle. This file handle will be used in all subsequent requests to files below `/users`.

When accessing files over NFS, an RPC call is made to `nfsd` (the NFS daemon) on the server machine, in which `nfsd` is passed the file handle, the name of the file to be accessed, and the user's user and group id. These are used in determining access rights to the specified file. In order to prevent unauthorized users from reading or modifying files, user and group ids on both hosts must be synchronized.

On most Un⋆x implementations, the RPC call is made by a daemon running on the client host. This is the *Block I/O Daemon*, `biod`. It is used to improve throughput by performing asynchronous I/O (read-ahead and write-behind). LINUX NFS, however, does not use `biod`; the kernel NFS layer places the RPC call itself instead. It is (currently) lacking read-ahead and write-behind, so that performance comparisons with other NFS clients are not favorable.

The biggest problem with the LINUX NFS code is that the LINUX UDP driver currently does not support datagram fragmentation, and that it has a datagram size limit of 1K. This means that for NFS mounts from systems that use large UDP datagrams (e.g., 8K on SunOS) by default, it needs to be downsized artificially. This, of course, further decreases transfer speed due to the increase in the RPC-imposed overhead.

The syntax for reducing both read and write transfer sizes to 1K varies among different operating systems, but to mount a volume from your LINUX machine to SunOS or a BSD-derived system you can use mount options like this:

```
# mount -t nfs linux-host:remote-dir local-dir -o rsize=1024,wsize=1024
```

5.1 Mounting an NFS Volume

NFS volumes[2] are mounted very much the way usual file systems are mounted. You invoke `mount` using the following syntax

[1] Note that you can omit the `-t nfs` argument, because `mount` sees from the colon that this specifies an NFS volume.

[2] One doesn't say file system, because these are not proper file systems.

```
# mount -t nfs nfs_volume local_dir options
```

nfs_volume is given as *remote_host*:*remote_dir*. Since this notation is unique to NFS file systems, you can leave out the `-t nfs` option.

There are a number of additional options that you may specify to `mount` upon mounting an NFS volume. These may either be given following the `-o` switch on the command line, or in the options field of the `/etc/fstab` entry for the volume. In both cases, multiple options are separated from each other by commas.

A sample entry in `/etc/fstab` might be

```
news:/usr/spool/news  /usr/spool/news   nfs   timeo=14,intr
```

This volume may then be mounted using

```
mount news:/usr/spool/news
```

An alternative command to be used in the absence of a `fstab` entry would be

```
mount -t nfs news:/usr/spool/news /usr/spool/news -o timeo=14,intr
```

There are a number of options that control the client's behavior when mounting an NFS volume, as well as options that configure the way the server treats mount requests for a certain directory.

Server options are given to the `nfsd` daemon in a file called `exports`, which is described in the following section. Client options are given to the `mount` command in one of the two ways described above, i.e. either on the command line using the `-o` switch, or in the `fstab` file. Options specified on the command line always override those given in the `fstab` file.

The list of all valid options is described in its entirety in the `nfs(5)` manual page that comes with Rick Sladkey's NFS server distribution. The following is an incomplete list of those you would probably want to use:

`rsize=`*n* and `wsize=`*n*
: These specify the datagram size used by the NFS clients on read and write requests, respectively. They currently default to 1024 bytes, due to the limit on UDP datagram size described above.

`timeo=`*n* This sets the time (in thenths of a second) the NFS client will wait for a request to complete. The default values is 0.7 seconds. If no confirmation is received within this time, the client will retry the operation with the timeout interval doubled. After reaching a maximum timeout of 60 seconds, a *major* timeout occurs. By default, a major timeout will cause the client to print a message to the console and start all over again, this time with an initial timeout interval twice that of the previous cascade. Potentially, this may go on forever. Volumes that show this kind of behaviour are called *hard-mounted*.

`hard` Explicitly mark this volume as hard-mounted. This is on by default.

`soft` Soft-mount the driver (as opposed to hard-mount). This causes the client to return an I/O error to the calling process whenever a major timeout occurs.

`intr` Allow signals to interrupt an NFS call. Useful for aborting when the server doesn't respond.

Whether you hard- or soft-mount a volume is not simply a question of taste, but also has to do what sort of information you want to access from this volume. For example, if you mount your X11 programs by NFS, you sure would not want you X session to go berserk just because someone brought the network to a grinding halt by firing up seven copies of `xv` at the same time, or by pulling out the Ethernet plug for a moment. By hard-mounting these, you make sure that you computer will wait until it is able to re-establish contact with you NFS-server. On the other hand, any data that is not that critical may as well be soft-mounted, so it doesn't hang you session in case the remote machine should be temporarily unreachable, or down. In case your network connection to the server is flakey, you may either increase the initial timeout using the `timeo` option, or hard-mount the volumes, but allow for signals interrupting the NFS call so that you may still abort any hanging file access.

5.2 The `/etc/exports` File

By default, `mountd` will not allow anyone to mount direcories from the local host. The directories that may be mounted by remote hosts must be defined in a file called `/etc/exports` (because `mountd` views this as "exporting" them to the client). Entries take the following form:

directory host`[(`*flag*`,...)]` *host* `[(`*flag*`,...)]` ...

Each line defines a directory, and the hosts allowed to mount it. A host name is usually a fully qualified domain name, but may additionally contain the '*' wildcard. For example, `lab*.foo.com` will apply to `lab01.foo.com` as well as `lab02.foo.com`. An optional list specifies the flags pertaining to it. Blank lines are ignored, and a # introduces a comment to the end of the line.

The NFS daemon honors a number of flags:

`insecure` Permit non-authenticated access from this machine.

`unix-rpc` Require UNIX-domain RPC authentication from this machine. This simply requires that requests originate from a reserved internet port (i.e. the port number has to be less than 1024). This option is on by default.

`root_squash` This is a security feature that denies the super user on the specified hosts any special access rights by mapping requests from uid 0 on the client to uid 65534 (-2) on the server. This uid should be associated with the user `nobody`.

`no_root_squash`
Don't map requests from uid 0. This option is on by default.

`ro` Mount file hierarchy read-only. This option is on by default.

`rw` Mount file hierarchy read-write.

`link_relative`
Convert absolute symbolic links (where the link contents start with a slash) into relative links by prepending the necessary number of `../`'s to get from the directory containing the link to the root on the server. This has subtle, perhaps questionable, semantics when the file hierarchy is not mounted at its root.

This option is on by default.

`link_absolute`
Leave all symbolic link as they are (the normal behavior for Sun-supplied NFS servers).

`map_identity` and `map_daemon`
The ownership information of files a NFS daemon provides to its clients usually only contains numerical user and group id's. If these numerical id's have the same user and group names associated, client and server are said

to share the same uid/gid space. The `map_identity` option tells the server to assume this is the case. This option is on by default.

The `map_daemon` option tells the NFS software to assume these are not identical. This may be the case if users on the two hosts have been assigned the same user names, but numerical ids do not match. The NET-2 release includes a `ugidd(8)` daemon which performs mapping between the two uid/gid spaces. The `map_daemon` option tells `nfsd` to use this.

An error parsing the `exports` file is reported to syslogd's `DAEMON` facility at level `NOTICE` whenever `nfsd` or `mountd` is started up.

Note that host names are obtained from the client's IP address by reverse mapping, so you have to have the resolver configured properly. If you use BIND and are very security-conscious, you should use enable spoof checking in your `/etc/host.conf` file (see section 3.9.3).

5.3 The NFS daemon

Requests from client hosts to files in a directory mounted via NFS are served by the NFS daemon, `nfsd`, via RPC. You will not want to run `nfsd` on your host unless you export directories to other hosts via NFS.

The server is generally not started by `inetd`, but at system boot time. It is a single-threaded server, in that it does not create an arbitrary number of instances of itself, one per request. Instead, a single process is started at sytem boot time, which listens on the NFS port (usually port 2049) for UDP packets. Packets which cannot be served immediately because the server is busy will be queued by the kernel. Note that it is important to start `nfsd` after `portmap` is running, because it has to register its RPC services. To start and NFS server on you machine, include the following in your network boot script (`/etc/rc.d/rc.inet2`):

```
/usr/etc/rpc.nfsd
```

Chapter 6

Setting up the Serial Hardware

There are rumors that there are some people out there in netland who only own one PC and don't have the money to spend on an Internet link. To get their daily dose of news and mail nevertheless, they are said to rely on bulletin board systems (BBS's) and UUCP networks that utilize pblic telephone networks.

This chapter is intended to help all those people who rely on modems to maintain their link.

6.1 Communication Software for Modem Links

There are a number of communction packages available for LINUX. Many of them are *terminal programs* which allow a user to dial into another comupter as if she was sitting in from of a simple terminal. The traditional terminal program for Un⋆ces is `kermit`. It is, however, slightly spartanic. There are more comfortable programs available that support a dictionary of telephone numers, script languages for calling and logging into remote computer systems, etc. One of them is `minicom`, which is close to some terminal programs former DOS users might be accustomed to. There is also an X-based communications package called `seyon`.

For people that run a bulletin board system, there is also a BBS package available for LINUX. It is called XXX.

Apart from terminal programs, there is also software that uses a serial link non-interactively to transport data to or from your computer. The advantage of this technique is that it takes much less time to download a few dozen kilobyte automatically, than it might take you to read your mail on-line in some mailbox, or browse a bulletin board for interesting articles. On the other hand, this requires more disk storage because of the loads of useless information you usually get.

The prototype of this sort of communications software is UUCP. It is a program suite that allows to copy file from one host to another, execute programs on a remote host, etc. It is frequently used to transport mail or news in private networks. A port of Ian Taylor's UUCP packages to LINUX is described in the following chapter. Other non-interactive communication software is, for example, used throughout Fidonet, although I haven't heard of a LINUX port yet.

6.2 Introduction to Serial Devices

The devices a Un⋆x kernel provides for accessing serial devices are typically called *ttys*. This is an abbreviation for *Teletype*™,[1], and is nowadays used for any character-based data terminal. Throughout this chapter, we will use the term exclusively to refer to the kernel devices.

LINUX distinguishes three classes of ttys: (virtual) consoles, pseudo-terminals (similar to a two-way pipe, used by application such as X11), and serial devices. The latter are also counted as ttys, because they permit interactive sessions over a serial connection; be it from a hard-wired terminal or a remote computer over a telephone line.

Ttys have a number of configurable parameters which can be set using the `ioctl(2)` system call. Many of them only apply to serial devices, since they need a great deal more flexibility to handle varying types of connections.

Among the most prominent line parameters are the line speed, and parity. But there are also flags for the conversion between upper and lower case characters, of carriage return into line feed, etc. The tty driver may also support various *line disciplines* which make the device driver behave completely different. For example, the SLIP driver for LINUX is implemented by means of a special line discipline.

There is a bit ambiguity about how to measure a line's speed. Sometimes it is given in Baud, and therefore called the *Baud rate*. You also frequently hear the term `Bit rate`, which is related to the line's transfer speed measured in bits per second (or bps for short). These two terms are however not interchangeable. The Baud rate refers to a physical characteristic of some serial device, namely the clock rate at which it drives its port. The bit rate rather denotes a current state of an existing serial connection between two points, namely the average number of bits transferred per second. It is important to know that these two values may deviate from each other.

For example, look at two computers communicating with each other over a telephone

[1]Teletype used to be one of the major manufacturers of terminals in the early days of Unix. I don't know if it still is.

line using two modems. Then each computer communicates with its modem at a given Baud rate, as do the modems among themselves. If the modems now use a compression technique, like V.42bis, the computers may send more data over the line, i.e. the bit rate measured between them may increase by a factor of up to three. This of course requires that the computers' Baud rates are high enough to handle this.

6.3 Accessing Serial Devices

Like all devices in Un⋆x system, serial ports are also accessed through device special files, located in the `/dev` directory. Until release 0.99.5 of the LINUX kernel, all tty devices used to have major number `4`, and were named `/dev/ttyS0`, `/dev/ttyS1`, etc.[2] At 0.99.5, Theodore T'so added a second tty device type supporting modem control.[3] These modem devices have major number `5`, and are called `/dev/cua0`, etc.

You may access a serial device both as a conventional tty, as well as a modem port, depending on the major number you access it by. Identical minor numbers for both device classes refer to the same port. If you have your modem on one of `COM1` through `COM4`, its minor number will be the `COM` port number plus `63`. If your setup is different from that, for example when using a board supporting multiple serial lines, please refer to section 6.4 below.

For use with a modem, you have to use a modem device, while the 'ordinary' serial devices are for direct serial connections, for example between a terminal and a computer, or between two computers. The latter is also called a *direct* connection.

Assume your modem is on `COM2`. Thus its minor number will be `65`, while its major number will be `5` for modem functionality. There should be a device `/dev/cua0` which has these numbers. List the serial ttys in the `/dev` directory. Columns 5 and 6 should show major and minor numbers, respectively:

```
$ ls -l /dev/cua*
crw-rw-rw-   1 root     root       5,  64 Nov 30 19:31 /dev/cua0
crw-rw-rw-   1 root     root       5,  65 Nov 30 22:08 /dev/cua1
crw-rw-rw-   1 root     root       5,  66 Oct 28 11:56 /dev/cua2
crw-rw-rw-   1 root     root       5,  67 Mar 19  1992 /dev/cua3
```

If there is no such device, you will have to create one: become super-user and type

[2] Or `/dev/ttys0`, etc, for earlier SLS releases.

[3] That is, monitoring of modem control lines, as well as `ioctl(2)` calls for reading and manipulating them.

```
# mknod -m 666 /dev/cua1 c 4 65
# chown root.root /dev/cua1
```

Some people suggest making `/dev/modem` a symbolic link to your modem device, so that casual users don't have to remember the somewhat unintuitive `/dev/cua1`. This works well with most terminal-based communication programs, like `kermit` or `minicom`. In UUCP configuration files, however, you should always use the real device name, because Taylor UUCP does not seem to follow symbolic links.

6.4 Serial Hardware

LINUX currently supports a wide variety of serial boards which use the RS-232 standard.[4] RS-232 is currently the most common standard for serial communcications in the PC world. It uses a number of circuits for transmitting single bits as well as for synchronization. Additional lines may be used for signaling the presence of a carrier (used by modems), and handshake.

Although hardware handshake is optional, it is very useful. It allows either of the two stations to signal whether it is ready to receive more data, or if the other station should pause until the receiver is done processing the incoming data. The lines used for this are called "Clear to Send" (CTS) and "Ready to Send" (RTS), respectively, which accounts for the colloquial name of hardware handshake, namely "RTS/CTS".

In PCs, the RS-232 interface is usually driven by a UART chip derived from the National Semiconductor 16540 chip, or newer versions thereof, the NSC 16550 or NSC 16550A. Some brands (most notably internal modems equipped with the Rockwell chipset) also use completely different chips that have been programmed to behave as if they were 16550's.

The main difference between 16540's and 16550's that the latter have a FIFO buffer of 16 Bytes, while the other only have a 1-Byte buffer.[5] This makes them suitable for speeds up to 9600 Baud, while higher speeds require a 16550-compatible chip. Besides these chips, LINUX also supports the 8250 chip.[6]

In the defalt configuration, the kernel looks for four standard serial boards on `COM1` through `COM4`. These will be assigned minor numbers `64` through `67`, as described above.

If you want to configure your serial ports properly, you should install Ted Tso's

[4] This is the same as CCITT's V.24 standard. Well, nearly...

[5] You wouldn't say FIFO here...

[6] **What sort of a beast is this? Anybody please enlighten me.**

`setserial` command along with the `rc.serial` script.[7,8] This script should be invoked from `/etc/rc` at system boot time. It uses `setserial` to configure the kernel serial devices. A typical `rc.serial` script looks like this:

```
# /etc/rc.d/rc.serial - serial line configuration script.
#
# Do wild interrupt detection
/etc/setserial -W /dev/cua*

# Configure serial devices
/etc/setserial /dev/cua0 auto_irq skip_test autoconfig
/etc/setserial /dev/cua1 auto_irq skip_test autoconfig
/etc/setserial /dev/cua2 auto_irq skip_test autoconfig
/etc/setserial /dev/cua3 auto_irq skip_test autoconfig

# Display serial device configuration
/etc/setserial -bg /dev/cua*
```

If your serial card is not detected, or the `setserial -bg` command shows an incorrect setting, you will have to force the configuration by explicitly supplying the correct values. Users with internal modems equipped with the Rockwell chipset are reported to experience this problem. If, for example, a the UART chip is reported to be a NSC 16450, while in fact it is NSC 16550-compatible, you have to change the configuration command for the offending port to

```
/etc/setserial /dev/cua1 auto_irq skip_test auto_config uart 16550
```

Similar options exist to force IRQ, `COM` port, and base address setting. Please refer to the `setserial(8)` manual page.

If your modem supports hardware handshake, you should make sure to enable it. Surprisingly as it is, communication progams do not attempt to enable this by default; you have to set this manually instead. This is best performed in the `rc.serial` script, using the `stty` command.

```
stty crtscts < /dev/cua1
```

[7] You best install this script in `/etc/rc.d`. Make sure to add a line to `/etc/rc` that invokes this script.

[8] To my knowledge, the `setserial` package is not yet part of the SLS release. You may have to obtain it separately from `tsx-11.mit.edu` below `/pub/linux/ALPHA/serial`.

6.5 Multiport Boards

Beside the usual four serial ports, LINUX supports a number of multiport cards. These may be configured into the kernel by editing `kernel/chr_drv/serial.c` in your kernel source directory (look into this file for which flags you might have to define), and recompiling the kernel. Note that this only means the kernel will attempt to locate them automatically at boot time. You may, however, force auto configuration after booting by using the `setserial` command. For example, if you have an AST Fourport on address `0x1A0`, you may do this by executing

```
# setserial /dev/cua4 auto_irq autoconfig
# setserial /dev/cua5 auto_irq autoconfig
# setserial /dev/cua6 auto_irq autoconfig
# setserial /dev/cua7 auto_irq autoconfig
```

Alternatively, you may force the setting of ports and IRQ by using the following commands:

```
# setserial /dev/cua4 uart 16450 port 0x1A0 irq 9 fourport
# setserial /dev/cua5 uart 16450 port 0x1A8 irq 9 fourport
# setserial /dev/cua6 uart 16450 port 0x1B0 irq 9 fourport
# setserial /dev/cua7 uart 16450 port 0x1B8 irq 9 fourport
```

A list of multiport boards and their default locations can always be found in the file `serial.c` and the documentation for the `setserial` package. Currently, the following boards are supported, listed together with the minor numbers they may be found on:

`64` `COM1` (port `0x3F8`)

`65` `COM2` (port `0x2F8`)

`66` `COM3` (port `0x3E8`)

`67` `COM4` (port `0x2E8`)

`68-71` An AST Fourport on port `0x1A0`

`72-75` Another AST Fourport on port `0x2A0`

`76-77` The third an fourth port of an Accent Async card on port `0x330`.

`78-79` Two spare ports.

`80-95` 16 ports starting at `0x100`, for any of the following boards: Usenet Serial Board II, Boca 1004, Boca 1008, Boca 2016, or Bell Technologies HUB16. You may also use a combination of up to four Boca 1004, or two Boca 1008's.

Note that the port number given is the default port for that board. This may be changed using the `setserial` command.

6.6 Setting up your Modem

Okay, there you sit, your modem hooked to your machine. By now you should have configured your serial boards according to the previous sections.

Now you may try to access your modem using `kermit` (`<Ctrl-\>` means that you hold down the Ctrl key and press \). Your input is marked like *this*:

```
$ kermit
C-Kermit 5A(188), 23 Nov 92, POSIX
Type ? or HELP for help
C-Kermit>set line /dev/cua1
C-Kermit>set speed 9600
/dev/cua1, 9600 bps
C-Kermit>connect
Connecting to /dev/cua1, speed 9600.
The escape character is Ctrl-\(ASCII 28, FS)
Type the escape character followed by C to get back,
or followed by ? to see other options.
ATZ
OK
...more chatting with the modem...
¡Ctrl-\¿C

(Back at local UNIX system)
C-Kermit>quit
$
```

Next, you may have to configure it specifically. There are a number of options you can set with your modem,[9] like whether it may answer the phone (you really should turn this off when you use the modem on the same line as your telephone `:-)`), what dialling mode to use (touch-tone or pulse), or V.42bis error correction. These options are usually set by

[9] Usually, there are some 40 switches or more that may be set, allowing three different settings on the average. This makes up for roughly 1.8 billion ways to set up your modem. The manual has 70 pages.

talking to the modem using special command sequences that are are listed in your modem's manual. I usually configure my modem using `kermit`, and write the configured values to the modem's non-volatile memory from where they are picked up when resetting it. Of course you can use any other terminal program, like `minicom`, for example.

6.7 Setting up your System for Dialing in

When setting up a serial line for use as a dialin port, you have to enable `getty` on that line. Its name stands for *get tty*.

There's a `getty` program available for LINUX from the `getty_ps` suite.[10] Some older SLS releases used to have another `getty` program that uses a somewhat different command line syntax, and does not use configuration files as described below. If you are not sure which one you have, look for `uugetty` in your `/etc` directory. If there is no such command, you are using the older version. `getty_ps` may be obtained it from `tsx-11.mit.edu` as either binary or source.

There are quite a number of parameters for tuning `getty`. I will not describe them in detail here, but merely give sample configuration files.

Assume your modem is on `/dev/cua1`. You will then have to create a file named `getty.ttyS1` in the `/etc/default` directory. For a Hayes compatible modem, it might look as follows:

```
# Sample getty configuration file for a Hayes compatible modem to allow
# incoming modem connections.

# Line to use to do initialization.
INITLINE=cua1

# Timeout to disconnect if idle...
TIMEOUT=60

# modem initialization string... Sets the modem to disable auto-answer
# format: <expect> <send> ... (chat sequence)
INIT="" \d+++\dAT\r OK\r\n ATH0\r OK\r\n AT\sM0\sE1\sQ0\sV1\sX4\sS0=0\r OK\r\n

# Waitfor string. If this sequence of characters is received over the line,
# a call is detected.
```

[10] Originally written by Paul Sutcliffe, and currently maintained by Kris Gleason (`gleasokr@rtt.colorado.edu`). As of this writing, `getty_ps` is at version 2.07b.

```
WAITFOR=RING

# This line is the connect chat sequence.  This chat sequence is performed
# after the WAITFOR string is found.  The \A character automatically sets
# the baudrate to the characters that are found, so if you get the message
# CONNECT 2400, the baud rate is set to 2400 baud.
#
# format: <expect> <send> ... (chat sequence)
CONNECT="" ATA\r CONNECT\s\A

# this line sets the time to delay before sending the login banner
DELAY=1

# Send short message instead of the whole /etc/issue file.
ISSUE=\t\tWelcome to @S!\nLogin as guest and retrieve README.\n
```

Basically, this makes `getty` initialize the modem, especially turning off auto-answer mode by setting `S0` to zero. It then waits for the phone to ring, to which Hayes compatible modems usually react by sending `RING` to the computer. `getty` will then pick up the phone and wait for the `CONNECT` string generated by the modem when the connection has been established. It is worth noting that with this setup, *autobauding* will be performed: the `CONNECT` message always includes the speed at which the connection has been established, and this value is used to set the line speed. However, this is a bit problematic with modems that perform compression, since the `CONNECT` message usually contains the *Baud* rate, while the effective line speed may be much higher. An alternative setup will be discussed below.

For direct serial lines, the setup is much simpler:

```
# Sample getty configuration file for a direct line.
# Timeout to disconnect if idle...
TIMEOUT=60

# Waitchar: do not claim the line until a character has arrived.
WAITCHAR=yes

# this line sets the time to delay before sending the login banner
DELAY=1

# Send short message instead of the whole /etc/issue file.
ISSUE=\t\tWelcome to @S!\nLogin as guest and retrieve README.\n
```

The line speed used in initializing the serial line will be used as an index into the

/etc/gettydefs file. This file is used by getty and getty to set line parameters and the like when displaying the login prompt.[11] You should make sure that if you expect your modem to operate at, say 19200bps, there has to be an entry like this:

```
# Modem line locked at 19200 Baud
19200# B19200 CS8 CRTSCTS # B19200 SANE -ISTRIP CRTSCTS#@S login: #B19200
```

Finally, to enable getty on the serial line, add the following entry to your /etc/sysvinittab file:[12]

```
# inittab entry for dialup line
s1:12:respawn:/etc/getty ttyS1 19200 vt100
```

A final word on autobauding (someone please correct me if I'm wrong): If you are using a modem which performs compression, like MNP-5 or V.42bis, you should disable autobauding, else the slow modem-computer link would throttle the connection to the same speed achieved without any compression. A better way is to lock the modem at the maximum speed that it may achieve (e.g. 9600bps for a V.42bis modem), and let the the modem slow down the serial driver using hardware handshake. Of course, this requires to enable hardware handshake with your modem.

[11]For more information, please refer to the gettydefs(5) manual page.

[12]Assuming that you are using Miquel van Smoorenburg's System V-compatible init. Other init programs may use a different format.

Chapter 7

Managing Taylor UUCP

7.1 Preliminary Remarks

UUCP was designed in the late seventies by Mike Lesk at AT&T Bell Laboratories to provide a simple dial-up network over public telephone lines. Since serial connections through modems are still the main electronic transport medium for private sites, UUCP has become a standard for networking software. Although there are many implementations running on a wide variety of hardware platforms and operating systems, they are compatible to a high degree.

The implementation of UUCP currently distributed with LINUX is Taylor UUCP Version 1.03.[1] It is included in SLS as well as Ed Carp's `mailpak`, which can be obtained separately from most LINUX FTP sites.

The purpose of this chapter is not to give you an exhaustive description of what the command line options for the UUCP commands are and what they do, but to give you an introduction on how to set up a working UUCP node. The first section gives a hopefully gentle introduction into how UUCP implements remote execution and file transfers. If you are not entirely new to UUCP, you might want to skip this and move on to section 7.3, on page 122, which explains the various files used to set up UUCP.

During the following chapter we will however assume that you are familiar with the user programs of the UUCP suite. These are `uucp` and `uux`. For a description, please refer to the on-line manual pages.

For those who don't find everything they need in this chapter, there is a very good book, "Managing UUCP and Usenet", written by Tim O'Reilly and Grace Todino.[2] I find it very

[1] Written and copyrighted by Ian Taylor, 1992.

[2] O'Reilly & Associates, Inc., *10th ed,* 1992. Email nuts@ora.com for more information, or to order.

useful.

The latest package of UUCP binaries compiled by Vince Skahan (also included in SLS) understands the new-style — a.k.a. "Taylor" — configuration files as well as BNU configuration files. If you want to use them, you should read the documentation that comes along with the package. This is a set of texinfo files that may be converted to GNU info files. However, they are quite readable without any further processing, too. Depending on popular demand, I may one day add a description of Taylor configuration options to this chapter.

7.2 Introduction

7.2.1 History

As with most software that has somehow become "standard" over the years, there is no UUCP which one would call *the* UUCP. It has undergone a steady process of evolution since the first version which was implemented in 1976. Currently, there are two major species which differ mainly in their support of hardware and their configuration. Of these, various implementations exist, each varying slightly from its siblings.

One species is the so-called "Version 2 UUCP", which dates back to a 1977 implementation by Mike Lesk, David A. Novitz, and Greg Chesson. Although it is fairly old, it is still in frequent use. Recent implementations of Version 2 provide much of the comfort of the newer UUCP species.

The second species was developed in 1983, and is commonly referred to as either BNU (Basic Networking Utilities), HoneyDanBer UUCP (being a combination of the authors' names[3]), or HDB for short. It was conceived to eliminate some of Version 2 UUCP's deficiencies, for example new transfer protocols were added, and the spool directory was split so that now there is one directory for each site you have UUCP traffic with.

Taylor UUCP Version 1.03 was released in March 1992. As distributed with LINUX, it is configured to be BNU compatible. Therefore, this document describes the setup of a UUCP node running BNU. Most of what is said throughout the document applies to BNU in general; features specific to Taylor UUCP will be marked accordingly.

Taylor UUCP 1.03 supports serial connections (either direct or via modem) and TCP/IP connections, and has drivers for protocols `g`, `e`, `f`, and `t`, which will be discussed later in this chapter.

[3] P. Honeyman, D. A. Novitz, and B. E. Redman.

7.2.2 Commands of the UUCP Suite

Besides the publicly accessible programs, `uux` and `uucp`, the UUCP suite contains a number of commands used for administrative purposes only. They are used to monitor UUCP traffic across your node, remove old log files, or compile statistics. None of these will be described. Also, Taylor UUCP 1.03 as distributed with LINUX doesn't have many of these, and they're fairly easy to understand. However, there is a third category, which comprises the actual UUCP "work horses". They are called `uucico` (where `cico` stands for copy-in copy-out), and `uuxqt`, which executes jobs sent from remote systems. Their command line options are described below.

7.2.3 Command Line Options

Below the command line options are listed as supported by `uucico`. Many may not be perfectly clear to you before reading the next section about how everything works.

`-s` *system* Call the named *system*.

`-f` Causes `uucico` to override access and retry time restrictions.

`-c` Don't log an error message when a call is not permitted at the current time.

`-S` *system* Same as `-fs` *system*.

`-r1` Start `uucico` in master mode. This is the default when `-s` or `-S` is given. All by itself, the `-r1` option causes `uucico` to try to call all systems in `Systems`, unless prohibited by call or retry time restrictions.

`-q` After calling out, do not start `uuxqt`.

`-w` After calling out, enter an endless loop accepting calls. See `-e`.

`-p` *port* Specifies a port to call out or listen to. In slave mode, this implies use of `-e`.

`-r0` Start `uucico` in slave mode. This is the default when no `-s` or `-S` is given. In slave mode, either standard input/output are assumed to be connected to a serial port, or the TCP port specified by `-p` is used.

`-l` Perform own authorization. `uucico` displays a login and password prompt to either standard output or the port specified by `-p`. Login ids and passwords are not checked against the usual `/etc/passwd` but `/usr/local/lib/uucp/passwd`.

`-e` Given this option, `uucico` enters an infinite loop waiting for connections, either on standard input, or a TCP port specified by the `-p` option. With this option, `uucico` executes its own authorization procedures.

`-I` *file* Use *file* as configuration file. This option is only available when it is compiled to use Taylor configuration files.

`-u` *login* A no-op included for compatibility with some versions of `uucpd`.

`-D` Do not detach from the controlling terminal. Normally, `uucico` detaches from the terminal and runs in the background.

`-x` *type*, `-X` *type*

Turn on debugging of the specified type. Several types may be given as a comma-separated list. The following types are valid: abnormal, chat, handshake, uucp-proto, proto, port, config, spooldir, execute, incoming, outgoing. Using `all` turns on all options.

For compatibility with other UUCP implementations, a number may be specified instead, which turns on debugging for the first *n* items in the above list.

Debugging messages will be logged to `.Admin/audit.local` below `/usr/spool/uucp`.

The following options are recognized by `uuxqt`.

`-s` *system* Only execute requests originating from *system*.

`-c` *command* Only execute requests for the given command.

`-x` *type* Same as for `uucico`.

`-I` *file* Same as for `uucico`.

7.2.4 Layout of UUCP Transfers and Remote Execution

Vital to the understanding of UUCP is the concept of *jobs*. Every transfer a user initiates with `uucp` or `uux` is called a job. It is made up of a *command* to be executed on a remote system, and a collection of *files* to be transferred between sites. One of both parts may be missing.

Since UUCP does not generally call the remote system immediately to execute a job (else you could make do with kermit), it temporarily stores the job description away. This is called *spooling*. The directory tree under which jobs are stored is therefore called the *spool directory*. It is generally located in `/usr/spool/uucp`. In addition to the job description, UUCP might have to store input files.

Likewise, incoming jobs will not be executed immediately, but only after the connection terminates.

The exact location and naming of spool files may vary, depending on some compile-time options. HDB-compatible UUCP's generally store spool files in a directory named `/usr/spool/uucp/`*site*, where *site* is the name of the remote site.

When a connection to the remote machine is established, UUCP transfers the files describing the job, plus any input files. On the remote machine, UUCP forwards or executes the job, depending on whether it is designated for another site or not.

To differentiate between important and less important jobs, UUCP associates a *grade* with each job. This is a single letter, ranging from `0` through `9`, `A` though `Z`, and `a` through `z`, in decreasing precedence. Mail is customarily spooled with grade `C`, while news are spooled with grade `N`. However, this may vary. During connections to a remote site, jobs with higher grade are transferred earlier. Grades are assigned using the `-g` flag when invoking `uucp` or `uux`.

It is also possible to disallow transfer of jobs below a given grade at certain times. This is also called the *maximum spool grade* allowed during a conversation and defaults to `z`. Note the terminological ambiguity here — a file is only transferred if it is *equal or above* the maximum spool grade.[4]

7.2.5 The inner workings of `uucico`

◇ To understand why `uucico` needs to know certain things, a quick description of how it actually connects to a remote system might be in order here.

When you execute `uucico -s` *system* from the command line, it first has to connect physically. The actions taken depend on the type of connection to open — e.g. when using a modem, it has to find a modem, and dial out. Over TCP, it has to call `gethostbyname()` to convert the name to a network address, find out which port to open, and bind the address to the corresponding socket.

After this connection has been established, an authorization procedure has to be passed. It generally consists of the remote system asking for a login name, and possibly a password. This is commonly called the *chat*. The authorization procedure is either performed by

[4] I can hear you: "He's a mathematician, right?" :-)

the usual `getty/login` suite, or — on TCP sockets — by `uucico` itself.[5] If authorization succeeds, the remote end fires up a `uucico`. The local copy of `uucico` which initiated the connection is referred to as *master*, the remote copy as *slave*.

Next follows the *handshake phase*: The master now sends its hostname, plus several flags. The hostname is usually what `hostname()` returns, although you can override this (see section 7.3.5). The slave checks this hostname for permission to log in, send and receive files, etc. The flags describe (among other things) the maximum grade of spool files to transfer. If enabled, a conversation count, or *call sequence number* check takes place here. With this feature, both sites maintain a count of successful connections, which are compared. If they do not match, the handshake fails. See section 7.3.6 on enabling this.

Finally, the two `uucico`'s try to agree on a common *transfer protocol*. This protocol governs the way files are transferred, checked for consitency, and retransmitted in case of an error. There is a need for different protocols because of the differing types of connections supported. For example, telephone lines require a "safe" protocol which is pessimistic about errors, while TCP transmission is inherently reliable and does not need this.

After the handshake is complete, the actual transmission phase begins. Both ends turn on the selected protocol driver. The drivers possibly perform a protocol-specific initialization sequence.

First, the master sends all files queued for the remote system whose spool grade is high enough. When it has finished, it informs the slave that it it is done, and that the slave may now hang up. The slave now can either agree to hang up, or take over the conversation. This is a change of roles: now the remote system becomes master, and the local one becomes slave. The new master now sends its files. When done, both `uucico`'s exchange termination messages, and close the connection.

We will not go into this in greater detail: please refer to either the sources or any good book on UUCP for this. There is also a really antique article floating around the net, written by David A. Novitz, which gives a detailed description of the UUCP protocol.

7.2.6 What UUCP needs to know

After setting up your hardware as explained in chapter 6, you have to gather the information UUCP needs to know.

First, you will have to figure out at what speed your modem and LINUX will communicate. Note that although your modem may only have a *Baud* rate of say 2400 Baud, various compressions done by your modem may result in effective transfer rate of up to 9600 Baud

[5] Taylor UUCP does it this way. There are many implementations of the `uucpd` service.

on the computer-modem link. In the following, your transfer rate will be referred to as the line's *speed*, measured in bits per seconds (bps).

Of course, if UUCP is to do anything, you will need the phone number of a system to call. Also, you will need a valid login id and possibly a password for the remote machine.[6]

You will also have to know *exactly* how to log into the system. E.g., do you have to press the `BREAK` key before the login prompt appears? Does it display `login:` or `user:`? This is necessary for composing the *chat script*, which is a recipe telling `uucico` how to log in. If you don't know, or if the usual chat script fails, try to call the system with a terminal program like `kermit`, and write down exactly what you have to do. Of course you can use any other terminal program, like `minicom`, for example, which many former DOS users might find more comfortable.

7.2.7 Site naming

In order to identify a machine in a network — be it local, in a given domain, or even world-wide — it has to have a name [7]. As long as you simply want to use UUCP for file transfers to or from sites you dial up directly, or on a local network, this name does not have to meet any standards. However, if you want to be able to send and receive mail or news, you will have to participate in a domain. Within this domain, you will have to coordinate your sitename with the domain administrator. For a discussion of domain and site naming, see chapter 8.

Note that *name* here refers to your site's *UUCP name* as opposed to your domain address. Suppose your mail address is `finn@swim.two.birds`, then your site's domain address will be `swim.two.birds`, while your UUCP host name is only `swim`. Think of UUCP sites as knowing each other on a first-name basis.

Your first task as UUCP administrator is to set your site's name. You do this by logging in as `root` and executing

```
# hostname sitename
```

A convenient (and customary) way to do this is from the `/etc/rc.local` script.

[6] If you're just going to try out UUCP, get the number of an archive site near you. Write down the login and password — they're public to make anonymous downloads possible. In most cases, they're something like `uucp/uucp` or `nuucp/uucp`.

[7] Due to limitations in some implementations of UUCP, the length of your your host name should not exceed 7 characters. The fully qualified domain name, of course, may be longer. Thus, `minas-tirith` is too long, but `minas.tirith.me` is OK.

7.3 UUCP Configuration files

All by itself, UUCP won't know anything about how to call a remote system. In contrast to a terminal program, UUCP was made to be able to do all jobs automatically. Once it is set up properly, interference by the administrator should not be necessary during daily routine.

In order to make the necessary information easily configurable, it is kept in a couple of *configuration files*. They all reside in the directory `/usr/local/lib/uucp`. Most of these files are only used when dialling out.

The Taylor UUCP package shipped with LINUX, configured to be BNU compatible, gets its information from the following files:

`Systems` This file describes all sites known to you. For each site, it specifies its name, at what times to call it, which number to dial (if any), what type of device to use, and how to log on.

`Devices` Contains entries describing each device available, together with baud rate and initialization strings.

`Dialcodes` Contains expansions for symbolic dialcodes.

`Dialers` Describes dialers used to establish a telephone connection.

`Permissions` This file controls permissions for file access and command execution on a per-system basis.

`Poll` On systems that run `uupoll` or the `uudemon.poll` script from `cron`, this file controls when systems should be polled. However, this script is currently not part of LINUX UUCP.

`Maxuuxqts` This file contains a decimal number, specifying how many copies of `uuxqt` may run at the same time.

UUCP configuration files are generally made up of one entry per line. An entry always has to begin in column 1 of the line. It may be continued across newlines with a backslash ('\'); the continuation need not begin in column 1. Fields within an entry are separated by white space (either spaces or tabs). A hash sign ('#') starts a comment, which extends to the next line feed. Lines that begin with white space and that are not continuation lines are ignored, too.

7.3.1 How to tell UUCP about other Systems — the `Systems` File

The `Systems` file describes the systems your machine knows about, one per line. It consists of the following fields:

sysname This is the UUCP name of the remote system this entry describes.

schedule This field lays out a timetable when the remote system may be called, how long to wait before retrying after a failure, and which jobs may be transferred.

device type This describes the type of device to be used for placing the call. This is a keyword naming an entry in the `Devices` file. It may be followed directly by a protocol selection, separated from it with a comma.

speed The speed at which to connect to the remote system. This is the serial line speed as described above in 7.2.6. It is measured in bits per second (bps for short).

phone Specifies the telephone number to use if the system is called via an automatic dialing unit. Otherwise, a hyphen (‘`-`’).

chat script This field contains a sequence of tokens, separated by white space. They contain text expected from and sent to the remote machine in order to log in.

Below, we will discuss each of these fields in somewhat greater detail.

Sysname This must be the name of the remote system. You should not specify an alias you invented, because `uucico` will check it against the remote system's name when it logs on.[8] A system name may appear more than once. To call this system, the entries are tried in the order given; if one fails, `uucico` moves on to the next.

schedule The *schedule* field contains a string which describes at what times it is allowed to call the remote system. This may be either due to limitations the remote host places on its services during business hours, or simply a question of telephone cost.

[8] Older Version 2 UUCP's don't broadcast their name when being called; however, newer implementations often do, and so does Taylor UUCP.

The field consists of a string made up of a *day* and a *time* subfield. *Day* may be any of `Mo, Tu, We, Th, Fr, Sa, Su` combined, or `Any, Never, Wk`. *Time* consists of a pair of clock values, separated by a dash (‘`-`’). They specify the range within calls may be placed. Any combination of these tokens is written without white space inbetween. Any number of *day/time* specifications may be grouped together with commas. For example, `MoWe0300-0730,Fr1805-2000` allows calls on Monday and Wednesdays from 3 a.m. to 7.30, and on Fridays between 18.05 and 20.00. When a time field spans midnight, say Mo1830-0600, it actually means Monday, the time between midnight and 6 a.m., and the time between 6.30 p.m. and midnight.

The special day fields `Any` and `Never` mean what they say: Calls may be placed at any or no time, respectively. `Any` may be modified by a time subfield. `Wk` is a shorthand for `MoTuWeThFr`.

Taylor UUCP also has two special schedule tokens, namely `NonPeak` and `Night`. They are shorthand for `Any2300-0800,SaSu0800-1700` and `Any1800-0700,SaSu`, respectively.

In Taylor UUCP, one can also attach a maximum spool grade to a schedule. This is done by appending a slash (‘/’) and the spool grade character to the schedule field. This determines the maximum grade of spool files to be transferred during a connection initiated by your UUCP. For example, you can use the following schedule field to allow only transfer of mail during peak hours, while news and UUCP transfers are deferred until the evening:

```
swim Any/C,NonPeak ACU 9600 ...
```

However, a caveat is in order here: First, the grade flag is not checked when a remote system calls in, so any jobs queued for the calling system will be sent.[9] It is also possible that the remote site runs some version of UUCP that ignores your request only to transfer files of certain grade, even when you call in. So, before relying on this feature, check if the neighboring site supports this. If it does, check with the administrator to attach appropriate spool grade specifications to his schedule.

You may also give a retry interval to `uucico`. This determines how long `uucico` should wait before calling this host again after a failed connection. The time is given in minutes and is offset by a semicolon. By default, Taylor UUCP uses a retry time which increases exponentially with each failure.

[9]This is probably a feature; you normally don't have to pay for incoming calls.

device type The *device type* is a name of a device specified in the `Devices` file. It may be immediately followed by a comma and a protocol specification. This is a list of letters, each naming a protocol to be offered the remote site in the order given. It is expected that the remote site selects from this list the first protocol known to it. If no protocols are specified, `uucico` chooses from a set of protocols suitable for the device.

For a discussion of protocols available, see section 7.6.3.

speed The *speed* field specifies a speed or a range of speeds at which we want to connect to the remote system. A speed is given as a decimal number, a range as two numbers separated by a dash. The entries "`-`" and `Any` match any speed.

You can also combine this with a modem class. This may be any alphabetic string preceding the speed specification. For example, some modems only support the HST protocol for high-speed connections, while others only understand V32.bis. Their greatest common denominator is 9600 bps. Assume you own both a HST and a V32.bis modem, and poll a machine being equipped with one HST modem. Then you sure would want `uucico` to use the HST modem. Therefore, your `Systems` entry for this site should contain `HST38400` in the speed field, and the HST modem would have a `Devices` entry with a speed field of `HST19200-38400`.

For TCP connections, the *speed* field should contain a dash.

phone number

If the remote system is to be reached over a telephone line, the *phone number* field contains the number the modem should dial. It may contain several tokens interpreted by `uucico`'s dialing procedure. An equals sign ('`=`') means to wait for a secondary dial tone, and a dash generates a one-second pause. Any embedded alphabetic string may be used to hide site-dependent information like area codes. Any such string is translated to a dialcode using the `Dialcodes` file.

If the remote system is connected directly, the phone number field should contain a dash ('`-`'). If this is a TCP connection, this field should contain a hostname that can be resolved by the name server, or an IP address.

chat script The *chat script* occupies the remainder of the line. It is a list of tokens, specifying strings expected and sent by the local `uucico` process. The intention is to make `uucico` wait until the remote machine sends a login prompt, then return the login id, wait for the remote system to send the password prompt, and send the password. Expect and send strings are given in alternation. Note that `uucico` automatically appends a carriage return character (‘`\r`’) to any send string. Thus, a simple chat script would look like

```
ogin: pablo ssword: catch22
```

You may note that the expect fields don't contain the whole prompts. This is to make sure that the login succeeds even if the remote system broadcasts `Login:` instead of `login:`.

However, `uucico` also allows for the case that the remote machine's `getty` needs to be reset before sending a prompt. For this, you can specify a sub-string, separated from the expect field by dashes. This is a list of send/expect pairs executed only if the main expect fails (i.e. a timeout occurs). One use of this feature is to send a `BREAK` if the remote site doesn't send a login prompt. The following example gives an allround chat script that should also work in case you have to hit return before the login appears. `""` designates the empty expect string.

```
"" \n\r\d\r\n\c ogin:-BREAK-ogin: pablo ssword: catch22
```

There are a couple of special strings and escape characters which may occur in the chat script. Note that not all characters are recognized by all BNU UUCP's. The following are legal characters in expect strings:

`""` A pair of double qoutes designates the empty string (expect nothing).

`\b` Backspace character.

`\t` Tab character.

`\N` NUL character.

`\r` Carriage return character.

`\s` Space character (‘ ’).

`\n` Newline character.

`\\` Backslash character (‘`\`’).

`\ddd` Character in octal digits `ddd`.

`\xdd` Character in hexadecimal digits `dd`.

On send strings, the following escape characters and strings are legal in addition to the above:

`EOT`	End of transmission character (‘`^D`’).
`BREAK`	End of transmission character (may not work on all systems).
`\c`	Suppress sending of carriage return at end of string.
`\d`	Delay sending for 1 second.
`\e`	Disable echo checking.
`\E`	Enable echo checking (wait for echo before continuing).
`\K`	Same as `BREAK`.
`\p`	Pause for fraction of a second.

7.3.2 Hiding dialcodes — the `Dialcodes` file

To make the phone numbers given in the `Systems` file more legible, as well as easier to maintain within large communities, you may replace parts of it with alphabetic strings. When dialing the number, the string is translated back to numbers. Common practice is to replace area codes with names.

The `Dialcodes` file is for translating these strings back into telephone numbers. An entry in this file consists of two fields: The first is the alphabetic string, the second is the phone number to substitute.

7.3.3 What devices there are — the `Devices` file

This section describes the `Devices` file. It contains information about the devices available to UUCP.

An entry consists of the following fields:

type The device type. It must be one of the following keywords:

`ACU`	The device is a modem. `ACU` stands for “automatic call unit”.
`Direct`	This specifies a direct line.
`TCP`	The connection is established via a TCP/IP socket.

It may be followed by a comma-separated list of protocols supported. It overrides those possibly specified in the `Systems` file.

port
: This is the device name. For direct serial links or modems, this will be the name of the special file in the `/dev` directory. Note that you should specify the real device file (i.e. `/dev/cua`*n*), and not a symbolic link like `/dev/modem`.

For a TCP/IP connection, either use a service name to be looked up in `/etc/services`, or a services number. The service name will usually be `uucpd` or `uucp`. If `uucico` cannot determine a service number, port 540 is used.

dialer
: This names a dialer device file. This is primarily useful for modems. For other connections, you should use a dash ('`-`'). In case of several direct connections, you may have to use this field for some sort of trickery (see 7.3.7).

speed
: The speed the device supports (direct or a phone lines only). This entry is matched against the speed field from the `Systems` file.

The string may be either a number, or two numbers separated by a dash ('`-`'), specifying a speed or a speed range in bits per second, respectively. A single dash or the keyword `Any` matches any speed.

You can also precede this with a modem class. See the explanation of the *speed* field in 7.3.1 for an example why one would want to use this.

In conjunction with a TCP connection, use a dash.

If the entry describes a direct line (type is `Direct`), you may not specify a speed range.

dialer-token pairs
: A dialer-token pair is a dialer type, followed by a token to be sent to the dialer. The token may either be `\D` or `\T`. Both denote the phone number as specified in the `Systems` file, where `\T` implies dialcode translation, and `\D` doesn't.

The dialer type is matched against an entry in the `Dialers` file.

Under normal circumstances, you will only have one modem wired to the device directly. Then, you should give only one dialer-token pair. However, if you access several modems over a switch, you will have a dialer-token pair for each. In this case, the token will be the string given to the switch in order to access the modem.

If the entry describes a direct line (type is `Direct`), this field is not used.

The device type specified in the `Systems` file is matched against the device types in the `Devices` file. The same device type may be used several times. If you have several modems connected to your machine, you will give at least one entry named `ACU` for each of them. You may even give several entries for the same modem, each at a different speed or with a different modem initialization.

7.3.4 How to dial a number — the `Dialers` file

The `Dialers` file describes the way dialers are used. Each modem or dialer type has exactly one entry.

Any entry consists of the following fields:

dialer This is the name of the dialer. It must match the *dialer* part of a *dialer-token* pair in the `Devices` file entry.

subst String of character translations. The first of each pair of characters is translated into the second when dialling the telephone number. This is generally used to substitute the '`=`' and '`-`' characters in the phone number.

modem-chat Organized simliarily to the chat script in the `Systems` file, it consists of pairs of expect/send strings used to initialize the dialer (or modem) and dial the phone number.

The following ecape characters are understood:

`\D` Send phone number without dialcode translation.

`\T` Send phone number with dialcode translation.

`\M` Do not require carrier.

`\m` Require carrier and fail if not present.

Apart from these, all escape characters specified for the `Systems` file chat script are understood.

7.3.5 The Do's and Dont's — The `Permissions` File

The `Permissions` file controls access *you* grant to *other* systems. Each entry is one line, possibly continued across a newline using a backslash ("\"). It consists of a list of assignments, separated by white space. An assignment is a token-value pair, separated by an equals sign (`=`), and should not contain white space. They may occur in any order. Any but the `LOGNAME` and `MACHINE` options have default values.

You may give different permissions to remote systems, based on whether they call you, or you call them. The first type is described by an entry containing `LOGNAME=`*login* which specifies the login id under which the remote system logs in. An entry containing `MACHINE=`*system* describes permissions for the remote system when we call it. You may also merge both entries for *system* into one entry by specifying both `LOGNAME` and `MACHINE`.

An entry specifying `MACHINE=other` applies to any machine calling in. This can be used to set permissions for any machine not explicitly mentioned.

Some options only apply when calling out, others only when a remote system calls in. The following entries may be used with both `LOGNAME` and `MACHINE` entries:

`REQUEST` If set to `yes`, the remote system may request files from you. Defaults to `no`.

`PUBDIR` The directory for public local access. Defaults to `/usr/spool/uucppublic`.

`READ` A colon-separated list of directories the remote system may request files from, and any subdirectories thereof. Defaults to `PUBDIR`.

`WRITE` A colon-separated list of directories the remote system may send files to, and any subdirectories thereof. Defaults to `PUBDIR`.

`NOREAD` A colon-separated list of directories excepted from `READ`.

`NOWRITE` A colon-separated list of directories excepted from `WRITE`.

`SENDFILES` If set to `yes`, jobs are transferred to the remote system when it calls. Otherwise, when set to `call`, your system will only transfer queued jobs when it calls out. If set to `no`, no jobs will be transferred at all.[10] Defaults to `call`.

`MYNAME` If you want to hide your system name when calling another machine, set `MYNAME` to your alias.

This option is very useful. Assume your machine is the gateway to a net called `daveg`, but its hostname is `cuxE91`. However, the site you call expects your name to be `daveg`. Instead of ripping apart your whole net, you can put `MYNAME=daveg` in your `Permissions` file, and `uucico` will report your hostname as `daveg` to the remote machine.

[10] This is specific to Taylor UUCP 1.03. In version 1.04, `no` is equivalent to `call`.

The following options are available for `LOGNAME` entries only:

`VALIDATE` This field is interpreted differently under versions 1.03 and 1.04. In UUCP 1.03, it specifies a colon-separated list of systems allowed to log in under this logname.[11] If this list is empty, any system name is valid.

In version 1.04, however, if a site is listed in the `VALIDATE` field, then it is *required* to use the corresponding `LOGNAME`. Other systems are still permitted to use this login. If you set up your system for dialup and allow anonymous UUCP, this option will come very handy. See section 7.5.

`CALLBACK` If set to `yes`, your system will hang up when the remote system calls you, and return the call. Defaults to `no`.

The following option is available for `MACHINE` entries only:

`COMMANDS` A colon-separated list of commands the remote system may execute on our machine. Defaults to `rnews:rmail`.

7.3.6 Be Paranoid — Call Sequence Checks

Call sequence checks are used to prevent and detect impostors. Imagine somebody else finds out the password you use to log into your mail feed. He or she could now call your feed, posing as you, and receive all your mail.

To guard against this, you can enable call sequence checks. For this, both machines keep track of the number of connections established so far. It is incremented with each connection. At login, the caller sends its call sequence number, and the callee checks it against its own number. If they don't match, the connection attempt will be rejected.

Even if some very clever person should detect your call sequence number as well as your password, you will find this out: the next time you try to log in, the remote `uucico` will refuse you, because the numbers don't match anymore! However, depending on the implementation of the remote `uucico`, it may still be possible for the impostor to log in.

With BNU-compatible UUCP's, this number is kept in the file `.Sequence` in the remote site's spool directory. It *must* be owned by `uucp`, and must be mode `600` (i.e. readable and writeable only by its owner). Call sequence checks are enabled by creating this file:

```
# cd /usr/spool/uucp/sitename # echo "4316" > .Sequence # chmod 600
.Sequence # chown uucp.uucp .Sequence
```

[11] This is also the way the O'Reilly book has it. According to Ian Taylor, this is wrong.

It is best to initialize this file with an arbitrary, agreed-upon start value. Otherwise, someone might manage to guess the number by trying out all smaller than, say, 20.

7.3.7 How to Specify a Direct Connection

Assume you use a direct line connecting your system `pablo` to `tiny`. How would you sepcify the device to use? Having read the above, you would certainly think: "Why, I use `Direct` in `Devices`, and...". But wait, what if you have *another* direct line, the second connecting you to `walt`? Which line is `uucico` assumed to pick? There is no way for it to see from an entry in `Devices` what system it is connected to.

There are several ways around this dilemma.

The first is to use a different name for each direct entry in `Devices`, e.g. one could take the name of the site at the other end of the wire instead of `Direct`. This method is widely used. However, this has a disadvantage: `uucico` decides by the entry's name if it is a direct line (`Direct`), a TCP connection (`TCP`), or a modem (anything else). Thus, using a name other than `Direct` causes `uucico` to assume that the line has a modem dangling off it. Since it might handle direct lines and modems differently, this could get you in trouble. However, you *should* do it this way if the connection is over a leased line (for which one generally uses modems on both ends).

Note, that since the line is not recognized as being direct, `uucico` will expect a dialer-token pair at the end of the entry. It is best to add a do-nothing entry named `direct` to the `Dialers` file, if the connection is a simple wire. When using a leased line, you will need a separate entry to access the modem, anyway.

The second way is to prepend any system identification to the speed, in effect defining a "modem class". You can now use `Direct` as the entry's name, making sure that the device is recognized as a direct line indeed. Since modem classes may not be implemented in all UUCP's, you may have problems using this method with `uucico`'s other than Ian Taylor's.

An example for both methods is given in the section below.

7.4 From System Name to Connect — How all this Works

After giving you an overview of what files UUCP uses, we now explain the process of accessing a system in greater detail, giving an extensive example.

7.4.1 Sample Files

Assume the sample files given in Fig. 7.1 through 7.4.

```
# /usr/local/lib/uucp/Systems
#
# This is our news/mail feed.
swim   Any/C,NonPeak  ACU,g  9600   Town=123456   ogin: joe ssword: Jabba
# alternate number, slow modem.
swim   Any/C,NonPeak  ACU,g  2400   Town=555555   ogin: joe ssword: Jabba
#
# The server for local distribution
server Any            TCP,e  -      srv.abc.com  ogin: joe ssword: ScrBlX
#
# A local machine with direct connection: using method #1
pablo  Any      pablo,g  38400      -            ogin: joe ssword: Catch22
#
# A local machine with direct connection: using method #2
tiny   Any      Direct,g tiny38400  -            ogin: joe ssword: iTsMe
```

Figure 7.1: The `Systems` file

7.4.2 Building up the Connection

◇ If `uucico` is called with the `-s` *sysname* option, it will check the `Systems` file for *sysname*. All entries for *sysname* are tried in the order given, until either a call succeeds, or there are no entries left.[12] First, the entry's schedule field is checked whether the call may be placed at the current time. No check of the maximum spool grade is done.

However, if `uucico` is invoked with flag `-r1`, it will loop over all entries in the `Systems` file. For each entry, the spool directory is scanned for jobs queued for this system. If there are no jobs, the respective system will not be called. If there are, the maximum grade of all jobs is compared against the spool grade specification in the schedule field. If there is no matching time/grade combination for the current entry, `uucico` continues scanning the system file; otherwise it tries to call the system.

[12] Alternate entries are also retried when the failure occured during the chat script, the handshake, or even during the transfer phase.

```
# /usr/local/lib/uucp/Devices
#
# TCP. Use service 'uucp' in /etc/services
TCP     uucp          -       -
#
# Hayes Modem on /dev/cua1
ACU     /dev/cua1     -       2400    hayes \T
ACU     /dev/cua1     -       9600    hayesv42 \T
#
# The direct connection to pablo, using dialer 'direct'
pablo   /dev/cua2     -       38400   direct
#
# The direct connection to tiny, no dialer
Direct  /dev/cua3     -       tiny38400
```

Figure 7.2: The Devices file

```
# /usr/local/lib/uucp/Dialers
#
# Hayes Modem at 2400 Baud, no compression
hayes     =,-,    "" AT OK ATZ OK AT\\N0M3E0Q1 OK \EATDT\T CONNECT
# Hayes Modem at 2400 Baud, V42bis
hayesv42  =,-,    "" AT OK ATZ OK AT\\N6M3E0Q1 OK \EATDT\T CONNECT
# Do-nothing dialer for direct lines
direct
```

Figure 7.3: The Dialers file

```
# /usr/local/lib/uucp/Permissions
#
# server: may do almost everything
LOGNAME=server MACHINE=server VALIDATE=server\
READ=/ SEND=/ \
REQUEST=yes SEND=yes\
COMMANDS=rnews:rmail:rsmtp:lpr:uux
# Ordinary mail and news accounts
LOGNAME=swim MACHINE=swim VALIDATE=swim\
REQUEST=yes SEND=yes\
COMMANDS=rnews:rmail
LOGNAME=pablo MACHINE=pablo VALIDATE=pablo\
REQUEST=yes SEND=yes\
COMMANDS=rnews:rmail
# mail-only link
LOGNAME=tiny MACHINE=tiny VALIDATE=tiny\
REQUEST=yes SEND=yes\
COMMANDS=rmail
# anon UUCP account.
LOGNAME=uucp MACHINE=other\
        COMMANDS=rmail REQUEST=yes SENDFILES=no
```

Figure 7.4: The Permissions file

If the entry matches, uucico will first check if there's already a connection open to this machine. This is signalled by a lock file in the spool directory. If there is one, the call fails. Otherwise, uucico will proceed and create a lock file itself.

Next, the device type is extracted from field 4 of the Systems file, and the Devices file is searched for a device matching the desired type. If it finds one, the speed fields are checked against each other, too. If the speed ranges specified in the Systems file entry and the Dialers file entry overlap, the highest common speed is selected. An entry containing Any or "-" matches any speed.

Then, uucico will open the device file or the TCP/IP socket. If the device is a modem, the dialer chat script specified in the Dialers file is executed first. Following this, the chat script from the Systems file is executed.

7.4.3 Calling out via Modem

Assume that you issued the command

```
$uucico -sswim
```

on the command line to connect to swim. Since uucico has been invoked with -s, it is satisified by the first entry which specifies Any as call time. If no lock file for swim exists, it will create one.

Then, uucico searches Devices for entries of type ACU at 9600 bps. It finds the modem on /dev/cua1, one entry using it at 2400 bps, and another at 9600 bps. The first entry doesn't match the transfer rate, but the second does. If cua1 is not locked, a lock file is created, and the call proceeds.

Next, uucico extracts the dialer-token pair from the entry. Since the token is \T, Town in the phone number will be substituted, using the Dialcodes file.

The dialer type, hayesv42, is matched against the dialer names in the Dialers file. A matching entry is found, and the "=" from the phone number is translated to the Hayes™-specific pause command ",".

Next, the modem is initialized using the modem chat script, and dials out. After receiving CONNECT from the modem, uucico executes the login chat script given in Systems: it waits for ogin:, then sends joe, waits for ssword: and sends Jabba. The remote end should now fire up its own uucico, and initial handshake between the two takes place (see 7.2.5).

Note that in this example, a single modem is set up so that it can be used at different speeds.[13]

[13] This is done by the \\N0 and \\N6 command in the modem chat script, respectively. I don't know if

7.4.4 Calling out via TCP

Assume that you issued the command

```
$ uucico -sserver
```

Exactly as in the above example, `uucico` searches `Systems`, now finding an entry describing `server`. The device field now specifies a TCP connection, and the phone number field contains the name the remote site responds to.

For TCP/IP connections, any `TCP` type device matches. Hence, you will only need one entry in your `Devices` file, just like in the example above.

`uucico` will now resolve the host name into an IP address using `gethostbyname()`. For this to work, you either have to have a `named` Domain Name Server installed, or `resolv.conf` and `hosts.equiv` set up correctly. Please refer to the networking FAQ.

Then the port number through which to reach the UUCP daemon on the remote host is determined from the second field in the `Devices entry`. This can either contain the port number itself, or a service name that is translated using `getservbyname()`.

Finally, the socket is bound to the address, and the login chat script is executed.

7.5 Setting up your System for Dialing in

If you want to set up your site for dialing in, you have to permit logins on your serial port, and customize some system files to provide UUCP accounts. This will be explained throughout this section.

7.5.1 Setting up `uugetty`

When setting up a serial line for use as a dialin port, you have to enable some `getty` program on this port. However, for ports you wantto use for both dialling in and out, the usual `getty` program is not enough, because it does not allow other programs to access the device. Instead, you have to use a companion program called *uugetty* that implements line sharing.

There's a `uugetty` program available for LINUX from the `getty_ps` suite. If you are running a recent SLS release, you will already have it — check this out in your `/etc` directory. Otherwise, you may obtain it from `tsx-11.mit.edu` as either binary or source.

that's standard Hayes, but my modem does it that way.

Line sharing between `uugetty` and other programs like `uucico`, `kermit` or `seyon` is implemented using UUCP lock files. When being invoked on serial line, say `/dev/ttyS1`, `uugetty` checks if there is a lock file named `LCK..cua1` in `/usr/spool/uucp`. If there is one, it puts itself to sleep for a minute. Otherwise, it initializes the line, and waits for an incoming connection. When accepting an incoming connection, it creates the lock file `LCK..cua1` itself, thus barring any other program from accessing the line. When another program tries to call out while `uugetty` is still waiting for an incoming connection, `uugetty` recognizes this and puts itself to sleep.

`uugetty` may be set up just the way you would configure `getty` for a serial port, except that the configuration file should now be called `uugetty.ttyS1` instead of `getty.ttyS1`. Everything else should go as described in section 6.7.

7.5.2 Providing UUCP Accounts

Next, you need one or more login names under which people may log into your system to do UUCP. Generally, this will be something like `uucp` for anonymous downloads, and a separate login for every site that polls you.

You should give all UUCP accounts a distinct group which has no special rights, for example `guest`. As shell, you enter `/usr/local/lib/uucp/uucico`. For example, if you have the shadow password suite installed, you may do this by invoking the `useradd` command with the following arguments:

```
# useradd -d /usr/spool/uucppublic -G guest \
-s /usr/local/lib/uucp/uucico monad
# useradd -d /usr/spool/uucppublic -G guest \
-s /usr/local/lib/uucp/uucico uucp
```

Afterwards, set the passwords for these accounts using the `passwd` command.

For UUCP, you should put the following lines into `Permissions`:

```
LOGNAME=uucp\
READ=/usr/spool/uucppublic\
NOREAD=/usr/spool/uucppublic/incoming\
WRITE=/usr/spool/uucppublic/incoming\
        COMMANDS=rmail REQUEST=yes SENDFILES=yes
LOGNAME=monad MACHINE=monad VALIDATE=monad\
        READ=/usr/spool/uucppublic WRITE=/usr/spool/uucppublic\
        COMMANDS=rnews:rmail:rsmtp REQUEST=yes SENDFILES=yes
```

The VALIDATE entry is necessary to prevent intruders from sneaking on monad's mail: Assume a system logs in as uucp and pretends its name is monad. If validation is not in effect, it is sent all jobs that are queued for monad![14]

Directories for any UUCP accounts are created by uucico as needed.

If you wish to prohibit interactive logins from the dialup port, possibly for security reasons, or because this would disrupt mail and news services, you may use the *porttime* feature of the shadow login suite. In /etc/login.defs, you set PORTTIME_CHECKS_ENAB to yes, and put the following into your /etc/porttime file:

```
# /etc/porttime
# Allow registered UUCP users, but throw out everybody else
ttyS1:uucp,monad:Al0000-2400
ttyS1:*:
```

This only allows the UUCP accounts monad and uucp to be accessed over the serial port, while everyone else will fail to log in. Details on the portttime file can be found in the corresponding section 4 manual page.

Note that anonymous UUCP in Taylor UUCP is currently somewhat broken. If you have enabled support for both HDB and Taylor-style configuration files — as is the case with the SLS UUCP —, uucico will fail to allow unkown systems to log in. This seems to be a deliberate lack of feature. If you want to provide anonymous UUCP, you either have to use Taylor-style configuration files, or recompile uucico with HDB support exclusively.

7.5.3 Accepting UUCP logins over TCP/IP

If you want to use Taylor UUCP to service requests over TCP/IP, you have to add the following to the file /etc/services:

```
uucpd   540/tcp
```

To make inetd service requests for it, add the following to /etc/inetd.conf:

```
uucpd   stream tcp nowait root /usr/local/lib/uucp/uucico uucico -l
```

Instead of using inetd, you may also start uucico at boot time with the -e Option. This makes uucico sit in a loop, listen to the port specified, and wait for requests.

[14]This also seems to be a problem with various version 2 implementations. Amiga-UUCP is one of them.

If a connection via TCP occurs, `uucico` will prompt for login and password. However, it will not use `/etc/passwd` for authorization, but rather its private `/usr/local/lib/uucp/passwd` file.

Of course you can select any other port number than 540, although this is the default used by `uucico`.

7.6 Miscellaneous

7.6.1 Troubleshooting

This section describes what may go wrong with your UUCP connection, and makes suggestions where to look for the error.[15] However, the questions were compiled by the author off the top of his head. There's much more that can go wrong.

In any case, enable debugging with `-xall`, and take a look at the output in `.Admin/audit.local` in the spool directory. It helps you quickly recognize where the problem lies. Also, I have always found it helpful to turn on my modem's speaker when it didn't connect. With Hayes™-compatible modems, this is accomplished by adding "`ATL1M1 OK`" to the modem chat in the `Dialers` file.

Something is wrong: The first check always should be if all file permissions are set correctly. `uucico` should be setuid `uucp`, and all files in `/usr/local/lib/uucp`, `/usr/spool/uucp` and `/usr/spool/uucppublic` should be owned by `uucp`. Remember to check the hidden files in the spool directory.

I can connect to the remote site, but the chat script fails: Look at the text you receive from the remote site. If it's garbled, this might be a speed-related problem. Otherwise, confirm if it really agrees with what your chat script expects. Remember the chat script starts with an expect string. If you receive the login prompt, then send your name, but never get the password prompt, insert some delays before sending it, or even in-between the letters. You might be too fast for your modem.

My modem does not dial: If your modem doesn't indicate that the `DTR` line has been raised when `uucico` calls out, you possibly haven't given the right device to `uucico`. Check `Devices`. Also, see 6.4 on how to find out the device. If your modems recognizes `DTR`, check with `kermit` that you can write to it. If this works, turn on echoing with `\E` at the start of the modem chat. If it doesn't echo your commands during the modem chat, check if your line speed is too high or low for your modem. If you see the echo, check if you have disabled

[15] This amounts to a short UUCP-FAQ. I would be glad if someone took this as a basis for a real FAQ.

modem responses, or set them to number codes. Verify that the chat script itself is correct. Remember that you have to write two backslashes to send one to the modem.

My modem tries to dial, but doesn't get out: Insert a delay into the phone number. This is especially useful when dialling out from a company's internal telephone net. For people in Europe, who usually dial pulse-tone: try touch-tone. In some countries, postal services have been upgrading their nets recently. Touch-tone sometimes helps.

I have extremely high packet loss rates: This looks like a speed problem. Maybe the link between computer and modem is too slow (remember to adapt it to the highest effective rate possible)? Or it is your hardware that is too slow to service interrupts in time. With a NSC 16550 chipset on your serial port, 38kbps are said to work reasonably well; however, without FIFOs (like 16450 chips), 9600 bps is the limit.

I can log in, but handshake fails: Well, there can be a number of problems. The output in the log file should tell you a lot. Look at what protocols the remote site offers (It sends a string `P`*protlist* during handshake). Maybe they don't have any in common (did you select any protocols in `Systems` or `Devices`?).

If the remote system sends `RLCK`, there is a stale lockfile for you on the remote system. If it's not because you're already connected to the remote system on a different line, ask to have it removed.

If it sends `RBADSEQ`, the other site has conversation count checks enabled for you, but numbers didn't match. If it sends `RLOGIN`, you were not permitted to login under this id. (Possibly you forgot to set `MYNAME` in `Permissions`?)

7.6.2 Log files

When compiling the UUCP suite to be HDB-compatible, there are several places where log files may go. Most commands will always generate some amount of informational information. This is logged to files below `/usr/spool/uucp/.Log`. This directory contains three more directories, named `uucico`, `uuxqt`, and `uux`. They contain the logging output generated by each of the corresponding commands, sorted into different files for each site. Thus, output from `uucico` when calling site `swim` will go into `.Log/uucico/swim`, while the subsequent `uuxqt` run will write to `.Log/uuxqt/swim`.

When enabling debugging output, this will go to the `.Admin` directory below `/usr/spool/uucp`. When calling out, debugging information will be sent to `.Admin/audit.local`, while the output from `uucico` when someone calls in will go to `.Admin/audit`.

7.6.3 Available line protocols

To negotiate session control and file transfers with the remote end, `uucico` uses a set of standardized messages. This is often referred to as the high-level protocol. During the initialization phase and the hangup phase these are simply sent across as strings. However, during the real transfer phase, a low-level protocol is employed which is mostly transparent to the higher levels. This is to make error checks possible when using telephone lines, among other things.

This low-level protocol can be chosen from among a set of protocols, depending on the nature of the connection used. However, not every implementation of `uucico` knows every protocol, so during initialization phase, both processes have to agree on a common protocol. As described in section 7.2.5, you can select a list of protocols to be offered to the remote `uucico` when your UUCP calls out. You can do this by appending a string of protocol names to the device field in `Systems`, offset by a comma. A protocol name is a single letter.[16]

The following protocols are available with Taylor UUCP 1.03:

g This is the most common protocol and should be understood by virtually all `uucico`'s. It does thorough error checking and is therefore well-suited for noisy telephone links. *g* requires an eight-bit clean connection.

It is a packet-oriented protocol which uses a slide-window technique (See below for an explanation of these verbal monsters).

Because of the overhead involved, *g* is not considered efficient for reliable links such as TCP connections.

t This is a protocol intended for use over a TCP connection. It is only for use with truly error-free networks. It uses packets of 1024 bytes and requires an eight-bit clean connection.

e This should basically do the same as *t*. The main difference is that *e* is a streaming protocol.

f It is intended for use with reliable X.25 connections.[17] It is a streaming protocol and expects a seven-bit data path; eight-bit characters are quoted, which can make it very inefficient.

[16] Note that protocol names are case-sensitive. Some implementations of `uucico` have a *G* protocol, for example, which is a slightly modified *g* protocol.

[17] X.25 is a standard issued by CCITT, describing the lower three layers (i.e. physical, data link, and networking) for digital connections over public phone lines. Special equipment is needed for X.25, namely a so-called *PAD*, meaning Packet-Assembler-Disassembler.

Protocols can be divided into two categories: streaming and packet-oriented protocols. The first type transfers a file as a whole, possibly computing a checksum over it. This is nearly free of any overhead, but requires a reliable connection, because any error will cause the whole file to be retransmitted. These protocols are not suitable for use over telephone lines. Although modern modems do quite a good job at error checking, there is no such thing as error detection between your computer and the modem.

On the other hand, packet protocols split up the file into several chunks of equal size. Each packet is sent and received separately. With the *g* protocol, a checksum is calculated for each packet, and the recipient has to acknowledge its correct transmission. To make this more efficient, sliding-window protocols were invented, which allow for a limited number (a window) of outstanding acknoledgements at any time. This greatly reduces the amount of time `uucico` has to wait during a transmission.

Packet sizes are generally powers of 2. The *g* protocol has a maximum window size of 7, and packet sizes ranging from 64 through 4096. With BNU configuration files, these parameters cannot be set by the user.[18] Taylor UUCP currently comes with a pre-compiled default of 7 windows and 64 byte packets.[19]

The width of the data path also makes a difference. When transmitting eight-bit characters over a seven-bit connection, they have to be quoted. Under worst-case assumptions, this doubles the amount of data to be transmitted, although compression done by the hardware may compensate for this. On the other hand, if you use a modem, an eight-bit clean connection forbids software handshake between modem and computer. In this case, make sure that both modems perform hardware handshaking.

Taylor UUCP version 1.04 adds a couple of new protocols. These are

i
: This is a bidirectional protocol which can send and receive files at the same time. It requires a full-duplex connection and an eight bit data path. You will only recognize an advantage over the *g* protocol if you are sending about as much as you are receiving.

G
: This is the System V Release 4 version of the *g* protocol. It is also understood by some other versions of UUCP.

a
: Similiar to ZMODEM. Requires an eight bit connection, but quotes certain control characters like `XON` and `XOFF`.

[18] This is possible when compiling Taylor UUCP to use "Taylor configuration files". You are basically on your own doing this. However, the sources contain `texinfo` files describing their format.

[19] You can't get above this. During the *g* protocol startup, both processes exchange their preferred packet and window sizes, and settle for the smaller of each.

All these are also packet protocols with many parameters. When using Taylor configuration files, they can be tuned by the user. With HDB, defaults are provided.

7.6.4 Notes

Taylor UUCP is covered by the GNU public license, which is reproduced in appendix B. The latest version is always available as `prep.ai.mit.edu:/pub/gnu/COPYING`.

There is a mailing list for users of Taylor UUCP. To subscribe (or leave), send mail to

```
taylor-uucp-request@gnu.ai.mit.edu
```

Please note that this mail is handled by a human being. To write a message to the list, simply send it to `taylor-uucp@gnu.ai.mit.edu`. Bug reports should also be directed to this list.

Chapter 8

Electronic Mail

One of the most prominent uses of networking is electronic mail, and has been so since the first networks were devised. It started as a simple service that copied a file from one machine to another, and appended it to the recipient's *mailbox* file. Basically, this is still what email is all about, although an ever growing net with its complex routing requirements and its ever increasing load of messages transported has made a more elaborate scheme necessary.

Various standards of mail exchange have been devised. Sites on the Internet adhere to one laid out in RFC 822, augmented by some RFCs that describe a machine-independent way of transferring special characters, and the like. Much thought has also been given recently to "multi-media mail", which deals with including pictures and sound in mail messages. Another standard, X.400, has been laid down by CCITT.

In this chapter, we will deal with what email is and what issues you as an administrator will have to deal with. Actual configuration details are covered throughout the next chapter.

8.1 What is a Mail Message?

A Mail message generally consists of a message body, which is the text the sender wrote, and special data specifying recipients, transport medium, etc., very much like what you see when you look at a letter's envelope.

This administrative data falls into two categories; in the first category is any data that is specific to the transport medium, and which may be regenerated by the transport software as the message is passed along. In UUCP networks, the *route* belongs here. In its entirety, this data is generally referred to as the *envelope*.

The second variety is any data necessary for handling the mail message, which is not particular to any transport mechanism. It is often used to generate the envelope data, or to determine the message's sender. In most networks, it has become standard to prepend this data to the mail message. This is the so-called *mail header*. It is offset from the *mail body* by an empty line.[1]

Most mail transport software in the Un⋆x world uses a header format outlined in a document named RFC 822.[2] Its original purpose was to specify a standard for use in the ARPA Internet, but since it was designed to be independent from any environment, it has been easily adapted to other networks, including many UUCP-based networks.

RFC 822 however is only the greatest common denominator. More recent standards have been conceived to cope with growing needs as, for example, data encryption, international character support, and "multi-media mail".

All these standards have in common that the header consists of several lines, separated by newline characters. A line is made up of a field name, beginning in column one, and the field itself, offset by a colon and white space. The format and semantics of each field vary depending on the field name. A header field may be continued across a newline, if the next line begins with a `TAB`. They may appear in any order.

Usually, all necessary header fields are generated by the mailer interface you use, like `elm`, `mush`, or `mailx`. Some however are optional, and may be added by the user. `Elm`, for example, allows to edit part of the message header. Others are added by the mail transport software.

Typical header field names and their meaning are:

`From:` This contains the sender's email address, and possibly the "real name". A complete zoo of formats is used here.

`To:` This is the recipient's email address.

`Subject:` Describes the content of the mail in a few words. At least that's what it *should* do.

`Date:` The date the mail was sent.

`Reply-To:` Specifies the address the sender wants the recipient's reply directed to. This may be useful if you have several accounts, but want to receive the bulk of mail only on the one you use most frequently. This field is optional.

---

[1] It is customary to append *signature* or `.sig` to a mail message, usually containing information on the author, along with a joke or a motto. It is offset from the mail message by a line containing "`-- `".

[2] There are other standards, of course. One more I know of is X.400, which was issued by the CCITT.

`Organization:`
The organization that owns the machine from which the mail originates. If your machine is owned by you privately, either leave this out, insert "private" or some complete nonsense. This field is optional.

`Message-ID:` A string generated by mail transport on the originating system. It is unique to this message. An example is `<199301111938.AA08057@jti.com>`.

`Received:` Every site that processes your mail (including the machines of sender and recipient) inserts such a field into the header, giving its site name, a message id, time and date it received the message, from which site and using which transport software. This is so that you can trace which route the message took, and can complain to the person responsible if something went wrong.

`X-`*anything:* No mail-related programs should complain about any header which starts with `X-`. It is used to implement additional features that have not yet made it into an RFC, or never will. This is used by the `linux-activists` mailing list, for example, where the channel is selected by the `X-Mn-Key:` header field.

The one exception to this structure is the very first line. It starts with the keyword `From` which is followed by a blank instead of a colon. It contains the route the message has taken in UUCP bang-path style, time and date when it was received by the last machine having processed it, and an optional part specifying which host it was received from. This field is there for backward compatibilty with some older mailers, but is not used very much anymore, except by mail user interfaces that rely on it to mark the beginning of a message in the user's mailbox. To avoid potential trouble with lines in the message body that begin with "`From `", too, it has become standard procedure to escape any such occurence by preceding it with "`>`".

8.2 How is Mail Delivered?

Generally, you will compose mail using a mailer interface like `mail` or `mailx`; or more sophisticated ones like `elm`, `mush`, or `zmailer`. These programs are called *mail user agents*, or MUA's for short. If you send a mail message, the interface program will in most cases hand it to another program for delivery. This is called the *mail transport agent*, or MTA. On some systems, there are different mail transport agents for local and remote delivery; on others, there is only one. The command for remote delivery is usually called `rmail`, the other `lmail`.[3]

The transport software used depends on the nature of the link. If the mail must be delivered over a network using TCP, SMTP is commonly used. SMTP stands for Simple Mail Transfer Protocol. It is defined in RFC 788 and RFC 821.

Over UUCP links, one will generally invoke `uux` to execute `rmail` on the remote system. However, it is also possible to produce a batch file that contains the SMTP commands that would normally be issued when a direct SMTP connection was used. This is called BSMTP, or *batched* SMTP. Then, `uux` is invoked to execute a command named `rsmtp` or `bsmtp` on the remote system, giving it this data as input. This program will process the input as if a normal SMTP connection had occurred.

The benefit of the latter method is that `rmail` takes the envelope on the command line, while `rsmtp` takes it from the mail batch. This effectively keeps the shell from misinterpreting characters as special shell characters when invoking `uux` or `rmail`.[4]

8.3 Email Addresses

To send a message to another person, you have to have some way to identify her to the person or organization delivering the message. This identification is commonly called an address. For electronic mail, an address is made up of at least the name of a machine handling the person's mail, and a user identification recognized by this system. This may be the recipient's login name, but may also be anything else. Other mail addressing schemes, like X.400, use a more general set of "attributes" which are used to look up the recipient's host in an X.500 directory server.

8.3.1 Various Address Formats

The way a machine name is interpreted, i.e. at which site your message will finally wind up, and how to combine this name with the recipient's user name greatly depends on the network you are on. In the original UUCP environment, the prevalent form was *path*`!`*host*`!`*user*, where *path* described a sequence of hosts the message had to travel before reaching the destination *host*. This construct is called the *bang path* notation.

[3] The local mail transfer agent in the SLS distributions of Linux is Fred van Kempen's `wmail`.

[4] Some may recall that this was exactly the problem with `smail-2.5`, which used the `system()` library call to execute `uux`. On Linux systems, this caused big trouble since `bash`'s history mechanism tried to expand any occurences of `!user`, so that exclamation marks had to be escaped before sending them to the shell.

Internet sites adhere to the RFC 822 standard, which uses a notation of *user@host.domain*, where *host.domain* is the host's fully qualified domain name. The middle thing is called an "at" sign. Because this notation does not involve a route to the destination host, but gives the (unique) hostname instead, this is called an *absolute* address. Today, many UUCP-based networks have adopted RFC 822, and will understand this type of address.

Now, these two types of addressing don't mix too well. Assume an address of `hostA!user@hostB`. It is not clear whether the '`@`' sign takes precedence over the path, or vice versa: Do we have to send the `hostB`, which mails it to `hostA!user`, or should it be sent to message to `hostA`, which fowards it to `user@hostB`?

Addresses that mix different types of address operators are called *hybrid addresses*. Most notorious is the above example. It is usually resolved by giving the path precedence over the '`@`' sign. In the above example, this means sending the message to `hostA` first.

However, there is a way to specify routes in RFC 822-conformant ways: `<@hostA,@hostB:user@hostC>` denotes the address of `user` on `hostC`, where `hostC` is to be reached through `hostA` and `hostB` (in that order). This type of address is freqeuently called a *route-addr address*.

Then, there is the '`%`' address operator: `user%hostB@hostA` will first be sent to `hostA`, which expands the rightmost (in this case, only) percent sign to an '`@`' sign. The address is now `user@hostB`, and the mailer will happily forward your message to `hostB` which delivers it to `user`. This type of address is sometimes referred to as "Ye Olde ARPANET Kludge". It was used on the ARPANET to reach hosts who were not registered officially with the Network Information Center, so that messages had to be sent via a mail relay that was known to handle mail for these "hidden" sites.

Other networks have still different means of addressing. DECnet-based networks, for example, use two colons as an address separator, yielding an address of *host*::*user*.[5] Lastly, the X.400 standard uses an entirely different scheme, by describing a recipient's by a set of attribute-value pairs, like country and organization. On FidoNet, each user is identified by a code consisting of four numbers, denoting country, node, point, and user.[6]

There are some implications to using these different types of addressing which will be described throughout the following sections. In a RFC 822 environment, however, you will rarely use anything else than absolute addresses like `user@host.domain`.

[5] When trying to reach a DECnet address from an RFC 822-environment, you may use "*host*::*user*"@*relay*, where *relay* is a known Internet-DECnet relay.

[6] Is this correct?

8.3.2 Bang Path Addresses

When sending mail to a host in a UUCP environment, simply knowing the user and hostname might not be enough: When you are in Europe and want to send a message to a person in the United States — how should the intermediate sites know where to forward the message to? In earlier times, the solution was the so-called *bang path*, being a list of hosts through which to forward the message, separated by exclamation marks (`!`) and followed by the user's name. To send a letter to Janet User on a machine named `moria`, you would have used the path `eek!foo!bar!swim!moria!janet`. This would have sent the mail from your host to `eek`, from there on to `foo`, and so on, until it finally reached `moria`. The number of machines to traverse before reaching the destination host is usually referred to as the number of *network hops*.

This technique is called *source routing*, because it leaves the responsibility of finding a route to the recipient with the sender.[7] This is very different from the way we have seen IP handle routing, where the actual routing decisions are made by the forwarding host.

The obvious drawback of source routing is that it requires you to remember much about the network topology, fast links, etc. Second, changes in the network topology — like links being deleted or hosts being removed — may cause messages to fail simply because you weren't aware of the change. And finally, in case you move to a different place, you will most likely have to update all these routes. One thing, however, that made the use of source routing necessary was the presence of ambiguous hostnames: For example, if there are two sites named `moria`, one in the U.S., and one in France. Which site now does `moria!janet` refer to? This can be made clear by specifying what path to reach `moria` through.

The first step in disambiguating hostnames was the founding of *The UUCP Mapping Project*. It is located at Rutgers University, and registers all official UUCP hostnames, along with information on their UUCP neighbors and their geographic location, making sure no hostname is used twice. The information gathered by the Mapping Project is published as the *Usenet Maps*, which are distributed regularly through Usenet.

Using the connectivity information provided in the maps, smart mail software relieves the sender of the message from finding a path to the recipient host. You only give it the destination address as `moria!janet` or `janet@moria.uucp`, and using the maps, it is able to construct a path from your site to `moria`.

This is only a slight variation of source routing, and is sometimes called *system routing*.

[7] Actually, the term *source routing* is a bit ambiguous. A different use of it is to denote a routing scheme that takes the sender address into account when determining how to process a message. For example, a mail hub might handle mail for several domains, some of which have bought IP service, while others have not. Messages originating from a host in one of the latter domains should not be forwarded over the Internet, while the others definitely should.

Now, the only effect of changes in the network is that the host's mail routing databases have to be updated. Usually, they are kept up-to-date by rebuilding them every time updated UUCP maps are published.

It is important to know that the bang path routes used by the mail software do not have to be *strict*, that is `eek` and `foo` from the above example are not required to be direct UUCP neighbors. This is called *loose source routing*, and only requires that the mail transport software on `eek` is able to find a route to `foo`. For example, `eek` might have a UUCP neighbor, `ernie`, which has a link to `bert`, which in turn is a neighbor of `foo`. The mailer on `eek` will then amend the path, so that after processing on `eek`, the message would be sent to `ernie` with a new recipient address of `bert!foo!bar!swim!moria!janet`.

A benefit of loose routing is that instead of complete paths, a path from a well-known site to the recipient site suffices. Well-known means, of course, that all hosts know how to route messages to this site. For example, `uunet` is frequently given as a starting point of routes. When you specify a path starting with `uunet`, the mail software will try to get the message to `uunet`, which will then route your message (or probably no-one will). For example, you could send your mail to `uunet!moria!janet`, and leave it to `uunet` to find a route to `moria`. Unnecessary to say that this method is highly inefficient.

8.3.3 Addresses in the Domain Name System

However, the scheme described above is not really perfect. Since the beginning of the eighties, the numbers of sites and computer networks have exploded, so that it is virtually impossible to keep routing information up to date. Also, many of the central sites are on the Internet now, where routing is completely different from a UUCP network. Last but not least, site names on UUCP networks are generally limited to seven, or at most eight characters, so there is some shortage of names.[8]

Therefore, host names have been organized in a hierarchy of *domains*, as described in section 2.3: A domain is a collection of sites that are related in some sense — be it because they form a proper network (e.g. all machines on a campus, or all hosts on BITNET), because they all belong to a certain organization (like the U.S. government), or because they're simply geographically close. Such a domain may itself be part of a larger domain, which is in turn part of an even larger domain. This relationship is described as being a *subdomain* of the larger domain. To produce a unique address for each machine, names inside a domain must be unique. Assume the owners of `moria` join a small non-profit organization, say Orcs And Thugs Association, running a network of UUCP sites and paying to other organizations

[8] For example, forget about any names from 'folklore' books, like *The Lord of the Rings*, or *The Hitchhiker's Guide to the Galaxy*.

for access to internation mail services. The network run by them may be called `orcnet`, and because they are a some unspecified sort of organization, they are located below the `org` top-level domain. Thus, `moria`'s proper name would be `moria.orcnet.org`. This is called its *fully qualified domain name*, and uniquely determines the site.

Now, Janet's address would be `janet@moria.orcnet.org`. However, most sites also understand it when you use domain names in bang paths, like `swim.two.birds!moria.orcnet.org!janet`. Note that there are still UUCP sites that are not part of any domain. These are considered part of the pseudo-domain `.uucp`.

The domain name system brings another type of routing to UUCP-based networks. A domain, say `foobar.com`, may decide to publish only a number of its UUCP sites, and hide the internal network. Now these public sites may be used as gateways to the internal network: any message to an unknown site below the domain `.foobar.com` will be sent to one of the gateways, which is assumed to forward the message properly.

This method is called *domain-based routing*. It is primarily useful for large domains, because this allows it to change its internal routing strategy without updating any map entry at the UUCP Mapping Project. New routing information only has to be propagated to the member sites of the domain, which makes it much more flexible. Inside the domain, any routing scheme may be used: through maps distributed inside the domain, a routing scheme based on the subdomain name, or some sort of geographically based routing.

8.4 How does Mail Routing Work?

The process of directing a message to the recipient's host is called *routing*. Apart from finding a path from the sending site to the destination, it involves error checking as well as speed and cost optimization.

There is a big difference between the way a UUCP site handles routing, and the way an Internet site does. On the Internet, the main job of directing data to the recipient host (once it is known by it's IP address) is done by the networking layer (usually running the IP protocol), while in the UUCP zone, it has to be supplied by the user, or must generated by the mail transfer agent.

8.4.1 Mail Routing in UUCP networks

Compared to the Internet, routing in UUCP-based networks is rather static. It generally happens by means of a so-called *paths* file. This is a file that translates host or domain names to UUCP bang paths.

On larger systems, this database is often generated from the UUCP maps distributed through Usenet.[9] There is a tool for converting these maps, called `pathalias`. For this reason, the paths file is also sometimes referred to as the pathalias database.

A map is simply a file containing a list of UUCP systems, for each site listing its location, any aliases or fully qualified domain names, and the sites it is linked to. Links are weighted with a certain *cost*. This is a symbolic cost computed from the speed and frequency of the connection (see section 8.5). From the map files, a program called `pathalias` creates a list of paths from your site to each system in the maps.

If you are a small leaf site and you're only connected to one site that provides you with mail and news, no paths file will be necessary at all. That is, the only routing you do is to accept any mail that is destined for you, and send anything else to your feed, which will route it for you. Therefore, there is no need to maintain any routing information from UUCP maps, because your feed will take care of that.

But even on most sites that do routing, not all addresses can be known. Therefore, messages to an unknown address are passed on to a site that is supposedly "smarter" and might know how to get the message to the recipient. This is called *smart host routing*.[10] Of course this does not apply to addresses you know you should be able to resolve. For example, if your site has complete routing information on the domain `orcnet.org`, and there is no site named `bush.orcnet.org`, you will certainly return any mail for that address instead of routing it to the smart host. One says that your site is *authoritative* for the domain `orcnet.org`.

Therefore, if you need to do routing for a small number of sites, a hand-written paths file (or one generated from a hand-written maps file) that handles mail for them but sends everything else to a smart host, may still be a safe bet.

8.4.2 Mail Routing on the Internet

On the Internet, little static routing information is available locally because of the overhead distributing this information would involve. Instead, the Internet Domain Name Service (or DNS for short) is used, which is a distributed database describing the net topology.[11] A host name is *resolved* (i.e. converted to an IP address) by a query to the local *name server*, usually called `named` on Un⋆x systems. Given a fully qualified domain name, it tries to find

[9] Maps for sites registered with The UUCP Mapping Project are distributed through the newsgroup `comp.mail.maps`; other organizations may publish separate maps for their network.

[10] One can view smart host routing as a variant of domain-based routing, namely for the root domain "`.`". In fact, this is the way it is implemented in some mailers, e.g. Fred van Kempen's `umail`.

[11] This is described in section 2.3. For more information on the Internet domain name system, refer to RFC 1034 and RFC 1035, and documentation on `named`.

the corresponding IP address using the information it has in its cache. If it doesn't, it may pass on the query to another name server that may have more information.

Many Internet service providers also offer mail and news transport services to sites and networks that are not on the Internet. To make this fact known, they publish a so-called `MX` record for these sites or domains.[12] It basically states that this site is willing to act as a mail forwarder for these sites. `MX` records may also be used by large organizations that want to have all their mail traffic handled by a limited number of special hosts only (i.e. *gateways*). These will have `MX` records for the whole domain.

An organization, say Foobar Inc., won't allow most of their machines to be on the Internet directly because of security reasons. Hence a message from `janet@groucho.edu` to `joe@missile-lab.foobar.com` cannot be delivered directly, because the host `missile-lab` may be shut off from any IP-traffic to outside of `foobar.com`. But another of Foobar's hosts, say `gateway.foobar.com`, will be on the Internet, and it will have an `MX` record set for the domain `foobar.com`. A mail transport agent on `groucho.edu` will therefore send the message to the gateway, which will deliver it to `joe@missile-lab`.[13] `MX` records also have a *preference* associated with them. This is a positive integer. If several mail exchangers exist for one host, the mailer will try to transfer the message to the exchanger with the lowest precedence value, and only if this fails will it try a host with a higher value.

Another type of resource record that is of interest to mail routing is the `WKS`, or *well-known service* record. On TCP/IP-based networks, peer processes generally connect to each other by attaching to a *port* on the remote host. The default port[14] for SMTP connection is port 25. If however a site is using a different port number for SMTP, it may make this (as well as other ports to well-known services) known using the `WKS` record.[15]

8.5 Pathalias and Map File Format

A pathalias database is used to map site names to UUCP bang paths. This provides the main routing information in UUCP-based networks. A typical entry qould look like this

[12] `MX` stands for Mail Exchanger. It is a resource record type in the DNS database.

[13] Note that the mailer might generate an envelope address that specifies the route the message should take. In our example it would look something like `<@gateway.foobar.com:joe@missile-lab.foobar.com>`. Specifying a path in this syntax is called *source routing*, or *route-addr address*. These expressions can be made arbitrarily complex.

[14] There are a number of widely-used Internet services that have default numbers; they are called *well-known services*. The assignment of numbers is regularly relased as an RFC, titled "Assigned Numbers". The latest release is RFC 1060. The assignment of serives to port numbers on your machine is stored in a file name `/etc/services`.

[15] For more information on the way mail routing in the Internet works, please refer to RFC 974.

(site name and path are separated by a TAB):

```
uunet           moria!ernie!bert!%s
uunet.uu.net    moria!ernie!bert!%s
```

This makes any message to `uunet` be delivered via `moria`, `ernie` and `bert`. Both `uunet`'s fully qualified name and its UUCP name have to be given if the mailer does not have a seperate way to map between these name spaces.

If you want to direct all messages to hostnames below a certain domain to a given relay, you may also specify a path in the pathalias database, giving the domain name as target, preceded by a dot. For example, if Fidonet may be reached through `swim!fidogate`, a pathalias entry might look like this:

```
.fidonet        swim!fidogate!%s
```

Writing a pathalias file is only acceptable when you are running a very small site that does not have to do much routing. A better way is to create a pathalias file from so-called map files using the `pathalias` tool.

A map file mainly consists of a list of systems, for each of them listing the sites it polls or is polled by. The system name begins in column one, and is followed by a comma-separated list of links. The list may be continued across newlines if the next line begins with a tab. Each link consists of the name of the site, followed by a cost given in brackets. The cost is an arithmetic expression, made up of numbers and symbolic costs. Lines beginning with a hash sign ('#') are ignored.

Routing information for all sites registered with the UUCP Mapping Project is regularly posted to `comp.mail.maps`, and allows you to produce a pathalias database for all systems listed therein. However, this information is rarely accurate or complete, so that you should either leave routing to major sites, or rely on special maps distributed by your network providers.

As an example, consider the site `moria`. It polls site `swim.two.birds` twice a day, `mordor.orcnet.org` hourly, and once per week `snorkel.com`. Moreover, the link to `snorkel` only uses a slow 2400 Baud modem, while the link to `mordor` is at 14400. Then its map entry would be something like

```
moria.orcnet.org
        swim.two.birds(DAILY/2),
        mordor.orcnet.org(HOURLY+FAST),
```

```
        snorkel.com(WEEKLY+LOW)

moria.orcnet.org = moria
```

The last line would make it known under its UUCP name, too. Note that it must be `DAILY/2`, because calling twice a day actually halves the cost for this link.

Using the information from such map files, `pathalias` is able to calculate "optimal" routes to any destination site listed in the paths file, and produce a pathalias database from this which can then be used for routing to these sites.

`pathalias` provides a couple of other features like site-hiding (i.e. making sites only accessible through a gateway) etc. See the manual page for `pathalias` for details, as well as a complete list of link costs.

The comments in the map file generally contain additional information on the sites described in it. There is a rigid format in which to specify this, so that it can be retrieved from the maps. E.g., there is a program called `uuwho`, which uses a database created from the map files to display this information nicely formatted.

When you register your site with an organization that distributes map files to its members, you generally have to fill out such a map entry.

Below is given an example for a map entry (in fact, it's the one for my site):

```
#N      monad, monad.swb.de, monad.swb.sub.org
#S      AT 486DX50; Linux 0.99
#O      private
#C      Olaf Kirch
#E      okir@monad.swb.de
#P      Kattreinstr. 38, D-64295 Darmstadt, FRG
#L      49 52 03 N / 08 38 40 E
#U      brewhq
#W      okir@monad.swb.de (Olaf Kirch); Sun Jul 25 16:59:32 MET DST 1993
#
monad   brewhq(DAILY)
# Domains
monad = monad.swb.de
monad = monad.swb.sub.org
```

The white space after the first two characters is a `TAB`. The meaning of most of the fields is pretty obvious; you will receive a detailed description from whichever domain you register with. The `L` field is the most fun to find out: it gives your geographical position in latitude/logitude and is used to draw the postscript maps that show all sites for each country, as well as world-wide.[16]

8.6 Message Grading

To distinguish between important and less important mail, it is possible to assign a *grade* to the message. The mail transport software may (or may not) evaluate this information, and probably use different transports depending on its value. For example, very urgent mail might be forwarded immediately, while ordinary mail is queued until the next regular contact is established. Also, treatment in case of an error may vary: normally, an undeliverable message will "bounce" and be returned to the sender in its entirety. Junk mail, on the other hand, could be thrown away without notice.

Normally, the message grade is extracted from a header field named `Precedence:`. It may contain a number of predefined tokens; if it doesn't exist, or contains an unknown token, the mailer will generally assume a default grade.

The tokens most widely used (because `sendmail` understands them) are, in decreasing order: `special-delivery`, `first-class`, and `junk`. Other mailers, like `smail`, add more tokens, or even allow for the definition of your own set of tokens. However, it is up to the administrator's whim what treatment she assigns to these tokens. She may even decide to give all of them equal precedence.

In the UUCP zone, message grades may be used easily to assign different service quality, by mapping them onto UUCP spool grades (see section 7.2.4). The UUCP spool grade determines at what times the message may be transferred, and hence how fast it travels to the next node. It is, however, up to the next node what level of service it assigns to the message's precedence.

8.7 Mail Software Configuration

There are a number of mail transport agents that have been implemented for Un$\star$x systems. One of the best-known is the University of Berkeley's `sendmail`, which is used on a number of platforms. The list of people having contributed to it is very long, and there isn't one

[16]They are posted regularly in `news.lists.ps-maps`. Beware. They're HUGE.

single author. There are two ports of `sendmail-5.56c` to LINUX available, one of which will be described below.

The mail agent commonly used with LINUX is `smail-3.1.28`, written and copyright by Curt Landon Noll and Ronald S. Karr. This is the one included in the SLS release and Ed Carp's mailpak package. In the following, we will refer to it simply as `smail`, although there are other versions of it which we don't describe here.[17]

Compared to `sendmail`, `smail` is rather young. When handling mail for a small site without complicated routing requirements (say, between networks using different addressing schemes), their capabilities are pretty close. For large sites, however, `sendmail` always wins, because it supports addressing formats like DECnet, etc.

Both `smail` and `sendmail` support a set of configuration files that have to be customized. Apart from the information that is required to make the mail subsystem run (such as the local hostname), there are many more parameters that may be tuned.

`sendmail`'s main configuration file is `sendmail.cf`. It looks as if your cat had taken a nap on you keyboard while pressing the shift key.[18] However, there's an extension to `sendmail`, called `IDA`, that provides for easier configuration. This is a tool based on the `m4` macro processor that turns simple configuration files into complex `sendmail.cf` files. One of the `sendmail` ports comes with support of IDA configuration files, which is why I describe it here. `smail` configuration files are more structured and easier to understand than `sendmail`'s, but don't give the user as much power in tuning the mailer's behavior. However, for small UUCP sites the work required in setting up any of them is roughly the same.

The next two chapters will describe a simple setup for both mailers. It will however not give a full treatment of all options, but will focus on the options the "average" LINUX site will need. These will be sites running UUCP as transport software, as well as sites on a LAN. There are many more options, and you can spend many happy hours in front of your computer configuring the fanciest features.

In chapters 9 and 10, we will give instructions on setting up `smail` and `sendmail` for the first time. The information provided there should suffice to get a UUCP-only leaf site operational.[19] The remainder of the current chapter will give you a short introduction to setting up `elm`, the common mail user agent on many Un⋆xish systems, including LINUX.

[17] Most noticeably, version 2.5 has been in Ian Taylor's official distribution of the source code of Taylor UUCP 1.03. The only thing it has in common with `smail-3.1.28` is the name.

[18] There's a saying that you aren't a real programmer if you haven't hacked a `sendmail.cf` file once; but if you've done it twice, you must be crazy.

[19] A *leaf site* is a site having only one link, without any foreign mail being routed through it.

8.8 Configuring elm

elm apparently stands for "electronic mail" and is one of the more reasonably named Un*x tools. It provides a full-screen interface with a good help feature, so we won't discuss here how to use elm, but will only dwell on its configuration options.

Theoretically, you can run elm unconfigured, and everything works well — if you are lucky. There are however a few options that must be set, although only required on occasions.

When started, elm reads a set of configuration variables from the elm.rc file in /usr/local/lib/elm/. Then, it will attempt to read the file .elm/elmrc in your home directory. This file usually does not exist, but is created when you choose "save options" from elm's options menu.

In both files, the same The set of options for the private elmrc file is also available in the global elm.rc file. Settings in your private elmrc file override those of the global file.

8.9 Global elm Options

In the global elm.rc file, you must set the options that pertain to your host's name. For example, at the Virtual Brewery, the file for vlager would contain the following:

```
#
# The local hostname
hostname = vlager
#
# Domain name
hostdomain = .linus.lxnet.org
#
# Fully qualified domain name
hostfullname = vlager.linus.lxnet.org
```

These options set elm's idea of the local hostname. Although this information is rarely used, you should set these opzions nevertheless. Note that these options only take effect when giving them in the global configuration file; when found in your private elmrc, they will be ignored.

8.10 elm and National Character Sets

Recently, there have been proposals to amend the RFC 822 standard to support various types of messages, such as plain text, binary data, Postscript files, etc.[20] This also allows to notify the recipient if a character set other than standard ASCII has been used when writing the document, for example using French accents, or German umlauts. This is supported by `elm` to some extent.

The character set used by LINUX internally to represent characters is usually referred to ISO-8859-1, although this is the name of the corresponding standard. Any message using characters from this character set should have the following line in its header:

```
Content-Type:  text/plain; charset=iso-8859-1
```

The receiving system should recognize this field and take appropriate measures when displaying the message. The default for `text/plain` messages is a `charset` value of `us-ascii`.

To be able to display messages with character sets other than ASCII, `elm` must know how to print these characters. By default, when `elm` receives a message with a `charset` field other than `us-ascii`[21], it tries to display the message using a command called `metamail`. Messages that require `metamail` to be displayed are shown with an '`M`' in the very first column in the overview screen.

Since LINUX' native character set is ISO-8859-1, however, calling `metamail` is not necessary to display messages using this character set. If `elm` is told that the display understands ISO-8859-1, it will not use `metamail` and display the message directly instead. This can be done by setting the following option in the global `elm.rc`:

```
displaycharset = iso-8859-1
```

Note that you should set this option even when you are never going to send or receive any messages that actually contain characters other than ASCII. This is because people who do send such messages usually configure their mailer to put the proper `Content-Type:` field into the mail header by default, whether or not sending ASCII-only messages.

Secondly, please note that this does not work. Tough luck, isn't it? The problem is that when dislaying the message with its builtin pager, `elm` calls a library function for each character to be printed to determine whether it is printable or not. By default, this function will only recognize ASCII characters as printable, and display all other characters as "`^?`". Usually, this function's behavior can be changed by setting the `LC_TYPE` environment

[20] Detailed in RFC 1049 (Content-Type: header field) and RFC 1154 (Encoding header field).

[21] Or a content type other than `text/plain`, for that matter.

variable to your national language. E.g. setting it to `french` will cause the function to recognize all french special characters as printable. This feature, however, has not been implemented in the standard C library yet.[22]

When sending messages that contain special characters from ISO-8859-1, you should make sure to set some more variables in the `elm.rc` file:

```
charset = iso-8859-1 textencoding = 8bit
```

This makes `elm` report the character set as ISO-8859-1 in the mail header, and send it as an 8 bit value (the default is 7 bit).

Of course, any of these options can also be set in the private `elmrc` file instead of the global one.

[22]As of this writing, `libc` is at release 4.4.1.

Chapter 9

Getting smail Up and Running

9.1 Introduction

This chapter will give you a quick introduction to setting up `smail`, and an overview of the functionality it provides. Being mainly compatible to sendmail in its behaviour, its configuration files are completely different.

The main configuration file is the `/usr/local/lib/smail//config`. You always have to edit this file to reflect values specific to your site. If you are only a UUCP leaf site, you will have relatively little else to do, ever.

Other configuration files that allow to configure routing and transport options may also be used; section 9.6 below will shortly dwell on this.

`smail` is able to handle both UUCP-based mail as well as delivery over SMTP using TCP/IP services. Depending on the amount of traffic you expect to have, you may also consider to queue all incoming mail and process it at regular intervals, or always deliver it immediately. If you choose to queue messages, `smail` will store away incoming mails below `/usr/spool/smail/` and not process them until you explicitly tell it to. Refer to section 9.5 on how to enable spooling.

Even when delivering immediately, `smail` will occasionally put messages into the queue when it finds immediate delivery fails for a transient reason. For SMTP connections, this may be an unreachable host; but messages may also be deferred when the file system is found to be full.

If you want to further dig into this topic, please refer to the manual page `smail(5)`. If it isn't included in the SLS or the `mailpak` distribution, get the source to `smail`.[1]

[1] Maybe I will also release an earlier version of this chapter as a stand-alone reference to `smail` that has

9.2 UUCP Setup

In the SLS `smail` distribution, you will find a file named `config.sample` in `/usr/local/lib/smail/`. Make a copy called `config` and edit it. It contains three statements:

```
#
# visible_domain list the domains (primary first) we say we're in
#
visible_domain=sea.wa.us:uucp
#
# next is the name in our headers
#
visible_name=victrola.sea.wa.us
#
# this is the path to our smarthost (this could be sysa!sysb!sysc etc.)
#
smart_path=quick
```

This is a sample file from Vince Skahan's site. You must change these values, or else Vince will receive all replies to your mail.

The first statement tells `smail` about the domains your site belongs to. Insert their names here, separated by colons. If your site name is registered in the UUCP maps, you should also add `uucp`. When being handed a mail message, `smail` determines your host's name using the `hostname(2)` system call, and checks the recipient's address against this hostname, tacking on all names from this list in turn. If the address matches either one of these names, or the unadorned hostname, the recipient is considered local. Otherwise, the recipient is considered remote.

`visible_name` should contain a single, fully qualified domain name of your site that you want to use on outgoing mails. This name is used when generating the sender's address on all outgoing mail. You must make sure to use a name that `smail` recognizes as local (e.g. the hostname with one of the domains listed in the `visible_domain` attribute). Otherwise, replies to your mails will bounce off your site.

The last statement sets the path used for smart-host routing (described in 8.4). With this sample setup, `smail` will forward any mail for remote addresses to the smart host. The path specified in the `smart_path` attribute will be used as a route to the smart host. Messages will be delivered via UUCP, therefore the attribute must specify a system known

detailed information on writing the `routers`, `directors` and `transports` configuration files.

to your UUCP software. Please refer to chapter 7 on making a site known to UUCP.

Assume your site is named `swim.two.birds`, and is registered with the maps as `swim`. Your smarthost is `ulysses`. Then your config file should look like this:

```
# Our domain names
visible_domain=two.birds:uucp
# Our name on outgoing mails
visible_name=swim.two.birds
# Use this as uucp-name as well
uucp_name=swim.two.birds
# Our smarthost
smart_host=ulysses
```

There's one option used in the above file that we haven't explained yet; this is `uucp_name`. The reason to use this option is this: By default, `smail` uses the value returned by `hostname(2)` for UUCP-specific things such as the return path given in the "`From` " header line. If your hostname is *not* registered with the UUCP mapping project, you should tell `smail` to use your fully qualified domain name instead.[2] This can be done by adding the `uucp_name` option to the `config` file.

There is another file in `/usr/local/lib/smail/`, called `paths.sample`. It is an example what a `paths` file might look like. However, you will not need one unless you have a mail link to more than one site. If you do, however, you will have to write one yourself, or generate one from the Usenet maps (see section 8.5). Please refer to section 9.8 for an explanation of the `paths` file.

9.3 Setup for a LAN

If you are running a site with two or more hosts connected by a LAN, you will have to select one host that handles your UUCP connection with the outside world. Between the hosts on your LAN, you will most probably want to exchange mail via SMTP. Assume we're back at the Virtual Brewery again, and `vstout` is set up as the UUCP gateway. This section will show you an example how to set up the configuration files. To make your setup work, you should also read the following section 9.4.

The configuration files used for the different hosts are as follows: on all hosts except

[2] The reason is this: Assume your hostname is `monad`, but is not registered in the maps. However, there is a site in the maps called `monad`, so every mail to `monad!root`, even sent from a direct UUCP neighbor of yours, will wind up on the other `monad`. This is a nuisance for everybody.

for the UUCP gateway (vstout), the smart_path attribute should point to vstout. On vstout itself, it should point to the real smart host that routes the Brewery's mail, moria.

In a networked environment, it is best to keep all user mailboxes on a single file system, which is NFS-mounted on all other hosts. This allows users to move from machine to machine, without having to move their mail around (or even worse, check some three or four machines for newly-arrived mail each morning). Therefore, one might also want to make sender addresses independent from the machine the mail was written on. Janet User, for example, could now specify janet@linus.lxnet.org instead of janet@vale.linus.lxnet.org. To achieve this, all hosts use the domain name as visible_name attribute, instead of their proper hostnames. Of course, one of the machines on linus.lxnet.org will have to recognize the domain's name as one of its own names. One will naturally choose the UUCP gateway for this.

The standard config file for hosts other than vstout looks like this:

```
# Our domain:
visible_domain=linus.lxnet.org
# What we name ourselves
visible_name=linus.lxnet.org
# Smart-host routing: via SMTP to vstout
smart_path=vstout
smart_transport=smtp
```

On the UUCP mail gateway vstout, the config file would look like this:

```
# Our hostnames:
hostnames=linus.lxnet.org:vstout.linus.lxnet.org:vstout
# What we name ourselves
visible_name=linus.lxnet.org
# in the uucp world, we're known as linus.lxnet.org
uucp_name=linus.lxnet.org
# Smart transport: via uucp to moria
smart_path=moria
smart_transport=uux
# we're authoritative for our domain
auth_domains=linus.lxnet.org
```

This config file uses a different scheme to tell smail the list of local hostnames. Instead of giving it a list of domains and letting it find the hostname with a system call, it gives an explicit list. This is because we want to include the domain name in the list. Now, `janet@linus.lxnet.org` is a valid address, provided there is a user account for `janet` (or an alias).

The `auth_domains` variable names the domains for which `vstout` is considered to be authoritative. That is, if `smail` receives any mail addressed to *host*`.linus.lxnet.org`, but `host` is not a local machine, it rejects the message and returns it to the sender. If this entry isn't present, any such message will be sent to the smart-host, who will return it to `vstout`, and so on until it is discarded for exceeding the maximum hop count.

9.4 Invocation and Command Line Options

To be able to send and receive mails, you should first make sure you have a command named `rmail`. When using `smail`, you should make this a link to the `smail` binary. For example, if you keep the latter in `/usr/local/bin`, go to the `/bin` directory and type

```
# ln /usr/local/bin/smail rmail
```

When composing and sending a mail message with a user agent like `elm`, the message will be piped into `rmail` for delivery, with the recipient list on the command line. The same happens with mail coming in viaa UUCP link.

There are, however, a number of other names `smail` may be linked to. Among them are `rsmtp`, which is analoguous to `rmail`, but is for receiving SMTP batches, and `smtpd` which makes it act as SMTP server when invoked from the `inetd` server.

For queue operation, it may also be invoked as `mailq`, which displays the mail queue, and `runq`, which makes it process the queue.

Finally, it may be linked to `sendmail`, which makes it behave in a way compatible to the original Berkeley `sendmail`.

All of these commands accept a common set of command line options; only `rmail` and `rsmtp` are restricted in what they accept. The list below does not attempt to be complete.

`-bd` Run in daemon mode. With this option, `mail` will put itself in the background, and wait for a connection to occur on the `smtp/tcp` port. When a connection occurs, it forks and conducts an SMTP conversation with the peer process.

`-bp` List information about the messages currently in `smail`'s queue directories. This option is on by default for `mailq`.

`-bS` Accept SMTP commands on standard input, but don't produce SMTP replies. This option is on by default for `rsmtp`.

`-v` Be verbose. The option may be followed by a number, indicating the level of verbosity. This option is equivalent to `-d`.

`-q`*interval* When running `smail` in daemon mode (`-bd` option), this specifies the intervals at which `smail` will run the queue.

If *interval* is a single number, it is taken to be the interval in seconds. You may, however, also use values like `2h30m`, which denotes 2 and a half hours. If *interval* is omitted altogether, a single queue run is performed, which is equivalent to invokig `runq`.

`-bR` The manpage says on this option: "Enter the hostile domain of giant mail messages, and RFC standard scrolls. Attempt to make it down to protocol level 26 and back." If you know what Rogue is, this will probably sound familiar to you.

For delivering mail over UUCP link, the above is almost enough. You should only make sure to run the queue once or twice a day to flush out any deferred messages. For example, you may add an invocation of `runq` to the super-user's crontab.

For SMTP to work, you should make sure you have the following entry in your `/etc/services` file:

```
smtp            25/tcp          # Simple Mail Transfer Protocol
```

To serve incoming SMPT requests, you either need to set up `inetd` to manage the `smtp` port, or run `smail` in daemon mode. If you want to use `inetd`, your `/etc/inetd.conf` file should contain

```
smtp    stream  tcp nowait  root  /usr/bin/smtpd smtpd
```

Remember you have to make `inetd` re-read this file by sending a `HUP` signal to `inetd` after making these changes.

If mail traffic inside your LAN is high, you may instead start `smail` in daemon mode (you should then make sure any `smtp` entry in `/etc/inetd.conf` is commented out). To start a `smail` daemon, you invoke it from `/etc/rc.d/rc.inet2` by

```
/usr/local/bin/smail -bd -q15m
```

Note that you have to configure SMTP on each system one way or other; you may also mix these two methods for different hosts.

9.5 Miscellaneous config Options

There are quite a number of options you may set in the config file, which, although useful, are not essential to running smail, and which we will not discuss here. Instead, we will only mention a few that might be especially useful:

error_copy_postmaster
: If this boolean variable is set, any error will generate a message to the postmaster. Usually, this is only done for errors that are considered to be due to a faulted configuration. The variable can be turned on by putting it in the config file, preceded by a plus ('+').

max_hop_count
: If the hop count for a message (i.e. the number of hosts already traversed) equals or exceeds this number, attempts to remote delivery will result in an error message being returned to the sender. This is used to prevent messages from looping forever. The hop count is generally computed from the number of Received: fields in the mail header, but may also be set manually using the -h option on the command line.

  The variable defaults to 20.

delivery_mode
: smail knows of three different modes of delivery; being in the foreground (immediate processing of incoming messages), in the background, (message is delivered by a child of the receiving process, with the parent process exiting immediately after forking), and queued. In this mode, the message is moved to the queue directory (generaly /usr/spool/smail//messages), from where it is picked up and processed further during a later queue run. This behavior can be determined by setting the variable delivery_mode to one of foreground, background, or queued. Incoming mail will always be queued regardless of this option if the boolean variable queue_only is set.

  If you turn on queuing, you have to make sure the queues are checked regularly; probably every 10 or 15 minutes. If you run smail as daemon, you have to add the option `-q10m`.

There is also a tool for checking the queue; this is called `mailq` and is a link to `smail`, too.

9.6 Mail Delivery

`smail` splits up mail delivery into three different tasks:

router — This is the task that resolves anything looking like an address into something else looking like an address. The difference is that after the router does its job, it is known to which host the message must be sent and which transport must be used. The destination may also be the local host.

director — Local addresses are given to directors which resolve any forwarding or aliasing. For example, the address might be an alias, a mailing list, or the user might want to forward her mail to another address. This may require additional routing.

transport — If a router has decided that a given address is remote, the message must be forwarded to the respective site. Depending on the nature of the links, different kinds of transport software are needed.

Local addresses have to be delivered accordingly.

With `smail`, one can configure these tasks in many ways. For each of them, a number of drivers are available, from which you can choose those you need. You describe them to `smail` in a couple of files, namely `routers`, `directors`, and `transports` located in `/usr/local/lib/smail/`. If these files do not exist, reasonable defaults are assumed that should be suitable for many sites that either use SMTP or UUCP for transport. If you want to change `smail`'s routing policy, or modify a transport, you should get the sample files from the `smail` source distribution,[3] copy the sample files to `/usr/local/lib/smail/`, and modify them according to your needs. Possibly, some future releases of `smail` binaries may include these files.

[3] The default configuration files can be found in `samples/generic` below the source directory.

9.7 Routing Messages

To find out which host to forward a message to, `smail` hands the destination address to a number of router drivers. These may be specified using the `routers` file; if this file does not exist, a set of default routers are used.

Before giving the address to the routers, it is split up into a *target*, which is the machine to send the message to, and the *remainder*. For an address like `foo!bar!user`, the target would be `foo`, and `bar!user` the remainder.

If a match is found by a router, a new target and remainder are generated, which are given to the transport. The target (also *next host*) is the address the transport is to deliver the message to, and the rmainder (also *next address*) is the envelope address to be given to the mail software on the remote host. In the above example, `smail` might find out that `foo` is to be reached through the path `ernie!bert`, using UUCP. It will then generate a target of `ernie`, and a remainder of `bert!foo!bar!user`.

When using the default setup, `smail` performs the following actions to determine how to deliver a message:

- If the target host address is one of the local hostnames recognized by `smail`, the message is handed to the director module, see section 9.9.
- Next, the router module checks if it can resolve the destination host address using the `gethostbyname(3)` or `gethostbyaddr(3)` library call. This will match any hostnames found by the resolving routines, either through lookups in `/etc/hosts`, or by querying the nameserver. (Refer to section 3.9 on how to configure the resolver.)

  IP addresses matched may be written as either a dotted quad, or enclosed in square brackets (e.g. `[192.72.2.1]`).

  If a host address is matched by this router, the message will be delivered over SMTP, unless the address is found to refer to the local host (for example `127.0.0.1`). In this case, it is handed to the director module.

  If your machine is on the Internet, these routers are not what you are looking for, because they do not support MX records.
- Next, `smail` will try to look up the target host (minus any trailing `.uucp`) in the pathalias database, if this exists. `smail` expects this in the `paths` file in `/usr/local/lib/smail/`.

  Mail to an address matched by this router will be delivered using UUCP, using the path found in the database.

- The host address (minus any trailing `.uucp`) will be compared to the output of the `uuname` command to check if the target host is in fact a UUCP neighbor. If this is the case, the message will be delivered using the UUCP transport.

  For two reasons, however, it is not generally a good idea to use this driver. For one, this used to fail occasionally with an error message that `uuname` returned a result code of 255. I do not know the reason for this. The second is that you may have more sites listed in your `Systems` file than you actually have mail links with. These may be sites you only exchange news with, but even more likely is that it contains information on sites you occasinally download files from via anonymous UUCP, but have no traffic with otherwise.

  To work around this, you may add specific paths for these sites to your `paths` file, or, if you have a `routers` file in `/usr/local/lib/smail/`, comment out this driver altogether.

- If the address has not been matched by any of the above routers, it will be delivered to the smart host. The path to the smart host as well as the transport to be used are set in the `config` file.

As described above, the default settings of `smail` are not really suitable for use on the Internet. You might want to use `sendmail` instead, but if you do want to use `smail`, you have to recompile it yourself. You may use the configuration files from Vince Skahan's `newspak` distribution. To compile in the BIND driver, you have to set the `DRIVER_CONFIGURATION` macro in the `EDITME` file to `arpa-network`. To make use of this driver, edit the `routers` file, commenting out the `inet_hosts` router that uses the `gethostbyname(3)` driver, and uncomment the one that uses the BIND driver instead.

9.8 The pathalias database

`smail`
expects to find the pathalias database in the `paths` file below `/usr/local/lib/smail/`. This must be a sorted ASCII file that contains entries which map destination site names to UUCP bang paths. The file has to be sorted because `smail` uses binary search for looking up a site. You may not insert comments in this file, and the site name must be separated from the path using a TAB. Pathalias databases are discussed in somewhat greater detail in section8.5.

If you generate this file by hand, you should make sure that you include all legal names for a site. For example, if a site is known by both a plain UUCP name and a fully qualified domain name, you have to add an entry for each of them. You best sort this file by piping it through the `sort(1)` command.

If your site is only a leaf site, however, then no `paths` file should be necessary at all: just set up the smart host attributes in your `config` file, and leave routing to your mail feed.

9.9 Delivering Messages to Local Addresses

This section covers the addressing modes available for local addresses.

The standard case is where a local address names a user on the machine to whose mailbox the message is to be delivered. But a local address may also be resolved to one or more remote ones if the user wants to have his mail forwarded to another site, or if the address was that of a mailing list.

Apart from "normal" addresses of local users, `smail` can handle other types of local message destinations, which will also be called addresses. For example, some directors will produce file names, pipe commands, and file inclusions.

The `local` driver Messages for local users that have not been redirected otherwise (for example through forwarding) will be appended to the user's mailbox, `/var/spool/mail/`*user*. This file must be writable by group `mail`. If it does not exist, it is created with mode `660`, owned by the user with a group of `mail`.

A *file name* is anything that begins with a slash ('`/`') or a tilde ('`~`)'. The latter, of course, is only possible if the filename was taken from a `.forward` file, or a forwarding entry in the mailbox. When delivering to a file, `smail` appends the messages to the file, creating it if necessary.

A *pipe command* may be any Un$\star$x command, preceeded by the pipe symbol ('`|`'). This causes the message to be piped into the command. If the invocation contains white space, it has to be enclosed in double quotes ('`"`'). Due to the security issues involved, care is taken not to execute the command if the address has been obtained in a somewhat dubious way (for example, if the alias file from which the address was taken was writable by everyone).

When delivering to a pipe command, `smail` gives the command along with the arguments to the shell (minus the leading '`|`'), and passes it the message on standard input.

The environment passed to the child process consists of the `TZ` variable as obtained from the calling process, and the following variables defined by `smail`:

`SENDER` contains the sender's address.

`BASENAME` and `SPOOL_FILE`
: contain the base and file name of the spool file.

`GRADE` amd `MESSAGE_ID`
: contain the message grade and the message id, respectively.

`PATH` and `SHELL`
: contain "`/usr/bin`" and "`/bin/sh`", respectively.

`PRIMARY_NAME`, `UUCP_NAME`, and `VISIBLE_NAME`
: contain the values of the corresponding variables as defined in the `config` file.

A *file inclusion* is an address of the form `:include:`*filename*, and may be used in the `aliases` database and mailing list. A file inclusion will cause `smail` to take the contents of the file as a comma-separated list of addresses to deliver the message to. The file may contain comments wherever white space is allowed; a comment starts with a hash sign ('`#`') and is terminated by the next new-line.

9.9.1 Local Users

The most common case for a local address is to denote a user's mailbox. This mailbox is located in `/var/spool/mail/` and has the name of the user. It is owned by her, with a group of `mail`, and is mode `660`.

She may, however, redirect her mail by having it forwarded to an alternative address (see below). To skip these checks, the user name may be prefixed with `real-`, in which case the message is delivered to the mailbox directly.

There are two addresses `smail` relies on to exist. These are `MAILER-DAEMON` and `Postmaster`. To secure delivery to these addresses, smail contains two implicit aliases which are used as a last resort. These map `MAILER-DAEMON` to `Postmaster`, and `Postmaster` to `root`, respectively. You may override this by supplying aliases for them in the `aliases` file.

9.9.2 Alias Files

`smail` is able to handle alias files compatible to those known by Berkeley's `sendmail`. Entries in the alias file may have the form

alias: *recipient*, ...

recipients is a comma-separated list of addresses. These may be local and remote addresses, as well as those described above. A recipient list may be continued across newlines if the next line begins with a TAB.

There is a special feature that allows to handle mailing lists from the alias file: if you specify "`:include:`*filename*" as recipient, `smail` will read the file specified, and substitute its contents as a list of recipients.

The main aliases file is `/usr/lib/aliases`. If you choose to make this file world-writable, `smail` wil not deliver any messages to shell commands given in this file. A sample file is given below:

```
# linus.lxnet.org Aliases file
usenet: phil
netadmin: janet
postmaster: janet
mailer-daemon: janet
# The development mailing list.
development: joe, sue, mark, biff
        /usr/local/lib/log/development
owner-development: joe
# Announcements of general interest are mailed to all
# of the staff
announce: :include: /usr/local/lib/smail/staff,
        /usr/local/lib/log/announce
owner-announce: root
# Send facsimiles
fax: "|/usr/local/bin/sendfax"
```

If an error occurs while delivering to an address generated from the `aliases` file, `smail` will attempt to send a copy of the error message to the "alias owner". For example, if delivery to `biff` fails when delivering a message to the `development` mailing list, a copy of the error message will be mailed to the sender, as well as to `postmaster` and `owner-development`. If the address does not exist, no additional error message will be generated.

9.9.3 Mailing Lists

Instead of using the `aliases` file, mailing lists may also be managed by means of files in the `lists` directory below `/usr/local/lib/smail/`. A mailing list named `nag-bugs` is described by the file `lists/nag-bugs`, which should contain the member's addresses, separated by commas. The list may be given on multiple lines, with comments being introduced by a hash sign ('`#`').

For each mailing list, a user (or alias) named `owner-`*mlist* should exist; any errors occurring when resolving an address are reported to this user. Also, it is used as the sender's address on the outgoing messages. (In the `Sender:` header field.)

9.9.4 Forward Files

`smail` supports two ways a user may request to have her mail forwarded to a different address. This may be done by putting

`Forward to` *recipient*,...

in her mailbox file (i.e. `/var/spool/mail/`*user*). Of this file, only the first line will be inspected. Alternatively, she might create a `.forward` file in her home directory which contains a comma-separated list of recipients, too. Unlike the first variety, all lines are read and interpreted.

Note that any type of address may be used. Thus, a practical example of a `.forward` file for vacations might be

```
real-janet, "|vacationreply"
```

The address `real-janet` delivers the incoming message to `janet`'s mailbox nevertheless, while the `vacationreply` command may return a short notification to the sender (taking the sender's address from the `SENDER` environment variable).

9.10 UUCP-based Transports

There are a number of transports compiled into `smail` that utilize the UUCP suite. Usually, this results in an invocation of `rmail` on the next host in the path giving it the message on standard input and the envelope address on the command line.

When handing a message to the UUCP transport, `smail` converts the target address to a UUCP bang path. For example, `user@host` will be transformed to `host!user`. Use of the '`%`' address operator is preserved, so `user%host@gateway` will become `gateway!user%host`. However, `smail` will never generate such addresses itself.

Alternatively, `smail` can produce SMTP batches to be fed to the `rsmtp` command on the remote site. These contain the commands the local mailer would issue if a real SMTP connection had be established. It is used in store-and-forward (e.g. UUCP-based) networks to protect the mail envelope from being garbled by the shell (see footnote 4 in section 8.2).

The BSMTP transport is not available by default, but may be enabled for some UUCP links using so-called *method* files (please refer to the `smail(5)` manual page for details). If you only have one UUCP link, and use the smart host router to get your mail over this link, you enable sending SMTP batches by setting the `smart_transport` configuration variable to `uusmtp` instead of `uux`.

To receive SMTP batches over UUCP, you must make sure that you have the unbatching command the remote site sends its batches to. If the remote site uses `smail`, too, you need to make `rsmtp` a link to `smail`. If the remote site runs `sendmail`, you should additionally have a shell script named `bsmtp` that does a simple "`exec rsmtp`".

9.11 SMTP-based Transports

`smail` currently supports a SMTP driver to deliver mail over TCP connections.[4] It is capable of delivering a message to any number of addresses to one single host, with the hostname being specified as either a fully qualified domain name that can be resolved by the networking software, or in dotted quad notation enclosed in square brackets. Generally, addresses resolved by the gethostbyname and gethostbyaddr router drivers, respectively, will be delivered to the TCPSMTP transport.

The SMTP driver will attempt to connect to the remote host immediately through the `smtp` port as listed in `/etc/services`. If it cannot be reached, or the connection times out, delivery will be reattempted at a later time.

Delivery on the Internet requires that routes to the destination host be specified in the *source routing* format described in 8.4, rather than as a bang path.[5] `smail` will therefore transform `user%host@gateway`, where `gateway` is reached via `host1!host2!host3`, into the source-route address `<@host2,@host3:user%host@gateway>` which will be sent as the message's envelope address to `host1`.

9.12 Hostname Qualification

Sometimes it is desirable to catch unqualified hostnames (i.e. those that don't have a domain name) specified in sender or recipient addresses, for example when gatewaying between two networks, where one requires use of fully qualified domain names. This is necessary on the Internet (where unqualifed hostnames should be mapped to the `.uucp` domain by default),

[4] The authors call this support "simple". For a future version of `smail`, they advertise a complete backend which will handle this more efficiently.

[5] However, the use of routes in the Internet is discouraged altogether.

but may also be also useful in the UUCP zone when users are known to use the bare hostname, although the site in question is only known by a qualified name.

The `qualify` file contains information used by `smail` to convert any addresses containing bare hostnames to fully qualified domain names by mapping host names onto their default domain. Qualification is performed on all addresses of the form *user@host* and *user%host@gateway* for locally generated mail.

Entries in the `qualify` file are single lines, consisting of

hostname domain

where *hostname* begins in column one. Such an entries states that *hostname* should be considered part of *domain*. A line containing a hash sign (`#`) as its first non-white character is considered a comment line. Entries are searched in the order they appear in.

A special hostname of "*" matches any hostnames, thus enabling to map all hosts not mentioned before into a `default` domain. It should only be used as the last entry.

The virtual brewery will not have a very big `qualify` file. It qualifies their local hosts, and maps any remaining hostnames to the `.uucp` domain. This is to make sure no mailer rejects their mail on the ground that the recipient's address contain an unqualified name.

```
# The Virtual Brewery
# /usr/local/lib/smail/qualify, last changed Feb 12, 1993 by okir
#
vstout       .linus.lxnet.org
lager        .linus.lxnet.org
vale         .linus.lxnet.org
vchianti     .linus.lxnet.org
vbardolino   .linus.lxnet.org
...
*            .uucp
```

Chapter 10

Installing and Using sendmail

10.1 Introduction

As already described above, `sendmail` relies on a configuration file named `sendmail.cf`, which is located in `/etc`. Writing such a file, or even adapting such a file by hand, is not really easy. To levitate the situation for sites that want to run `sendmail` but don't have a real `sendmail` buff handy, there have been various solutions that supply a set of configuration files for several standard setups, which may be modified more or less easily.

In 1998, Lennard Lovstrand developed a kit to produce `sendmail.cf` using the `m4` macro processor, and added a few features to `sendmail`. This has become known as the IDA `sendmail` enhancement kit, named after the Linkoping University's Computer Science Institute he was working at then.

There is a `sendmail` port available for LINUX that includes IDA support. This port is due to Rich Braun.[1] and can be found at `tsx-11.mit.edu` as

```
sendmail-5.56c+IDA-1.4.4-beta.tar.Z
```

Another `sendmail` port is available, too, which uses a different setup procedure than IDA.

IDA is a very sophisticated configuration package that provides a variety of mail routing and gatewaying facilities. It is not the intention of this book to cover all aspects of IDA, let alone `sendmail` itself. If you are interested in details, please refer to the original IDA documentation. A good place to look is also the `sendmail.cf` file itself. Extra comments may be included into the file by defining `M4COMMENTS` in the IDA file (see below). There is

[1] To be reached at `richb@jti.com`.

also a Postscript description included, which describes all options and extensions. A good place to start if you are interested in `sendmail` itself is probably Craig Hunt's "TCP/IP Network Administration" (see [Hunt92]); it has a separate chapter on `sendmail`.

10.2 Installing sendmail

To install `sendmail`, unpack the tar file in some safe place, and move the files to their proper places. To preserve ownership and mode, which is critical for most files, do this as `root`, and either use the `mv(1)` command, or give the `-p` option to `cp(1)`. Do not throw awawy the source tree (`usr/local/src`), because you will still need it.

Then create the directories `/usr/spool/mqueue`, which is to hold the mail queue, and `/usr/local/lib/mail`, which we will refer to as the "lib" directory below. Both should be owned by `root`, and should have a mode of `755`. Go to the `ida/lib` below the source directory, and copy `sendmail.hf` to the lib directory. The other files serve as examples of IDA support files, which will be explained below.

If you want your `sendmail` installation to act as SMTP server, too, you have to decide whether you want it to be managed by `inetd`, or run as a separate daemon. The latter is ony advisable in case of considerable traffic.

To run `sendmail` as a daemon, put the following into `/etc/rc.d/rc.inet2`:

```
if [ -x /usr/lib/sendmail ]; then
        /usr/lib/sendmail -bd -bq15m
fi
```

This will start `sendmail` as a daemon process, which processes the mail queue every 15 minutes. Make sure you do not have any `smtp` service defined in your `inetd.conf` file.

If you want `sendmail` to be managed by `inetd` instead, you have to make sure to define a `smtp` service in `inetd.conf`:

```
smtp   stream  tcp  nowait  daemon  /usr/lib/sendmail sendmail -bs
```

With this setup, you still have to make sure the mail queue is processed periodically. This can be done by adding the following line to `root`'s crontab:

```
10   *    *    *    *    /usr/lib/sendmail -q >/dev/null
```

The next step is the most important one: creating the sendmail configuration file, which is explained in the following section.

10.3 Creating sendmail.cf

The IDA configuration set consists of a bunch of master files, located in **ida/cf** below the **sendmail** source directory. (When untarring the distribution from the root directory, this should go somewhere below **/usr/local/src**).

To produce a configuration file for your site, you have to go to this directory, and create a **m4** macro file that sets a few macros to reflect your site's requirement. The best way to start is to copy the file local.m4 to *yoursite*.m4, where you replace *yoursite* with you site's name (what else), and edit it. Below, a sample IDA file is shown for the site **swim.two.birds**, which has a single link to a host named **ulysses**:

```
# Sendmail configuration file for swim.two.birds
define(ALIASES, LIBDIR/aliases)
define(LOCAL_MAILER_DEF, mailers.sco)
dnl ### Postmaster gets copy of bounced mail. ###
define(POSTMASTERBOUNCE)
define(PSEUDODOMAINS, BITNET UUCP)
define(PSEUDONYMS, swim.two.birds localhost.two.birds)
define(DEFAULT_HOST, swim.two.birds)
dnl ############# Local UUCP names ##############
define(UUCPNAME, swim)
define(UUCPNODES, |uuname|sort|uniq)
define(BANGIMPLIESUUCP)
define(BANGONLYUUCP)
dnl ########## Our smart relay system ###########
define(RELAY_HOST, ulysses)
define(RELAY_MAILER, UUCP-A)
dnl ######### Define dbm lookup tables ##########
dnl ## pathalias database
dnl define(PATHTABLE, LIBDIR/pathtable)dnl
dnl ## target-transport mapping
dnl define(MAILERTABLE, LIBDIR/mailertable)dnl
dnl ## Mapping between DNS and UUCP zone
dnl define(DOMAINTABLE, LIBDIR/domaintable)dnl
dnl define(UUCPXTABLE, LIBDIR/uucpxtable)dnl
dnl ########## Define this for verbose #########
dnl ########## comments in sendmail.cf #########
dnl # define(M4COMMENTS)dnl
```

```
include(Sendmail.mc)
```

The file mainly consists of a number of `define` statements. These do what they say: They define a macro (the first argument), giving it the optional second argument as value. Note that, depending on the version of `m4` you are using, the macro processor might issue wrnings about `define` statements with only one argument. The second important command is `dnl`, which means "delete to new line", and starts a comment. Thus, hash signs have no special meaning to `m4`, so that they will also appear in the output file. The last statement in the IDA file must be the `include` statement, which includes the `sendmail` master file.

Looking at the file, you may find that a macro named `LIBDIR` is used in filename definitions, without the macro itself being defined anywhere. This macro is passed to `m4` on the command line when being invoked from the `Makefile`, and is given a value of `/usr/local/lib/mail`. If you feed the IDA file to `m4` manually, you should either pass it a value for `LIBDIR`, else the `Sendmail.mc` file will assume a default value.

The meaning of most of the other macros will be discussed below. If you need more information about the facilities offered by IDA, either refer to the Poscript documentation enclosed in the package, or read the comments in the `Sendmail.mc`.

Once you have adapted all macros to reflect the settings of your system, you build the `sendmail.cf` file from it by doing

```
# make yoursite.cf
# install -o root -g root -m 644 yousite.cf /etc/sendmail.cf
```

This creates the file *yoursite*`.cf`, which you then copy to `/etc/sendmail.cf`. For performance reasons, `sendmail` processes this file only once, and then stores an internal representation in a file called `sendmail.fc` in the `/etc` directory. This is called the *frozen* configuration file. It is built by issuing

```
# sendmail -bz
```

10.4 Invocation and Command Line Options

Unlike `smail`, `sendmail` comes along with a couple of binaries required for it to work. One of these is the `lmail` binary, the local mailer used to deliver messages to a user's mailbox. It is located in `/usr/bin`. Another is `rmail`, which is needed when receiving mail via UUCP. It does little more than executing `sendmail` itself. You can find it in `/bin`.

The `sendmail` binary itself is located in `/usr/lib`. Which may seem a bit odd, but this has been the standard for roughly the last few million years or so.

In the sections above, you have already come across the most important command line options you need for day-to-day operations. We will sum them up here again. There are also a number of "alias" names `sendmail` may be linked to, which turn on some specific mode of operation. They will be mentioned alongside with the equivalent command line options.

`-bd` Start up in daemon mode. This puts `sendmail` in the background, where it listens to the SMTP port, and periodically runs the mail queue when given the `-q` option.

`-bq`*interval* Run the mail queue every *interval*. The argument is made up of a number and a unit: `h` for hour, and `m` for minute. You may also combine this to `1h30m`.

`-bq` If specified all by itself, `sendmail` will unconditionally run the mail queue once and exit. This is equivalent to invoking `sendmail` as `runq`.

`-bs` Perform an SMTP session over standard input.

`-bb` Process standard input as a BSMTP batch. This is equivalent to invoking `sendmail` as `bsmtp`.

`-bi` Build the aliases `dbm` database. See section 10.8.2 below. This is equivalent to invoking `sendmail` as `newaliases`.

`-bz` Build the frozen configuration file, `/etc/sendmail.fc`.

`-bp` Print all jobs currently in the mail queue and exit. This is equivalent to invoking `sendmail` as `mailq`.

10.5 Routing with `sendmail`

To be able to configure `sendmail` properly, you first have to know a little about how it routes a message.

Routing in sendmail is essentially performed by a set of rewriting rules in the `sendmail.cf` file. These rules perform some sort of text matching on the recipient address, look up the address in various tables, and try to associate a suitable *mailer* with it. "Mailer" is the generic term for a delivery or forwarding service in all `sendmail` documentation, analogous to "transport" for `smail`. There's some wizardry involved in parsing the address etc., but we will nevertheless explain what it basically does. The various tables being used will be explained in detail below, as are the different mailers chosen by the routing alogrithm.

Please note that the description *only applies to IDA 1.4.4* included in the `sendmail` distribution by Rich Braun; other `sendmail` or IDA versions may have entirely different configuration files, and hence might have a completely different approach to address resolution. Also note that this does not discuss the various resolving procedures for DECnet and XNS, which are also supported.

When resolving an address `sendmail` takes the following steps:

1. If the target hostname is a name of the local host, the user name is checked against the `aliases` file. If it is aliased to one or more other addresses, the process is repeated with these new addresses. Otherwise it is delivered locally.

2. If the entire address, including the remote hostname, is found in the `aliases` file, it is replaced by the aliased address, and the whole process is repeated with the new address.

3. The target host is now definitely remote. `sendmail` will look it up in the mailer table, a file that maps domain names to mailer-host pairs. If a match is found, the message is delivered to the host specified, using the associated mailer.

4. The resolver library will then be called to resolve the hostname, unless it is in one of the pseudo-domains `.uucp` or `.BITNET`. This may be configured to either look up the name in `/etc/hosts`, or query the name server, taking `MX` records into account. For details, please refer to section 10.7.3.

   If the hostname could be resolved to an IP address, the message will be delivered via the `TCP` mailer, using the canonical hostname as returned by the name server as the envelope address.

5. Failing this, the target name is looked up in the UUCP translation table that maps fully qualified domain names to (possibly different) UUCP names. If the resulting UUCP name (or the original name if no match was found) is a direct UUCP neighbor, it will be sent through the `UUCP` mailer.

6. If the target is found in the pathalias database, the message will be sent to the first node in the path. The mailer to be used will be determined by repeating the whole process with the new target host.

7. If the message could not be routed using one of the above schemes, it is sent to the smart host, if defined.

10.6 Setting the Site Name

First, let's start with the macros that set your site's names: `DEFAULT_HOST` gives your host's site name, as should be used as the sender host name on all outgoing mail. Any other names your site is known by — except your UUCP name — should be given in the `PSEUDONYMS` macro. Host names are separated by space or tabs. Note that this list should also include `localhost`, possibly qualified by your site's domain.

If your site is registered in the UUCP maps, set `UUCPNAME` to its official UUCP name (sans trailing `.uucp` domain). If you are not a registered UUCP site, uncomment the `define` statement by preceding it with `dnl`.

10.7 Routing Options

IDA offers a number of options to control the different routing schemes described above in section 10.5. We will describe them below, each in a different subsection.

10.7.1 Alias Options

Alias lookups are enabled by giving setting the `ALIASES` macro to the `aliases` file name. Please refer to section below for information of the file format.

10.7.2 Mailer Table Options

Mapping of target hostnames to mailers is enabled by defining the `MAILERTABLE` macro in the IDA configuration file. It must be set to the name of the mailer table file. We will not describe the use of mailer tables in this document; please refer to the original IDA documentation.

10.7.3 Address Resolution

If you want to deliver mail over a TCP/IP network, be it on the Internet, or in an isolated network, you have to make sure the resolver library is set up properly. You may have done this already when setting up services like the printer daemon, but we will go through the necessary steps nevertheless.

When trying to resolve a hostname to an IP address, **sendmail** uses the `gethostbyname(3)` and `res_search(3)` library calls. Depending on the options selected in the files `host.conf` and `resolv.conf`,[2] they get their information from either one or more name servers, or from the local `/etc/hosts` file.

When a name server is in use at your network, or even on your local machine, you should enable DNS lookups in `host.conf` using the option

```
order bind hosts
```

The IP address of the name server(s) to be queried must be given in `resolv.conf`, using the `nameserver` statement. For example, assuming a name server has been installed on `vlager` at the Virtual Brewery, this file would read

```
# /etc/resolv.conf for the Virtual Brewery
domain        linus.lxnet.org
nameserver    192.72.1.1
```

If your machine is a standalone site in terms of TCP/IP networking, or you are in an isolated network without Internet connectivity, you may probably not run a name server. To be able to run **sendmail** nevertheless, you must configure the resolver library accordingly. First, the `order` statement in `/etc/host.conf` should only use the `/etc/hosts` file:

```
order hosts
```

Second, to keep **sendmail** from searching for unknown name servers when looking for `MX` records, you also have to make sure there is no `nameserver` statement in `resolv.conf`.

10.7.4 UUCP Routing

If your site has several UUCP links, you should configure the appropriate UUCP options. To enable pathalias routing, the location of the pathalias database must be given using the `PATHTABLE` macro. If this macro is not set, no pathalias routing will be used.

To give **sendmail** a list of all your neigboring UUCP sites, use the `UUCPNODES` macro. The default value in the sample file, "`|uuname|sort|uniq`", uses the sorted output of the **uuname** command to obtain this list. Note that if you use this setting, you have to update

[2] Described in section 3.9.

`sendmail`'s frozen configuration file every time you add or delete system from your UUCP configuration files (see section 10.3).

As said before in section 9.7, use of the `uuname` output is debatable, so you might either hard-code a list of all your UUCP neighbors here, or define the macro to the (absolute) name of a file where you keep a list of all UUCP neighbors you exchange mail with.

There are a couple of other options that control mapping of UUCP names to the `.uucp` domain (`BANGONLYUUCP` and `BANGIMPLIESUUCP`), as well as the treatment of hybrid addresses mixing '`!`' and '`@`'. We will not discuss them here; please refer to the IDA documentation for details.

By default, any messages routed using the pathalias database will be delivered using the `UUCP` mailer. If you prefer to use a different mailer, for example `UUCP-A`, you may change this by setting the `UUCPMAILER` macro to `UUCP-A`[3] or any other UUCP-based mailer.

10.7.5 Smart Host Routing

To enable smart host routing, you have to set the macros `RELAY_HOST` and `RELAY_MAILER`, respectively. `RELAY_HOST` specifies the name of the mail relay you use for smart host routing. This may either be a fully qualified domain name or a UUCP name, depending on the transport you use. The transport is selected using the `RELAY_MAILER` macro. For a UUCP link, you should enter `UUCP-A` here, and `TCP` for a TCP/IP link.

10.8 `sendmail` Support Files

This section describes the various tables used by the `sendmail` configuration described above. These files reside in `/usr/local/lib/mail`, and all of them are so-called `dbm` databases.

All these files are optional, except for the alias database. Some of them will rarely be used by minor sites, so we will only mention them shortly.

10.8.1 Creating `dbm` Databases

`dbm` is a database format that allows fast lookups due to hashing techniques.[4] It consists of a list of key-value pairs, both usually being ASCII strings.

[3] See section 10.9.3 below.

[4] In fact, there is not one single `dbm` format, but every implementation — and there are several — use a slightly incompatible variant.

A dbm database is created from some master file, usually a human-readable ASCII file, and stored in two files, one for the data, the other for hashing information. They are usually called *dbase*.pag and *dbase*.dir, respectively.

It is very important to remember that whenever you edit the master database, the changes do not take effect unless you update the dbm files as well. For easier maintenance, you should keep a small Makefile around that updates the the databases.

The dbm files are built using the dbm(1) command, except for the aliases database, which is given special treatment. For example, to build the mailertable database from the master file, issue the command:

```
# dbm -d mailertable load mailertable
```

This reads the file mailertable and produces both mailertable.pag and mailertable.dir.

10.8.2 The aliases file

Most important among the sendmail support files is the aliases file. It is used to map addresses to a list of other addresses, which allows for mailing lists, mail forwarding, etc. Unlike smail, sendmail not only allows to introduce aliases for local users, but also for addresses of users on other sites.

A sample aliases file might look like this:

```
# linus.lxnet.org alias file
# set of standard addresses
MAILER-DAEMON: postmaster
postmaster: root
# disallow mail to ftpmailer@decwrl.dec.com
ftpmailer@decwrl.dec.com: /dev/null
# local echo of Linux-Activists
linux-activists: sue, mark, al, anne,
        jeff@ziggy.uucp, /v/pub/linux/maillog
owner-linux-activists: al
```

When setting up a mail site, you should make sure the addresses postmaster and MAILER-DAEMON are defined. The easiest way to do so is using an aliases entry as above.

The third alias is a rather drastic way to prevent users from using the ftpmailer service, which allows you to conduct FTP sessions by sending the list of commands to decwrl, which will connect to the target server for you. In small UUCP networks, having people use such services can be a real nuisance. However, I'm not sure if the alias really prevents this; it's just an example.

For more effective use, the `aliases` file must be converted to a `dbm` database for `sendmail` to be able to use it. However, this is not done using the standard `dbm(1)` command, but by invoking `sendmail` with the `-bi` option:

```
# sendmail -bi
```

This is due to the fact that a slightly different `dbm` format is used for the `aliases` database. While the `dbm` files are rebuilt, `sendmail` will notice this and wait for the update to complete.

10.8.3 The `pathtable` File

`sendmail`'s pathalias database is usually called `pathtable`. The plain-text file has the usual format produced by the `pathalias(1)` tool. A sample file is given in section 8.5.

10.8.4 Miscellaneous `dbm` Files

There are a couple of support files we will not discuss in this book. The one you probably would want to use is the `mailertable` database, which maps destination sites or domains onto a gateway and a transport (mailer). For example, to be able to send mail to hosts on BITNET, you either have to rely on a smart host, or explicitly name a BITNET gateway to send all messages for BITNET addresses to.

There is also a pair of databases to map UUCP site names onto fully qualified domain names and vice versa. These are `uucpxtable` and `domaintable`. Closely related to this is the list of well-known UUCP relays, which may be given to `sendmail` in yet another `dbm` database (The database name is given in the IDA configuration file in the `UUCPRELAYS` macro). Paths through any of these relays will have the path leading up to the relay removed, and have it replaced by the "optimal" path.

For use with other networks, like XNS or DECnet, there are also a couple of databases that may be used. If you do need these options, please refer to the original documentation.

10.9 sendmail Mailers

10.9.1 Local Addresses

When `sendmail` receives a message for a local address, and the user name is not found in the `aliases` file, it goes on to check if there is a local user matching this name.

First, it checks if the recipient name is indeed a valid account name on the local system. If it is, the user's home directory is extracted from the `/etc/passwd` password file and checked for a `.forward` file. If one exists, the message is delivered to the list of comma-separated addresses specified in this file.

Messages to local users which have not been aliased to some other address are delivered by invoking the `lmail` program. Note that an `lmail` program is part of the `sendmail` package, which does little more than appending the message to the recipient's mailbox file.

`sendmail` allows you to deliver mail to programs and files as well. Any address beginning with a slash ('`/`') is assumed to be a file name, and the message is appended to this file. The address syntax for delivering to a command is

`|`*command*   *or*   `|"`*command arguments*`"`

The second form is needed whenever the command takes arguments. Note that opposed to `smail`, `sendmail` will not pass on any environment variables to the program invoked.

10.9.2 SMTP Delivery

There are also a number of SMTP-based mail services. The generic SMTP mailer is `TCP`, which is automatically chosen when the target hostname has been resolved by the name server.

Some variations of this exist, which may be used to appease mailers that need special address formatting of, for example, relative UUCP addresses. For these, some additional rewriting of the recipient's envelope address is done, which is described in the `sendmail.cf` file. To use any of these mailers, you should specifiy them explicitly in the `mailertable` file.

10.9.3 UUCP-based mailers

As configured, `sendmail` offers support for three varieties of mailer delivery over UUCP. The first two, `UUCP` and `UUCP-A`, both deliver messages to the `rmail` command on the remote site. The only difference is that `UUCP-A` is a bit smarter in that it tries to format addresses as close to RFC 822 as possible.

The `UUCP-B` mailer may be used to deliver messages using batched SMTP. It produces a BSMTP file (as explained in section 8.2) and sends it to the `bsmtp` program on the remote host. In this respect it is different from `smail`, who sends the files to the `rsmtp` program. If you expect to receive mail in BSMTP batches, make sure to link `sendmail` to `bsmtp`; when invoked as `bsmtp`, it will eat the batch and process it accordingly.

Note that for `sendmail` to receive BSMTP batches created by `smail`, you have to have a `rsmtp` program. However, a shell script like the following might do:

```
#!/bin/bash
exec /usr/local/bin/bsmtp
```

The default to use for delivering mail over a UUCP link is to use the dumb `UUCP` mailer. If you want to override this for specific mail links, you have to do so in the `mailertable` file. For example, if you want to use the `UUCP-A` mailer for mail forwarded to `swim.two.birds` because you know their mailer speaks RFC 822, while for your link to `moria.orcnet.org`, you prefer BSMTP, your `mailertable` would contain lines like these:

```
swim.two.birds        UUCP-A!swim
moria.orcnet.org      UUCP-B!moria
```

Chapter 11

Netnews

11.1 Usenet History

The idea of network news was born in 1979 when two graduate students, Tom Truscott and Jim Ellis, thought of using UUCP to connect machines for the purpose of information exchange among Un⋆x users. They set up a small network of three machines in North Carolina.

Initially, traffic was handled by a number of shell scripts (later rewritten in C), but they were never released to the public. They were quickly replaced by "A" news, the first public release of news software.

It was however not designed to handle more than a few articles per group and day. When volume continued to grow, it was rewritten by Mark Horton and Matt Glickman, who called it the "B" release (a.k.a. Bnews). The first public release of Bnews was version 2.1 in 1982. It was expanded continuously, with several new features being added. Its current version is Bnews 2.11.

Another rewrite was done and released in 1987 by Geoff Collyer and Henry Spencer; this is release "C", or Cnews. In the meanwhile there have been a number of patches to Cnews, the most prominent being the Cnews Performance Release. On sites that carry a large number of groups, the overhead involved in frequently invoking the `relaynews` command is significant. (`relaynews` is responsible for dispatching incoming articles to the hosts that request news from the specified groups.) The performance release adds an option to `relaynews` that allows to run it in *daemon mode*, in which the program puts itself in the background once and for all times.[1]

[1]Until the system is brought down, that is.

The Performance Release is the Cnews version currently included in SLS and `newspak`.

All news releases up to "C" are primarily targeted for UUCP networks, although they may be used in other environments as well. Though, efficient news transfer over networks like TCP/IP, DECNet, or related requires a new scheme. This was the reason why, in 1986, the "Network News Transfer Protocol", NNTP, was introduced. It is based on network connections, and specifies a number of commands to interactively transfer and retrieve articles.

11.2 How Does Usenet Handle News?

Today, Usenet has grown to enormous proportions. Sites that carry the whole of netnews usually transfer something like a trifle fourty megabytes a day.[2] Of course this requires much more than pushing around files. So let's take a look at the way most Un⋆x systems handle Usenet news.

The basic unit of Usenet news is the article. This is a message a user writes and "posts" to the net. In order to enable news sytems to deal with it, it is prepended with administrative information, the so-called article header. It is very similar to the mail header format laid down in RFC 822, in that it consists of several lines of text, each beginning with a field name terminated by a colon, which is followed by the field's value.[3]

Articles are submitted to one or more *newsgroups*. One may view a newsgroup as a forum for articles relating to a common topic. All newsgroups are organized in a hierarchy, with each group's name indicating its place in the hierarchy. This often makes it easy to see what a group is all about. For example, anybody can see from the name `comp.os.linux` that it refers to a computer operating system named Linux.

News are distributed through the net by various transports. The historical medium used to be UUCP, but today the main traffic is carried by Internet sites. The algorithm used is *flooding*: Each site maintains a number of links (*news feeds*) to other sites. Any article received by the local news system is forwarded to them, unless it has already been seen at the local site, in which case it is discarded. The same applies to articles injected into the news stream by a local user.

To distinguish articles and recognize duplicates, Usenet articles have to carry a message id (specified in the `Message-Id:` header field), which combines the posting site's name and

[2] Wait a moment: 40 Megs at 2400 bps, that's 40 million by 300, that is... mutter, mutter,... Hey! That's 37 hours!

[3] The format of Usenet news messages is specified in RFC 1036, "Standard for interchange of USENET messages".

a serial number into "<*serial*@*site*>". For each article processed, the news system logs this id into a *history* file against which all newly arrived articles are checked.[4]

This algorithm is modified in that the flow between any two sites may be limited by two criteria: for one, an article is assigned a distribution (in the `Distribution:` header field) which may be used to confine it to a certain group of sites. On the other hand, the newsgroups transmitted may be limited by either the sending or receiving system, depending on the transport used. The set of newsgroups and distributions allowed for transmission to a site are usually kept in the `sys` file.

The sheer number of articles usually requires that improvements be made to the above scheme. On UUCP networks, the natural thing to do is to collect articles over a period of time, and combine them into a single file, which is compressed and sent to the remote site. This is called *batching*.[5]

An alternative technique is the *ihave/sendme* protocol that prevents duplicate articles from being transferred in the first place, thus saving net bandwidth. Instead of putting all articles in batch files and sending them along, only the message ids of all articles that would otherwise be batched are combined into a giant "ihave" message and sent to the remote site. It reads this message, compares it to its history file, and returns the list of articles it wants in a "sendme" message. Only these articles are then sent. Of course, ihave/sendme only makes sense if it involves two big sites that receive news from several independent feeds each, and who poll each other often enough for an efficient flow of news.

Sites that are on the Internet generally rely on TCP/IP-based software that uses the Network News Transfer Protocol, NNTP.[6] It allows to transfer news between feeds and provides Usenet access to single users on remote hosts.

NNTP knows three different ways to transfer news. One is ihave/sendme, also referred to as *pushing* news. The second technique is called *pulling* news, in which the client requests a list of articles in a given newsgroup or hierarchy that have arrived at the server's site after a specified date, and chooses those it cannot find in its history file. The third mode is for interactive newsreading, and allows to retrieve articles from specified newgroups, as well as posting article with incomplete header information.

At each site, news are kept in a directory hierarchy below `/usr/spool/news`, each article in a separate file, and each newsgroup in a separate directory. The directory name is made up of the newsgroup name, with the components being the path components. Thus,

[4] This used to be one of the worst problems with B news, because it searched the history file linearly. The time required for this is proportional to the square of the number of articles. C news generally uses `dbm` databases which support hash tables.

[5] The golden rule of netnews, according to Geoff Collyer: "Thou shalt batch thine articles."

[6] Described in RFC 977.

`comp.os.linux` articles are kept in `/usr/spool/news/comp/os/linux`. The articles in a newsgroup are assigned numbers in the order they arrive. This number serves as the file's name. The range of numbers of articles currently online is kept in a file called `active`.

Since disk space is a finite resource only,[7] one has to start throwing away articles after some time. This is called *expiring*. Usually, articles from certain groups and hierarchies are expired at a fixed number of days after they arrive. This may be overridden by the poster by specifying a date of expiry in the `Expires:` field of the article header.

11.3 A Description of Cnews

One of the most popular software packages for Netnews is Cnews. It was designed for sites that carry news over UUCP links. Its main component is `relaynews`, which takes incoming articles, stores it on the local host, and forwards it to the appropriate sites.

Cnews stores its configuration files in `/usr/local/lib/news`, and most of its binaries in the `/usr/local/lib/news/bin` directory. Articles are kept below `/usr/spool/news`. You should make sure virtually all files in these directories are owned by user `news`, group `news`. Most problems arise from files inaccessible to Cnews. Make it a rule for you to become user `news` using `su` before you touch anything in there.[8]

In the following, we describe all Cnews configuration files in detail, and show you what you need to do to keep your site running.

11.3.1 Delivering News

Articles may be fed to Cnews in several ways. When a local user posts an article, the newsreader usually hands it to the `inews` command, which completes the header information. News from remote sites, be it a single article or a whole batch, is given to the `rnews` command, which stores it in the `/usr/spool/newsin.coming` direcotry, from where it will be picked up at a later time by `newsrun`. With any of these two techniques, however, the article will eventually be handed to the `relaynews` command.

For each article, `relaynews` first checks if the article has already been seen at the local site by looking up the message id in the `history` file. Duplicate articles will be dropped. Next, `relaynews` looks at the `Newsgroups:` header line to find out if the local site requests articles from any of these groups. If it does, `relaynews` tries to store the article in the

[7] Some people claim that Usenet is a conspiracy by modem and hard disk vendors. This is of course ridiculous.

[8] The only exceptions is `setnewsids`, which is used to set the real user id of some news programs. It must be owned by `root` and must have the setuid bit set.

corresponding directory in the news spool area, and the article's message id will be logged to the `history` file. Otherwise, it drops the article.

If `relaynews` fails to store an incoming article because a group it has been posted might not exist, the article will be moved to the `junk` group.[9] `relaynews` will also check for stale or misdated articles and reject them. Incoming batches that fail for any other reason are moved to `/usr/spool/news/in.coming/bad`, and an error message is logged.

After this, the article will be relayed to all other sites that request news from these groups, using the transport specified for each particular site. To make sure it isn't sent to a site that already has seen it, each destination site is checked against the article's `Path:` header field, which contains the list of sites the article has traversed so far.[10] Only if the destination site's name does not appear in this list will the article be sent to it.

Cnews is commonly used to relay news between UUCP sites, altough it is also possible to use it in a NNTP environment. To deliver news to a remote site — either single articles or whole batches — `uux` is used to execute the `rnews` command on the remote site, and feed the article or batch to it on standard input.

When batching is enabled for a given site, Cnews does not send any incoming article immediately, but appends its path name to a file, usually `out.going/`*site*`/togo`. Periodically, a *batching* program is executed from a crontab entry,[11] which puts the articles in one or more files, optionally compresses them, and sends them to `rnews` at the remote site. The configuration options that apply to batching are described below in section 11.3.5

Figure 11.1 shows the news flow through `relaynews`. Articles may be relayed to the local site (denoted by `ME`), to some site named `ponderosa` via email, and a site named `moria`, for which batching is enabled.

11.3.2 Installation

To install Cnews, untar the files into their proper places if you haven't done so yet, and edit the configuration files listed below. They are all located in `/usr/local/lib/news`. Their formats will be described in the following sections.

[9] Note that there may be a difference between the groups that exist at your site, and those that your site is willing to receive. For example, the subscription list may specify `comp.all`, which means all newsgroups below the `comp` hierarchy, but at your site, you have only created spool directories for a number of `comp` groups.

[10] It is in fact a bang path style return address of the poster. It is not wise to use this, though, since not all systems that exchange news do exchange mail. Better use the address given in `From:`.

[11] Note that this should be the crontab of `news`, in order not to mangle file permissions.

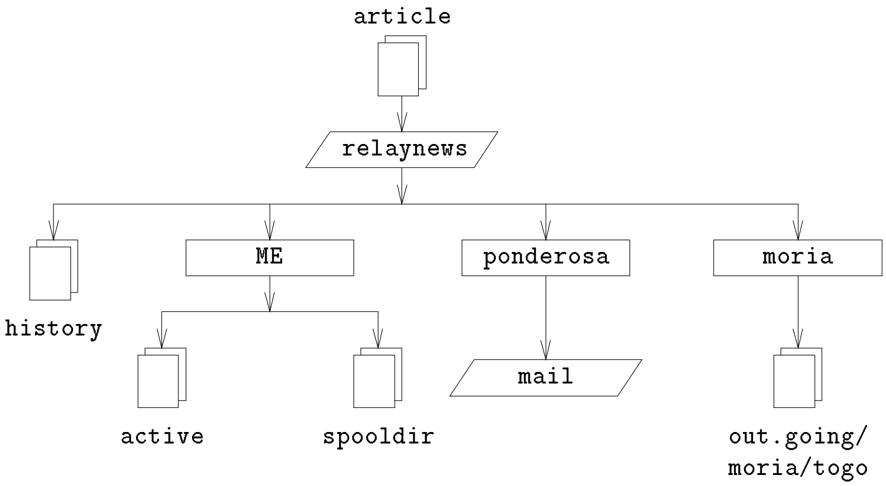

Figure 11.1: News flow through `relaynews`.

`sys` You probably have to modify the `ME` line that describes your system, although using `all/all` is always a safe bet. You also have to add a line for each site you feed news to.

If you are a leaf site, only a line that returns any locally generated articles to your feed is needed. Assume your feed is `moria`, then your `sys` file should look like this:

```
ME:all/all::
moria/moria.orcnet.org:all/all:f:
```

`organization`
Put your organization's name here. For example, "`Virtual Brewery, Inc.`".

`mailname` Your site's mail name, e.g. `linus.lxnet.org`.

`whoami` Your site's name for news purposes. Quite often, the UUCP site name is used, for example `linus`.

`explist` You should probably edit this file to reflect your preferred expiry times for some special newsgroups. Space considerations may play an important role in it.

To create an initial hierarchy of newsgroups, proceed as follows: Create the following newsgroups using the `addgroup` command: `to.`*mysite*, `to.`*feedsite*, `junk`, and `control`.

You should create the `to.*` groups regardless of whether you plan to use ihave/sendme or not.

Then ask the site that feeds you to send you a `checkgroups` message. They can do this by composing an article that contains their `newsgroups` file, and has a header line of "`Control: checkgroups`". They should then post it to `to.`*mysite* (and make sure no other site gets it). When this article arrives at your site, Cnews will put the article's contents in its `newsgroups` file, and compose a shell script that contains the necessary `addgroup` commands to create all groups, which it sends to the newsmaster. You should then save this to a file, remove the mail header, and feed it to a shell. Voilà!

Cnews requires a user to send error messages and status reports to. By default, this is `usenet`. If you use the default, you have to set up an account for it, probably forwarding all of its mail to one or more responsible persons. (Chapters 9 and 10 explain how to do so for `smail` and `sendmail`). You may also override this behavior by setting the environment variable `NEWSMASTER` to the appropriate name. You have to do so in `news`' crontab file, as well as every time you invoke an administrative tool manually.

While you're hacking `/etc/passwd`, make sure that every user has her real name in the `pw_gecos` field of the password file (this is the fourth field). It is a question of Usenet netiquette that the sender's real name appears in the `From:` field of the article. Of course, you will want to do so anyway when you use mail.

11.3.3 The sys file

The `sys` file, located in `/usr/local/lib/news`, controls which hierarchies you receive and forward to other sites. Although there are a maintenance tools named `addfeed` and `delfeed`, I think it's better to maintain these files by hand.

The file contains entries for each site you forward news to, as well as a description of the groups you will accept. An entry looks as follows:

site`[/`*exclusions*`]:`*grouplist*`[/`*distlist*`][:`*flags*`[:`*cmds*`]]`

Entries may be continued across newlines using a backslash ('\'). A hash sign ('`#`') denotes a comment.

site This is the name of the site the entry applies to. One usually chooses the site's UUCP name for this. There has to be an entry for your site in the `sys` file, too, else you will not receive any articles yourself. The special site name `ME` denotes your site.

Since Cnews checks *site* against the site names in the `Path:` header field, you have to make sure they really match. Some sites use their fully qualified domain name in this field, or an alias like `news.`*site.domain*. To prevent any articles from being returned to these sites, you have to add these to the exlusion list, separated by commas.

grouplist This is a comma-separated subscription list of groups and hierarchies for that particular site. A hierarchy may be specified by giving either the hierarchy's prefix (such as `comp.os` for all groups whose name starts with this prefix), or by giving the prefix with `all` appended (e.g. `comp.os.all`). A hierarchy or group is excluded from forwarding by preceding it with an exclamation mark ('`!`'). If a newsgroup is checked against the list, the longest match applies. For example, if *grouplist* contains

```
!comp,comp.os.linux,comp.folklore.computers
```

no groups from the `comp` hierarchy except `comp.folklore.computers` and all groups below `comp.os.linux` will be fed to that site.

If the site requests to be forwarded all news you receive yourself, enter `all.all` as *grouplist*.

distlist is offset from the *grouplist* by a slash, and contains a list of distributions to be forwarded. Again, you may exclude certain distributions by preceding them with an exclamation mark. All distributions are denoted by `all`. Omitting *distlist* implies a list of `all`.

For example, you may use a distribution list of `all,!local` to prevent news for local use only from being sent to remote sites.

There are usually at least two distributions: `world`, which is often the default distribution used when none is specified by the user, and `local`. There may be other distributions that apply to a certain region, state, country, etc. Finally, there are two distributions used by Cnews only; these are `sendme` and `ihave`, and are used for the sendme/ihave protocol.

The use of distributions is a subject of debate. For one, some newsreaders (most notorius is a Mac newsreader) create bogus distributions by simply using the top level hierarchy, for example `comp` when using to `comp.os.linux`. Distributions that apply to regions are often questionable, too, because news may travel outside of your region when sent across the Internet.[12] Distributions applying to an organization, however, are very well meaningful,

[12]It is not impossible for an article posted in, say Hamburg, to go to Frankfurt via `reston.ans.net`.

for example to prevent confidential information from leaving the company network. This purpose, however, is generally served better by creating a separate newsgroup or hierarchy.

flags
This describes certain parameters for the feed. It may be empty, or a combination of the following:

`F`
This flag is used to enable batching of articles. The *cmds* field contains a pathname for placing the article lists.

`f`
This is almost identical to the `F` flag, but allows Cnews to calculate the size of outgoing batches more precisely.

`L`
This tells Cnews only to transmit articles posted at your site. This flag may be followed by a decimal number *n*, which makes Cnews only transfer articles posted within *n* hops from your site. Cnews determines the number of hops from the length of the `Path:` field.

`I`
This flag makes Cnews produce an article list suitable for use by ihave/sendme.

`n`
This creates batch files for use with the active NNTP transfer client, `nntpxmit` (see section 11.4.5). The batch files contain the articles filename along with its message id.

`u`
This tells Cnews only to batch articles from unmoderated groups.

`m`
This tells Cnews only to batch articles from moderated groups.

You may use at most one of `F`, `f`, `I`, or `n`.

cmds
This field contains a command to be executed for each article, unless batching is enabled. This should only be used for very small feeds; otherwise the load on both systems will be too high.

If batching is enabled using either of the `F` or `f` flags, Cnews expects to find a directory name in this field rather than a command. The same applies when using NNTP or ihave/sendme. (For batching and delivery, it uses the batcher and transport specified in the `batchparms` file, see section 11.3.5). This directory is used to create various batching-related files in it, among other the list of all articles queued for this site.[13]

The directory must be given as relative to `/usr/spool/newsout.going`. If the field is empty, the directory name defaults to the site name.

When setting up Cnews, you will most probably have to write your own `sys` file. To help you with it, we give a sample file for `linus.lxnet.org` below, from which you might copy what you need.

```
# We take whatever they give us.
ME:all/all::

# We send everything we receive to moria, except for local and
# brewery-related articles.
moria/moria.orcnet.org:all,!to,to.moria/all,!local,!brewery:f:

# We mail comp.security to jack@ponderosa.uucp
ponderosa:comp.security/all::rmail jack@ponderosa.uucp

# swim gets a minor feed
swim/swim.two.birds:comp.os.linux,rec.humor.oracle/all,!local:f:
```

11.3.4 The `active` file

The `active` file is also located in `/usr/local/lib/news` and lists all groups known at your site, and the articles currently online. You will rarely have to touch it, but we explain it nevertheless for sake of completeness. Entries take the following form:

newsgroup: *high low perm*

newsgroup is, of course, the group's name. *low* and *high* are the lowest and highest numbers of articles currently available. These are 6-digit decimal numbers. If no articles are currently available, they take the values `1` and `0`, respectively.

perm is a parameter detailing the access users are granted to the group. It takes one of the following values:

`y` Users are allowed to post to this group.

`n` Users are not allowed to post to this group. However, the group may still be read.

[13]This list is kept in the file `togo`.

`x` — This group has been disabled locally. This happens sometimes when news admininistrators (or their superiors) take offense in articles posted to certain groups.

Articles received for this group are not stored locally, although they are forwarded to the sites that request them.

`m` — This denotes a moderated group. When a user tries to post to this group, an intelligent newsreader will notify her of this, and send the article to the moderator instead. The moderator's address is taken from the `moderators` file in `/usr/local/lib/news`.

`=`*real-group* — This marks *newsgroup* as being a local alias for another group, namely *real-group*. All articles posted to *newsgroup* will be redirected to it.

In Cnews, you will generally not have to access this file directly. Groups may be added or deleted locally using `addgroup` and `delgroup` (see below in section 11.3.10). When groups are added or deleted for the whole of Usenet, this is usually done by sending a `newgroup` or `rmgroup` control message, respectively. Never send such a message yourself! For instructions on how to create a newsgroup, read the monthly postings in `news.announce.newusers`.

A file closely related to `active` is `active.times`. Whenever a group is created, Cnews logs a message to this file, containing the name of the group created, the date of creation, whether it was done by a `newgroup` control message or locally, and who did it. This is for the convenience of newsreaders who may notify the user of any recently created groups. It is also used by the `NEWGROUPS` command of NNTP.

11.3.5 Article Batching

Article batching is performed by `sendbatches`, located in `/usr/local/lib/news/bin/batch`. It should be executed once per hour or even more frequently, depending on the volume of traffic.

Its operation is controlled by the `batchparms` file in `/usr/local/lib/news`. This file describes the maximum batch size allowed for each site, the batching and optional compression program to be used, and the transport for delivering it to the remote site. Unless you have special requirements for one of your feeds, you will not have to touch this file. Entries take the following format:

site size max batcher muncher transport

The meaning of these fields is as follows:

site — This is the name of the site the entry applies to. A default entry may be given, using the special site name `/default/`.

`sendbatches` extracts the site name from the entry, and checks the `out.going/`*site*`/togo` file for any batched articles.

size — This is the maximum size of article batches created (before compression). An exception from this are articles whose size exceeds *size*; these are put in a single batch nevertheless.

max — is the maximum number of batches created and scheduled for transfer before batching stalls for this particular site. This is useful in case the remote site should be down for a longer time, because it prevents Cnews from cluttering your UUCP spool directories with zillions of newsbatches. Cnews determines the number of queued batches using the `queulen` script in `/usr/local/lib/news/bin`. Vince Skahan's newspak release should contain a script for BNU-compatible UUCPs. If you use a different flavor of spool directories, for example, Taylor UUCP, you might have to get a different script from the Cnews source, or write your own.[14,15]

batcher — This field contains the command used for producing a batch from the list of articles in the `togo` file. For regular feeds, this is usually `batcher`. For other purposes, alternative batchers may be provided. For example, the ihave/sendme protocol requires to turn the article list into ihave or sendme control messages, which are posted to the newsgroup `to.`*site*. This is performed by `batchih` and `batchsm`.

muncher — The *muncher* field specifies the command used for compression. Usually, this is `compcun`, a script that produces a compressed batch.[16] Alternatively, you might provide a muncher that uses `gzip`, say `gzipcun` (to be clear: you have to write it yourself). You have to make sure that `uncompress` on the remote site is patched to recognize files compressed with `gzip`.

If the remote site is not have an `uncompress` command, you may specify `nocomp` which does not do any compression.

---

[14]The current source of Cnews does not contain a `queuelen` script for a Taylor-style spool hierarchy.

[15]If you don't care about the number of spool files (because you're the only person using your computer, and you don't write articles by the megabyte), you may replace the script's content by a simple `exit 0` statement.

[16]As shipped with Cnews, `compcun` uses `compress` with the 12 bit option, since this is the least common denominator for most Un$\star$x sites. You may produce a copy of it, say `compcun16`, where you use 16 bit compression. The improvement is not too impressive, though.

transport The last field describes the transport to be used. A number of standard commands for different transports are available whose names begin with `via`. They are given the site name as an argument on the command line. This is derived from the *site* field by stripping of anything after and including the first dot or slash.

There are two commands for use of `uux` to execute `rnews` on the remote system; `viauux` and `viauuxz`. The latter sets the `-z` flag for (older versions of) `uux` to keep it from returning success messages for each article delivered. Another two commands are for mail transport; both send the article to the user `enews` on the remote system, they are called `viaemail` and `viapmail`, with the latter protecting the article against stupid mailers by prepending 'N' to each line. For a complete list of these transports, refer to the `newsbatch(8)` manual page.

All commands from the last three fields must be located in `/usr/local/lib/news/bin/batch`. Most of them are scripts, so that you may easily tailor new tools for your personal needs. They are invoked as a pipe, with the list of articles fed to the batcher on standard input, which produces the batch on standard output. This piped into the muncher, and so on.

When `sendbatches` is invoked without an argument, it handles all batch queues. The interpretation of "all" depends on the presence of a `/default/` entry in `batchparms`. If one is found, all directories in `/usr/spool/news/out.going` are checked, otherwise, it cycles through all entries in `batchparms`.

A sample file is given below.

```
# batchparms file for the brewery
# site        | size   |max    |batcher  |muncher    |transport
#-------------+--------+-------+---------+-----------+-----------
/default/       100000  22      batcher   compcun     viauux
swim             10000  10      batcher   nocomp      viauux
```

11.3.6 Expiring News

In Bnews, expiring used to be performed by a program called `expire`, which took a list of newsgroups as argument, together with a time specification after which articles had to be expired. To have different hierarchies expired at different times, you had to write a script that invoked `expire` for each of them separately. Cnews offers a more convenient solution to this: in a file called `explist`, you may specify newsgroups and expiration intervals. A command called `doexpire` is usually run once a day from `cron`, which processes all groups according to this list.

Additionally, you may want to retain articles from certain groups even after they have been expired; for example, you might want to keep programs posted to `comp.sources.unix`. This is called *archiving*. `explist` permits to mark groups for archiving.

An entry in `explist` looks like this:

grouplist perm times archive

grouplist — *grouplist* is a comma-separated list of newsgroups to which the entry applies. Hierarchies may be specified by giving the group name prefix, optionally appended with `all`. For example, for an entry applying to all groups below `comp.os`, you might either enter `comp.os` or `comp.os.all` in *grouplist*.

When expiring news from a group, the name is checked against all entries in `explist` in the order given. The first matching entry applies. For example, to throw away the majority of `comp` after four days, except for `comp.os.linux.announce` which you want to keep for a week, you simply have an entry for the latter, which specifies a seven-day expiry period, followed by that for `comp`, which specifies four days.

perm — The *perm* field details if the entry applies to moderated, unmoderated, or any groups. It may take the values `m`, `u`, or `x`, which denote moderated, unmoderated, or any type.

perm — The third field, *times*, usually only contains a single number, being the number of days after which articles will be expired if not assigned an artificial expiry date due to an `Expires:` field in the article header. Note that this is the number of days counting from its *arrival* at your site, not the date of posting.

The *times* field may, however, be more complex than that. It may be a combination of up to three numbers, separated from each other by a dash. The first denotes the number of days that have to pass before the article is considered a candidate for expiry. It is rarely useful to use a value other than zero. The second field is the above-mentioned default number of days after which it will be expired. The third is the number of days after which an article will be expired unconditionally, regardless of whether it has an `Expires:` field or not. If only the middle number is given, the other two take default values. They may be specified using the special entry `/bounds/`, which is described below.

perm The fourth field, *archive*, denotes whether the newsgroup is to be archived, and where. If no archiving is intended, a dash should be used. Otherwise, either a full path name (pointing to a directory) is used, or an at sign (‘`@`’). The at sign denotes the default archive directory which must then be given to `doexpire` by using the `-a` flag on the command line. An archive directory should be owned by `news`. When `doexpire` archives an article from, say `comp.sources.unix`, it stores it in the directory `comp/sources/unix` below the archive directory, creating it if not existent. The archive directory itself, however, will not be created.

There are two special entries in your `explist` file that `doexpire` relies on to exist. Instead of a list of newsgroups, they have the keywords `/bounds/` and `/expired/`. The `/bounds/` entry contains the default values for the three values of the *times* field described above.

The `/expired/` field determines how long Cnews will hold on to lines in the `history` file. This is needed because Cnews will not remove a line from the history file once the corresponding article(s) have been expired, but hold on to it in case a duplicate should arrive after this date. If you are fed by only one site, you can keep this value small, otherwise a couple of weeks is advisable on UUCP networks, depending on the delays you experience with articles from these sites.

A sample `explist` file is reproduced below:

```
# keep history lines for two weeks. Nobody gets more than three months
/expired/                       x       14      -
/bounds/                        x       0-1-90  -

# groups we want to keep longer than the rest
comp.os.linux.announce          m       10      -
comp.os.linux                   x       5       -
alt.religion.kibology           u       10      -
rec.humor.oracle                m       10      -
soc.feminism                    m       10      -

# Archive *.sources groups
comp.sources,alt.sources        x       5       @

# defaults for tech groups
comp,sci                        x       7       -

# enough for a long weekend
misc,talk                       x       4       -
```

```
# throw away junk quickly
junk                            x       1       -

# control messages are of scant interest, too
control                         x       1       -

# catch-all entry for the rest of it
all                             x       2       -
```

With expiring in Cnews, there are a number of potential troubles looming. One is that your newsreader might rely on the third field of the active file, which contains the number of the lowest article on-line. When expiring articles, Cnews does not update this field. If you need (or want) to have this field represent the real situation, you need to run a script called `upact` regularly.

Second, Cnews does not expire by scanning the newsgroup's directory, but simply checks the `history` file if the article is due for expiry.[17] If your history file somehow gets out of sync, articles may be around on your disk infinitely, because Cnews has literally forgotten them.[18] You can repair this using the `addmissing` script in `/usr/local/lib/news/bin/maint`, which will add missing articles to the `history` file, or `mkhistory`, which re-builds the file from scratch. Don't forget to become `news` before invoking it, else you will wind up with a `history` file unreadable by Cnews.

11.3.7 Miscellaneous Files

There are a number of files that control Cnews' behavior, but are not essential to its functioning. All of them reside in `/usr/local/lib/news`. We will describe them shortly.

`organization`
: This file contains a single line that identifies your organization. If the machine is owned by Foobar, Inc., put this name in the file. If it is privately owned, enter "private site", or anything else you like. Most people will not call your site properly configured if you haven't customized this file.

`newsgroups`
: This is a companion file of `active` which contains a list of newsgroups names, along with a one-line description of its charter. This file is automatically up-

[17] The article's date of arrival is kept in the middle field of the history line, given in seconds since January 1, 1970.

[18] I don't know *why* this happens, but it does from time to time.

dated when Cnews receives a `checknews` control message (see section 11.3.8).

`localgroups` If you have a number of local groups that you don't want Cnews to complain about every time you receive a `checknews` message, put their names and descriptions in this file, just like they shall appear in `newsgroups`.

`mailpaths` This file contains the moderator's address for each moderated group. Each line contains the group name, followed by the moderator's email address (offset by a tab).

Two special entries are provided as default. These are `backbone` and `internet`. Both provide — in bang-path notation — the path to the nearest backbone site, and the site that understands RFC 822-style addresses (`user@host`). The default entries are

```
internet        %s
backbone        %s
```

You will not have to change the `internet` entry if you have `smail` or `sendmail` installed, because they understand RFC 822-addressing.

The `backbone` entry is used whenever a user posts to a moderated group whose moderator is not listed explicitly. If the newsgroup's name is `alt.sewer`, and the `backbone` entry contains *path*`!%s`, Cnews will mail the article to *path*`!alt-sewer`, hoping that the backbone machine is able to forward the article.

`whoami` and `mailname`

These files contain the site's name as used in locally generated articles (`whoami`) and mail sent by Cnews (`mailname`), respectively. Cnews may send mail to the news gurus, indicating errors, or in reply to a `sendsys` control message.

If these files do not exist, Cnews tries to obtain the hostname from various sources, with `hostname(1)` being the last resort.

These files are provided in case you want to use a host alias instead of your (canonical) host name. This is useful to hide the physical location of your news system, so that you may move it easily from one machine to another.

`server` You need this file if you are running a cluster of machines, with articles kept on a central node, whose spool area is mounted by the other hosts via NFS. It contains the name of the machine that acts as the "news server", in that every article may only be posted by this machine. Cnews takes care of this in the following way: When posting an article through `inews`, it checks if `server` exists, and if the name contained in it is its own hostname. If so, it proceeds by giving the article to `relaynews`; otherwise, it passes it on to `inews` on the central host by executing `rsh`.

Note that this requires that you have an equivalent account on the server machine that lets you in without asking for a password. You therefore have to have an `.rhosts` file set up properly in `news`' home directory, or an entry in `/etc/hosts.equiv` on the server. Please refer to section 11.3.9 for details.

`distributions`
: This file is not really a Cnews file, but it is used by some newsreaders, and `nntpd`. It contains the list of distributions recognized by your site, and a desciption of its (intended) effect. For example, my site has the following file:

```
world          evrywhere in the world
local          Only local to this site
nl             Netherlands only
mugnet         MUGNET only
fr             France only
de             Germany only
orcnet         All *.orcnet.org sites
brewery        Virtual Brewery only
```

`log`
: This file contains a log of all Cnews activities. It is culled regularly by running `newsdaily`; copies of the old logfiles are kept in `log.o`, `log.oo`, etc.

`errlog`
: This is a log of all error messages created by Cnews. These do not include articles junked due to wrong group, etc. This file is mailed to the newsmaster (`usenet` by default) automatically by `newsdaily` if it is found to be non-empty.

: `errlog` is cleared by `newsdaily`, old copies are kept in `errlog.o` and companions.

`batchlog`
: This logs all runs of `sendbatches`. It is usually of scant interest only. It is also maintained by `newsdaily`.

`watchtime`
: This is an empty file created each time `newswatch` is run.

11.3.8 Control Messages

The Usenet news protocol knows a special category of articles which evoke certain responses or actions by the news system. These are called *control* messages. They are recognized by the presence of a `Control:` field in the article header. There are several types of them, all of which are dealt with by shell scripts located in `/usr/local/lib/news/ctl`. Most of these will perform their action automatically at the time the article is processed by Cnews, without notifying the newsmaster. By default, only `checkgroups` messages will be handed to the newsmaster,[19] but you may change this by editing the scripts.

The one most widely known is the `cancel` message, with which a user may cancel an article sent by her earlier. This effectively removes the article from the spool directories, if it exists. The `cancel` message is forwarded to all sites that receive news from the groups affected, regardless of whether the article has been seen already or not. This is to take into account that the original article has been delayed over the cancellation message. Some news systems allow users to cancel other person's messages; this is of course a definite no-no.

Two messages dealing with creation or removal of newsgroups are the `newgroup` and `rmgroup` message. Newsgroups below the "usual" hierarchies may only be created after a discussion and voting has been held among Usenet readers. The rules applying to the `alt` hierarchy allow for something close to anarchy. For more information, see the regular postings in `news.announce.newusers`. Never send a `newgroup` ar `rmgroup` message yourself unless you definitely know that you are allowed to.

The `checkgroups` message mentioned above is sent by "central authority" (a non-entity on Usenet) to synchronize a site's `active` file with the realities of Usenet. When Cnews receives a `checkgroups` message, it rewrites the `newsgroups` file, adding the groups in `localgroups`. If there are mismatches between the `checkgroups` list and the `active` file, it sends a message to the netnews administrator (usually `usenet`) that contains a list of control commands to bring your site up to date.[20]

Finally, there are three messages that may be used to find out something about the network's topology. These are `sendsys`, `version`, and `senduuname`. They cause Cnews to return to the sender the `sys` file, a software version string, and the output of `uuname(1)`, respectively. Cnews is very laconic about `version` messages, it returns a simple, unadorned

[19]There's a lovely typo in RFC 1036 (p.12):

> 3. Control Messages
>
> [...] Implementors and administrators may choose to allow control messages to be carried out automatically, or to queue them for annual processing.

[20]Note that the message, depending on the organization that expedits it, need not contain all Usenet newsgroups. Make sure you don't remove any groups accidentally by believing the message blindly.

"C".

Again, you should *never* issue such a message, unless you have made sure that it cannot leave a your (regional) network. Replies to `sendsys` messages can quickly bring down a UUCP network.[21]

11.3.9 Cnews in an NFS Environment

A simple way to distribute news within a local network is to keep all news on a central host, and export the relevant directories via NFS, so that newsreaders may scan the articles directly.

The advantage of this method over NNTP is that the overhead involved in retrieving and threading articles is significantly lower. NNTP, on the other hand, wins in a heterogeneous network where equipment varies widely among hosts, or where users don't have equivalent accounts on the server machine.

When using NFS, of course, articles posted on a host have to be forwarded to the central machine, because accessing adminstrative files (like `active`) might otherwise expose the system to race-conditions. Also, you might want to protect your news spool area by exporting it read-only, which necessitates forwarding to the central machine, too.

Cnews handles this transparently. When posting an article, the newsreader software usually invokes `inews`. This is a script which runs a number of checks on the article, completes the header, and checks the file `server` in `/usr/local/lib/news` if the host it's running on is indeed the server. Otherwise, it invokes `inews` on the server host via `rsh`. Note that this setup requires that the client hosts have a standard Un⋆x environment (including a working `awk`, `sed`, and Bourne shell). The `inews` script also uses a number of binary commands from the Cnews distribution (e.g. `getdate`), so that you have to have different sets of binaries for different architectures.

For the `rsh` invocation to work properly, the user needs to have an equivalent account on the remote system, i.e. one to which she can login without being asked for a password. How this can be achieved is explained in section 4.5.

Make sure that the hostname given in `server` literally matches the output of the `hostname(1)` command on the server machine, else Cnews will be looping forever when trying to deliver the article.

[21] I wouldn't try this on the Internet, either.

11.3.10 Maintenance Tools and Tasks

Despite of the complexity of Cnews, a news administrator's life can be fairly easy, because Cnews provides you with a wide variety of maintenance tools. Unless stated otherwise, they are located in `/usr/local/lib/news/bin/maint`. Note that you must become user `news` before invoking these commands. Running them as super-user may render these files inaccessible to Cnews.

`adddirs` Creates directories for all groups in the `active` file which do not yet have a spool directory.

`addfeed` Add a feed to your `sys` file. It is invoked as follows:

> `addfeed [-L] [-f` *feedtype*`]` *site grouplist*

site is the name of the site to be added; `-L` signals if the `L` flag should be set on the feed — that is, only locally generated news is returned. The `-f` option may be used to specify a batching flag (`f` or `F`); it defaults to `f`. *grouplist* is the list of newsgroups you are willing to send to *site*, optionally followed by a list of distributions. If none are given, `addfeed` assumes all distributions are sent to that site.

A special *grouplist* of `=`*othersite* may be used to copy another feed's group list.

`addgroup` Adds a group to your site locally. The proper invocation is

> `addgroup` *groupname* `y|n|m|=`*realgroup*

The second argument has the same meaning as the flag in the `active` file, meaning that anyone may post to the group (`y`), no-one (`n`), that it is moderated (`m`), or that it is an alias for another group (`=`*realgroup*).

You might also want to use `addgroup` when the first articles in a newly created group arrive earlier than the `newgroup` control message that is intended to create it.

`delgroup` Allows you to delete a group locally. Invoke it as

> `delgroup` *groupname*

You still have to delete the articles that still reside in the newsgroup's spool directory. Alternatively, you might leave it to the natural course of events (a.k.a. `expire`) to make them go away.

`addmissing` Adds missing articles to the `history` file. Run this script when there are articles that seem to hang around forever.[22]

`newsdaily` The name already says it: runs this once a day. It is an important script that helps you keep log files small, retaining copies of each from the last three runs. It also tries to sense any anomalies, like stale batches in the incoming and outgoing directories, postings to unkown or moderated newsgroups, etc. Resulting error messages will be mailed to the newsmaster.

`newswatch` This is a script that may be run regularly to look for anomalies in the news system. It is intended to detect problems that will have immediate effect on the operability of your news system, like stale lock files that don't get removed, unattended input batches, and disk space shortage. It is invoked as

`newswatch` *minfree maxbatch*

`newsboot` This script should be run at system boot time. It removes any lock files left over when news processes were killed at shutdown, and closes and executes any batches left over from NNTP connections that were terminated.

`newsrunning` This resides in `/usr/local/lib/news/bin/input`, and may be used to disable unbatching of incoming news. This may be desirable during work hours. You may turn off unbatching by invoking

```
/usr/local/lib/news/bin/input/newsrunning off
```

It is turned on by using `on` instead of `off`.

11.4 A Description of NNTP

Due to the different network transport used, NNTP provides for a vastly different approach to news exchange. NNTP stands for "Network News Transfer Protocol", and is not a particular software package, but an Internet Standard.[23] It is based on a stream-oriented connection — usually over TCP — between a client anywhere in the network, and a server on a host that keeps netnews on disk storage. The stream connection allows to interactively negotiate article transfer with nearly no turnaround delay, thus keeping the number of duplicate articles low. Together with the Internet's high transfer rates, this adds up to a news transport that surpasses the original UUCP networks by far. While some years ago it was not uncommon for an article to take two weeks or more before it arrived in the last

[22] Ever wondered how to get rid of that "Help! I can't get X11 to work with 0.97.2!!!" article?

[23] Formally specified in RFC 977.

corner of Usenet, this is now often less than two days; on the Internet itself, it is even within the range of minutes.

Various commands allow clients to retrieve, send and post articles. The difference between sending and posting is that the latter may involve articles with incomplete header information.[24] Article retrieval may be used by news transfer clients as well as newsreaders.

NNTP provides for an active and a passive way of news transfer, colloquially called "pushing" and "pulling". Pushing is basically the same as the Cnews ihave/sendme protocol. The client offers an article to the server through the "`IHAVE` <*msgid*>" command, and the server returns a response code that indicates whether it already has the article, or if it wants it. If so, the client sends the article, terminated by a single dot on a separate line.

Pushing news has the single disadvantage that it places a heavy load on the server system, since it has to search its history database for every single article.

The opposite technique is pulling news, in which the client requests a list of all (available) articles from a group that have arrived after a specified date. This query is performed by the `NEWNEWS` command. From the returned list of message ids, the client selects those articles it does not yet have, using the `ARTICLE` command for each of them in turn.

The problem with pulling news is that it needs tight control by the server over which groups and distributions it allows a client to request. For example, it has to make sure that no confidential material from newsgroups local to the site are sent to unauthorized clients.

There is also a number of convenience commands for newsreaders that permit to retrieve the article header and body separately, or even single header lines from a range of articles. This allows to keep all news on a central host, with all users on the (presumably local) network using NNTP-based client programs for reading and posting. (Alternatively, you could export the news directories to the clients, so that they may directly access the spool area. This eliminates the overhead introduced by the NNTP session layer.

An overall problem of NNTP is that it allows the knowledgeable to insert articles into the news stream with false sender specification. This is called *news faking*.[25] An extension to NNTP allows to require a user authenticication for certain commands.

There are a number of NNTP packages available. One of the more widely known is the NNTP daemon by Brian Barber and Phil Lapsley. It's most recent version is `nntpd-1.5.11`, which will be described below. You may either get the sources and compile it yourself, or use the `nntpd` from Fred van Kempen's `net-std` binary package, which as it as `/etc/in.nntpd`.

There is also a package called "Internet News", or INN for short. It provides both

[24] NNTP always adds at least one header field, which is `Nntp-Posting-Host:`. It contains the client's host name.

[25] The same problem exists with SMTP.

NNTP and UUCP-based news transport, and is said to be more suitable for large news hubs. It is currently at version 1.4. Since I don't know much about it, there won't be a section on this (yet). It is said to compile with some patches, which should be available from `sunsite.unc.edu`.

The `nntpd` package consists of a server and two clients for pulling and pushing news, respectively, as well as an `inews` replacement. They live in a Bnews environment, but with a little tweaking, they will be happy with a Cnews environment, too. We will have a look at these components and their interaction with Cnews below.

11.4.1 nntpd

The NNTP server is called `nntpd`, and may be compiled in two ways, depending on the expected load on the news system. It may be configured as either a standalone server that is started at system boot time from `/etc/rc.d/rc.net2`, or as a daemon managed by `inetd`.[26] In the latter case you have to have the following entry in `/etc/inetd.conf`:

```
nntp    stream  tcp nowait      news    /etc/in.nntpd    nntpd
```

If you configure `nntpd` as standalone, make sure there's no such line in `inetd.conf`, and if there is, comment it out. In both cases, you have to make sure there's the following line in `/etc/services`:

```
nntp    119/tcp   readnews untp    # Network News Transfer Protocol
```

11.4.2 NNTP Access.

Access to NNTP resources is governed by the file `nntp_access` in `/usr/local/lib/news`. Lines in the file describe the access rights granted to systems. They have the following format:

site `read|xfer|both|no post|no [!`*exceptgroups*`]`

If a client connects to the NNTP port, `nntpd` attempts to obtain the host's fully qualified domain name through reverse mapping (see section 2.3 on 39). Note that this returns the host's canonical name. The client's hostname and IP address are checked against the *site* field. Matches may be partial or exact. If an entry matches exactly, the entry's information

[26]The `nntpd` from Fred van Kempen's `net-std` package is configured for `inetd` service.

applies; if it is partial, it only applies if there is no other match following which is at least as good. *site* may be one of the following:

hostname — This is a fully qualified domain name of a host. If this matches the client's canonical hostname literally, the entry applies, and all following entries are ignored.

IP address — This is an IP address in dotted quad notation. If the client's IP address matches this, the entry applies, and all following entries are ignored.

domain name — This is a domain name, specified as `*.`*domain*. If the client's hostname matches the domain name, the entry matches.

network name — This is the name of a network as specified in `/etc/networks`. If the network number of the client's IP address matches the network number associated with the network name, the entry matches.

subnet name — This is the name of a subnet of the local network as specified in the `/etc/networks` file. The subnet mask is obtained by querying the network interfaces.[27] If the network number of the client's IP address matches the network number associated with the network name, the entry matches.

`default` — If the entry's name is `default`, it matches any client.

Entries with more general site specification should be specified earlier, because any matches by these will be overridden by later, more exact matches.

The tokens from the second and third field describe the access rights granted to the client. The first details the permissions the read news, and transmit news (using ihave/sendme). A value of `both` enables both, `no` denies access altogether. The third field grants the client posting rights (the difference between sending articles and postiong was described above). If the second field contains `no`, the third field is ignored.

Capitalization of these tokens causes `nntpd` to require authenticication for read, transfer, and post commands.

The fourth field is optional, and contains a comma-separated list of groups the client matched by this entry may is denied access to.

[27] This option is only enabled if `-DSUBNET` has been defined as compile-time option.

11.4.3 NNTP Authorization

As described above, nntpd allows for authorization procedures for certain clients. This is implemented by means of a new NNTP command named AUTHINFO. Using this command, the client may transmit a user name and a password to the NNTP server, which will validate these.

nntpd will check user name and password against the /etc/passwd database, and verify that the user belongs to the nntp group.

The NNTP clients included in the nntpd package expect to find the authorization data in a file called /etc/nntp.sys which contains triples of hostnames, user names, and passwords. These describe the user name and password to be used on a per-server basis. Note that the passwords are not encrypted, so that you must set the file's modes to 600 and make it owned by news.

A sample nntp.sys file might look like this:

```
news.fubar.edu nntpclnt Fr0b0zz
flap.jack.cookie.com intnews hurz
```

This tells the NNTP clients to identify themselves as user nntpclnt when, for example, connecting to news.fubar.edu, and to provide Fr0b0zz as password.

Note that the current implementation of NNTP authorization is only experimental, and has therefore not been implemented very portably. The result of this is that it only works with plain-style password databases; shadow passwords will not be recognized.

11.4.4 nntpd Interaction with Cnews

When receiving an article, nntpd has to deliver it to the news subsystem. Depending on whether it was received as a result of an IHAVE or POST command, the article is handed to rnews or inews, respectively. Instead of invoking rnews, you may also configure it (at compile time) to batch the incoming articles and move the resulting batches to /usr/spool/news/in.coming, where they are left for relaynews to pick them up at the next queue run.

When you compile nntpd, make sure it is compatible with your Cnews configuration. Beside the correct paths, this includes making sure they agree on the format of your history file. For example, some people link Cnews with a replacement for the dbm library, called dbz. For nntpd to work properly, it thus has to be linked with dbz, too.[28] A typical symptom of

[28] I don't know if this is true for newspak.

`nntp` and Cnews disagreeing on the database format are error messages in the system log that `nntpd` could not open it properly, as well as duplicate articles received via NNTP. A good test is to pick some articles from your spool area and batch them for NNTP transfer to your own site. When being offered them by `nntpxmit`, `nntpd` should reject them.

11.4.5 nntpxmit

The `nntpd` packages also contains sources for two NNTP clients for pushing and pulling, respectively. The first is performed by `nntpxmit`. It takes a list of articles and tries to transfer them to the remote host via ihave/sendme. The list of articles may be composed using a `sys` file entry with the `n` flag. (See section 11.3.3 above).

There is a `nntpsend` script in the `nntpd` sources which takes this list and feeds it to the pushing client. However, this script leans heavily on the Bnews side, so you might have to do some tweaking.

11.4.6 nntpxfer

`nntpxfer` is a client for pulling news. It is invoked as

`nntpxfer` *site* [*grouplist YYMMDD HHMMSS* [<*distlist*>]]

where *site* is the site you want to receive articles from. For each site, it keeps a file named `nntp.`*site* in the newsspool directory that contains the date of the last connect, the list of groups it is to request, and an optional list of distributions. If you specify these on the command line, this file will be ignored.

`grouplist` and `distlist` must be given as a comma-separated list without intermediate blanks. Negation of news hierarchies and distributions using an exclamation mark is permitted.

`nntpxfer` connects to *site* on the `nntp` port, and queries it for news having arrived in the listed groups after the last connection. The server responds by returning a list of message ids, which `nntpxfer` checks against the local `history` file. Articles not yet seen are requested. However, `nntpxfer` will never transfer more than 500 articles at once. If there are more articles than it can transfer, it will not update the time and date fields in the `nntp.`*site* file.

Like `nntpd`, `nntpxfer` may be configured to either batch incoming articles, or directly hand them to `rnews`.

Chapter 12

Newsreader Configuration

Newsreaders are intended to offer the user functionality that allows her to easily acess the functions of the news system, like posting articles, or skimming the contents of a newsgroup in a comfortable way. The quality of this interface is subject of endless flame wars.

There are a couple of newsreaders available which have been ported to LINUX. Below I will describe the basic setup for the three most popular ones, namely **tin**, **trn**, and **nn**.

One of the most effective newsreaders is

```
find /usr/spool/news -name '[0-9]*' -exec cat {} \; | more
```

This is the way Un⋆x die-hards read their news.

The majority of newsreaders, however, is much more sophisticated. They usually offer a full-screen interface with separate levels for displaying all groups the user has subscribed to, for displaying an overview of all articles in one group. and for individual articles.

At the newsgroup level, most newsreaders display a list of articles, showing their subject line, and the author. In big groups, it is impossible for the user to keep track of articles relating to each other, although it is possible to identify responses to earlier articles.

A response usually repeats the original article's subject, prepending it with "`Re: `". Additionally, the message id of the article it is a direct follow-up to may be given in the `References:` header line. Sorting articles by these two criteria generates small clusters (in fact, trees) of articles, which are called *threads*. One of the tasks in writing a newsreader is devising an efficient scheme of threading, because the time required for this is proportional to the square of the number of articles.

Here, we will not dig any further into how the user interfaces are built. All newsreaders currently available for LINUX have a good help function, so you ought to get along.

In the following, we will only deal with administrative tasks. Most of these relate to creation of the threads databases and accounting.

12.1 tin Configuration

The most versatile newsreader with respect to threading is `tin`. It was written by Iain Lea and is loosely modeled on an older newsreader named `tass`.[1] It does its threading when the user enters the newsgroup, and it is pretty fast at this unless you're doing this via NNTP.[2] You may improve this by cyclically updating your index file with the `-u` option, or by invoking it with the `-U` option.

Usually, `tin` dumps its threading databases in the user's home directory below `.tin/index`. This may however be costly in terms of resources, so that you should want to keep a single copy of them in a central location. This may be achieved by making `tin` setuid to `news`, for example, or some entirely unprivileged account.[3] `tin` will then keep all thread databases below `/usr/spool/news/.index`. For any file access or shell escape, it will reset its effective uid to the real uid of the user who invoked it.[4]

A better solution is to install the `tind` indexing daemon that runs as a daemon and regularly updates the index files. This daemon is however not included in any release of LINUX, so you would have to compile it yourself. If you are running a LAN with a central news server, you may even run `tind` on the server and have all clients retrieve the index files via NNTP. This, of course, requires an extension to NNTP. Patches for `nntpd` that implement this extension are included in the `tin` source.

The current version of `tin` included in SLS has no NNTP support compiled in. If however, you compile it yourself, or there should be a release with NNTP support one day, here's how: When invoked as `rtin` or the `-r` option, alternatively, `tin` tries to connect to the NNTP server specified in the file `/etc/nntpserver` or in the `NNTPSERVER` environment variable. The `nntpserver` file simply contains the server's name on a single line.

[1] Written by Rich Skrenta.

[2] On my 486DX50, it takes roughly 30 seconds to thread 1000 articles on `comp.os.linux`. Over NNTP to a loaded news server, this would be somewhere above 5 minutes.

[3] However, do *not* use `nobody` for this. As a rule, no files or commands whatsoever should be associated with this user.

[4] This is the reason why you will get ugly error messages when invoking it as super user. But then, you shouldn't work as `root`, anyway.

12.2 trn Configuration

trn is the successor to an older newsreader, too, namely rn (which means *read news*). The "t" in its name stands for "threaded". It was written by Wayne Davidson.

Unlike tin, trn has no provision for generating its threading database at run-time, but uses those prepared by a program called mthreads. It has to be invoked regularly from cron to update the index files. If you're receiving news during the night, you will customarily run it once in the morning, else you might want to do so more frequently.

Not running mthreads, however, doesn't mean you cannot access new articles, it only means you will have all those lovely "Is 386BSD better than Linix??!?!" articles scattered across your article selection menu, instead of a single thread you can wipe out in one keystroke (harharhar).

mthreads is invoked with a list of newsgroups you want it to thread. The list is made up in exactly the same fashion as the one in the sys file:

```
mthreads comp,rec,!rec.games.go
```

will thread all of comp and rec, except for rec.games.go (people who play Go don't need fancy threads). You perform threading of all groups found in your active file by specifying a group list of all, or none at all.

Sites that have very heavy traffic may also run mthreads in daemon mode. If it is started at boot time using the -d option, it puts itself in the background, and wakes up every 10 minutes to check if there are any newly-arrived articles. To do so, put the following line in your /etc/rc.d/rc.local script:

trn also needs to have old articles removed from the index files. By default, only articles whose number is below the low water mark will be removed from the index files.[5] Articles above this number who have been expired nevertheless (because the oldest article has been assigned an long expiry date by an Expires: header field) may be removed by giving mthreads the -e option. When running in daemon mode, it will put in such an "enhanced" expiry run once a day. By default, this is the first run after 12:30am.

[5]Note that Cnews doesn't update this low water mark automatically; you have to run upact to do so. Please refer to section 11.3.6.

12.3 `nn` Configuration

`nn`, written by Kim F. Storm, claims to be a newsreader whose ultimate goal is not to read news. It's name stands for "No News", and its motto is "No news is good news. `nn` is better."

To achieve this ambitious goal, `nn` comes along with a large assortment of maintenance tools that not only allow generation of threads, but also extensive checks on the consistency of these databases, accounting, gathering of usage statistics, and access restrictions. There is also an administration program called `nnadmin`, which allows you to perform these tasks interactively. It is very intuitive, hence we will not dwell on these aspects, and only deal with the generation of the index files.

The `nn` database manager is called `nnmaster`. It is usually run as a daemon, started from the `rc.local` script. It is invoked as

```
/usr/local/lib/nn/nnmaster -l -r -C
```

This enables threading for all newsgroups present in your `active` file.

Equivalently, you may invoke `nnmaster` periodically from `cron`, giving it a list of groups to act upon. This list is very similar to the subscription list in the `sys` file, except that it uses blanks instead of commas. Instead of the fake group name `all`, an empty argument of `""` should be used to denote all groups. A sample invocation is

```
/usr/local/lib/nn/nnmaster !rec.games.go rec comp
```

Note that the order is significant here: The leftmost group specification that matches always wins. Thus, if we had put `!rec.games.go` after `rec`, all articles from this group had been threaded nevertheless.

`nn` offers several methods to remove expired articles from its databases. The first is to update the database by scanning the news group directories and discarding the entries whose corresponding article is no longer available. This is the default operation obtained by invoking `nnmaster` with the `-E` option. It is reasonably fast unless you're doing this via NNTP. Method 2 behaves exactly like a default expiry run of `mthreads`, in that it only removes those entries that refer to articles whose number is below the low water mark in the `active` file. It may be enabled using the `-e` option. Finally, a third strategy is to discard the entire database and recollect all articles. This may be done by giving `-E3` to `nnmaster`. The list of groups to be expired is given by the `-F` option in the same fashion as above. However, if you have `nnmaster` running as daemon, you must kill it (using `-k`) before expiry can take place. However, since this kills the `nnmaster` daemon, you have to re-start it with the original options afterwards. Thus the proper command to run expire on all groups using method 1 is:

```
/usr/local/lib/nn/nnmaster -kF ""; /usr/local/lib/nn/nnmaster -lrC
```

There are many more flags that may be used to fine-tune the behavior of `nn`. If you are concerned about removing bad articles or digestifying article digests, read the `nnmaster` manual page.

`nnmaster` relies on a file named `GROUPS`, which is located in `/usr/local/lib/nn`. If it does not exist initally, it is created. For each newsgroup, it contains a line that begins with the group's name, optionally followed by a time stamp, and flags. You may edit these flags to enable certain behavior for the group in question, but you may not change the order in which the groups appear.[6] The flags allowed and their effects are detailed in the `nnmaster` manual page, too.

[6]This is because their order has to agree with that of the entries in the (binary) `MASTER` file.

Appendix A

A Null Printer Cable for PLIP

To make a Null Printer Cable for use with a PLIP connection, you need two 25-pin connectors (called DB-25) and some 12-conductor cable. The cable must be at most 15 meters long.

If you look at the connector, you should be able to read tiny numbers at the base of each pin, from 1 for the pin top left (if you hold the broader side up) to 25 for the pin bottom right. For the Null Printer cable, you have to connect the following pins of both connectors with each other:

D0	2—15	ERROR
D1	3—13	SLCT
D2	4—12	PAPOUT
D3	5—10	ACK
D4	6—11	BUSY
SLCTIN	17—17	SLCTIN
GROUND	25—25	GROUND
ERROR	15— 2	D0
SLCT	13— 3	D1
PAPOUT	12— 4	D2
ACK	10— 5	D3
BUSY	11— 6	D4

All remaining pins remain unconnected. If the cable is shielded, the shield should be connected to the DB-25's metallic shell on one end only.

Appendix B

The GNU General Public License

Printed below is the GNU General Public License (the *GPL* or *copyleft*), under which Linux is licensed. It is reproduced here to clear up some of the confusion about Linux's copyright status—Linux is *not* shareware, and it is *not* in the public domain. The bulk of the Linux kernel is copyright ©1993 by Linus Torvalds, and other software and parts of the kernel are copyrighted by their authors. Thus, Linux *is* copyrighted, however, you may redistribute it under the terms of the GPL printed below.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

B.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

B.2 Terms and Conditions for Copying, Distribution, and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this

License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or

distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

B.3 Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> ⟨*one line to give the program's name and a brief idea of what it does.*⟩ Copyright ©19yy ⟨*name of author*⟩
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author Gnomovision
comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.  This is
free software, and you are welcome to redistribute it under certain
conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> ⟨*signature of Ty Coon*⟩, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Annotated Bibliography

Books

Meta: Well, maybe this doesn't look very neat yet. Any suggestions for improvement welcome. Also, the list below is a very quick shot. So anybody who would like to amend it or "enhance" the abstracts is invited to do so.

[Hunt92] Craig Hunt: *TCP/IP Network Administration.* O'Reilly and Associates, 1992.

If the LINUX Network Administration Guide is not enough for for you, get this book. It deals with everything from obtaining an IP address to troubleshooting your network to security issues.

Its focus is on setting up TCP/IP, that is, interface configuration, the setup of routing, and name resolution. It includes a detailed description of the facilities offered by the routing daemons `routed` and `gated`, which supply dynamic routing.

It also describes the configuration of application programs and network daemons, such as `inetd`, the `r` commands, NIS, and NFS.

The appendix has a detailed reference of the `gated`, `named`, and a description of Berkeley's `sendmail` configuration.

[Stern92] Hal Stern: *Managing NIS and NFS.* O'Reilly and Associates, 19xx.

This is a companion book to Craig Hunt's "TCP/IP Network Administration" book. It covers the use of NIS, the Network Information System, and NFS, the Network File System, in extenso.

[OReilly89] Tim O'Reilly and Grace Todino, 10th ed: *Managing UUCP and Usenet.* O'Reilly and Associates, 1992. ISBN 0-93717593-5.

This is the standard book on UUCP networking. It covers Version 2 UUCP as well as BNU. It helps you setting up your UUCP node from the start, giving practical tips and solutions for many problems, like testing the connection, or writing good chat scripts. It also deals with more exotic topics, like how to set up a travelling UUCP node, or the subtleties present in different flavors of UUCP.

The second part of the book deals with Usenet and netnews software. It explains the configuration of both Bnews (version 2.11) and Cnews, and introduces you to netnews maintenance tasks.

[Tanen89] Andrew S. Tanenbaum: *Computer Networks.* Prentice Hall International, 1989. ISBN 0-13-166836-6[1].

This book gives you a very good insight into general networking issues. Using the OSI Reference Model, it explains the design issues of each layer, and the algorithms that may be used to achieve these. At each layer, the implementations of several networks, among them the Arpanet, are compared to each other.

The only drawback this book has is the abundance of abbreviations, which sometimes makes it hard to follow what the author says. But this is probably inherent to networking.

[Feit93] Sidnie Feit: *TCP/IP — Architecture, Protocol, and Implementation.* McGraw-Hill, 1993. ISBN 0-07-020346-6.

This is an all-singing-all-dancing book on TCP/IP networking. In fact, its more like an encyclopedia than like a book. It describes the major protocols, like IP, TCP, and UDP in great detail, and works its way up to the application programs. It explains everything quite exhaustively. For example, the chapter on `telnet` includes a tour of the `telnet` network terminal.

[1] The ISBN under which it is available in North America might me different.

This book is for people who want to know more about *how* TCP/IP and its applications work, but don't want to read RFCs.